Formal Verification of an Incremental Garbage Collector

Solange Coupet-Grimal *
Université de Provence, Marseille

Catherine Nouvet †
Ecole Nationale Supérieure des Télécommunications, Paris

January 27, 2003

Abstract

We specify an incremental garbage collection algorithm and we give a formal proof of its correctness carried out using the proof assistant Coq. The algorithm is represented as an infinite transition system and we establish safety and liveness properties. This work relies on the axiomatization of Linear Temporal Logic in the Calculus of Inductive Constructions given in [3] and based on a co-inductive representation of program executions. Although motivated by integrating the dynamic memory management to the Java Card platform, this study applies more generally to real-time embedded systems and to devices with virtually infinite memory.

Keywords Formal methods, type theory, temporal logics, co-induction, garbage collectors.

1 Introduction

The Java Card technology offers an interoperable multi-application based smart card platform and is being broadly deployed. The work presented in this paper is the result of a collaboration between the University of Provence, Gemplus, and Inria. It aims at extending the formal description of the Java Card platform taking into account the dynamic memory management, a feature that will be provided in the version 3.0 of Java Card. Indeed, the existing SmartCards suffer some limitations, notably the absence of garbage collector. All the created objects are persistent. Consequently, creating objects is done parsimoniously and this seriously curbs developing applications. For this reason, SmartCards designers are currently developing modular generic flexible operating systems, that are capable to evolve during the SmartCards life and to integrate memory management ([4]). Moreover, as SmartCards support

*Laboratoire d'Informatique Fondamentale de Marseille, CMI, 39 rue Joliot-Curie, F-13453, Marseille, France. Solange.Coupet@cmi.univ-mrs.fr.
†Nouvet@enst.fr

highly critical demands, formal specification of their functionalities as well as formal verification of security and safety properties are a crucial issue which gives rise to intensive international research.

In this framework, we present a formalization of a tricolor *mark-and-sweep* garbage collector (GC) in the Calculus of (Co)-Inductive Constructions [2, 15, 6]. A strong requirement for a GC to be embedded in SmartCards is to withstand the withdrawal of the card whatever the state of the process. Moreover in this interactive context (even though not real time, strictly speaking) the time of handling a command must be very short (less than one second) to be acceptable to the user (see [4]). Therefore, the algorithm we deal with has to be incremental. The specification we give is abstract enough to cover several actual implementations and we do not make any assumption on the set of memory cells. For these reasons, in spite of the fact it is motivated by the Java Card technology, this work also applies to other kind of real-time embedded systems and to devices with virtually infinite memory (such as internet).

The GC is modelled using state transition systems in Linear Temporal Logic (LTL) and we establish that the program is safe, in the sense that any freed cell is garbage. We also prove that provided the GC is called infinitely often, any inaccessible cell is eventually freed. This formal verification relies on a co-inductive axiomatization of LTL described in [3]. It also involves an axiomatization of finite sets that we compare with that given in [11] and that is used to count the memory cells of a given color in order to establish termination properties. All the proofs are carried out in the Coq proof assistant [18].

This paper is organized as follows. Section 2 is dedicated to describing and modelling of the garbage collection algorithm. Then, in section 3, we establish the safety property, and in section 4 the liveness property. Some aspects of our study are discussed in section 5 and related work in section 6. We conclude in section 7.

2 Specifying the algorithm

The encoding of the algorithm in the Calculus of Inductive Constructions relies on a general specification of infinite state systems given in [3]. Before presenting this encoding, let us describe informally the algorithm.

2.1 The Algorithm

Overview

Our algorithm is a variant of a 3-color mark and sweep algorithm first introduced in [5]. In this work, the user's program (the mutator) and the collector are considered to be running concurrently. In [12], the problem is reformulated assuming a single thread of control (no concurrency) and assuming that once the collector is invoked, it executes a whole phase of garbage collection, after which the mutator is resumed. A crucial question for a memory management system embedded on smart cards is to withstand the withdrawal of the card whatever the state of the process. So we have

given higher priority to this aspect to the detriment of garbage collecting efficiency. Here we present an incremental algorithm designed so that the collector can be interrupted at any time to let the mutator run.

In this kind of algorithm, the memory is modelled by a directed graph (the `heap`) with a special node called the root. The nodes represent the memory cells and there is an edge from a node `n` to a node `m`, if `n` contains a reference to `m`. In our work, we do not make any finiteness assumption on the set of cells. Our graph is then (possibly) infinite. However the nodes are colored with four colors instead of three: `white`, `grey`, `black`, and `free`. There is always a finite number of "marked" nodes, that is of white, grey, or black nodes. Only the finite subgraph of "marked" nodes will be under consideration except when the mutator needs a new cell which is then removed from the set of the free (unmarked) nodes. The aim of the tracing is to determine the current marked nodes that are accessible from the root. For that purpose, the subgraph is traversed from the root and the nodes are colored so that the following invariant property holds: a node is white when it has not yet been considered, it is grey when it has been considered but its children have not, and it is black when itself and its children have been considered. At the end of the marking, there will no longer remain any grey nodes, and all the reachable nodes will be black.

However, because of the incremental aspect of the garbage collector, the marking actions are interleaved with mutator actions that can add or remove edges in the graph. Consequently, these marking actions must be designed carefully in order to preserve the invariant.

Let us now examine the transitions of our memory management system.

The Transitions

We introduce three control states, namely `mutate`, `mark`, and `sweep`, that indicate that the current process is either the mutator, the marking process, or the collecting process.

The mutator can perform four actions. The first three do not change the control state `mutate`. These are:

- `add_edge` : it adds an edge between two accessible nodes `n` and `m`. Moreover, if `m` is white and `n` is black, then `m` becomes grey.
- `remove_edge` : it removes an edge between two accessible nodes.
- `alloc` : it picks out a free node `n`, removes the edges to its children if any, and adds `n` as a new black child of the root.

The mutator transitions should not deal with changing the colors. However, in the particular case of adding an edge from a black node to a white node, it is mandatory to change the color of the target and to grey it, for the invariant to be preserved. This concerns only one node and does not affect the mutator efficiency too much. In the same way, each new allocated cell is colored in black. Note that, as `remove_edge`

does not change the colors, certain nodes may become unreachable when being black or grey. Thus, these unreachable nodes will not be freed during the current cycle of garbage collecting. We will show that they will eventually be freed in the next cycle.

The fourth action, `gc_call`, makes a call to the marking phase of the garbage collector, and thus it turns the control to `mark`. It acts as follows:

- `gc_call` :
  - if all the nodes are black or free, then it re-initializes the marking phase
  - otherwise, if there exists a grey node, it blackens a grey node and greys its white children
  - otherwise, it frees a white node.

The re-initialization of the marking phase consists in greying the root and whitening the black nodes. Then all the marked nodes are white, except the root, since this re-initialization takes place when there are no grey nodes left.

Three transitions are enabled if the control is `mark`:

- `mark_node` : enabled only if there exists a grey node. It blackens a grey node and greys its white children. The control remains `mark`.
- `gc_free` : enabled only if there are no grey nodes and at least one white node. It frees a white node. The control becomes `sweep`.
- `gc_stop` : no action is performed. It simply returns to the mutator: the state becomes `mutate`.

The last two transitions are related to the collection phase characterized by the control `sweep`.

- `gc_free1` : enabled if there exists a white node. It frees a white node and the control remains `sweep`.
- `gc_end` : no action is performed. It simply returns to the mutator: the control becomes `mutate`.

One can notice that the marking process as well as the cell collection process can be interrupted at any time thanks to the transitions `gc_end` and `gc_stop`.

Initial State

To complete the description of the transition system, it remains to define the initial state. Initially:
– all states but the root are free,
– the root is black,
– there are no edges,
– the control is `mutate`.

The algorithm we have designed is a variant of that in [5]. The major difference lies in the fact that in [5] the number of nodes is finite and the nodes are examined in a specific order both during the marking phase and during the sweeping phase. This makes it possible to split the action that re-initializes the marking into finer-grained actions that can be interleaved with the mutator actions.

2.2 Specification in the Calculus of Constructions

We have given a specification that is general enough to cover several possible implementations. For example we never make it explicit which node precisely is picked out to be freed or blackened, nor the way in which the cells are represented.

Data types

Our formal development is parameterized by a set `node` representing the cells and by a particular node `rt` for the root. We set only one axiom: the decidability of the equality over the set `node`.

`Axiom eq_dec_node:` $(\forall$`n,m:node) (n=m)`$\vee$`(n`$\neq$`m)`

The constant `heap` denotes the type `node`$\rightarrow$`node`$\rightarrow$`bool` of the incidence matrices representing the graphs. We choose the relation to range over the set `bool` of booleans instead of `Prop` to be able to decide (in the logical meaning of the word) whether there exists an edge or not between two nodes. The constant `marking` stands for the type `node`$\rightarrow$`color`. Now, the states of the transition system under consideration are triples consisting of a `heap`, a `marking`, and a `control` state. In the following, if `s` is a state, `(hp s)`,`(mk s)`, and `(ctl s)` denote each of these three components.

Transitions as inductive relations

Each transition is specified as an inductive binary relation on the states. Let us consider, for example, the `gc_free1` transition.

Definition 1 The transition `gc_free1` is inductively defined as follows. Let `s` and `t` be two states. The formula `(gc_free1 s t)` holds if the following conditions are satisfied:

- both controls are equal to `sweep`,
- the heaps are identical,
- the markings differ only for a node `n` which is white in `s` and free in `t`.

Note that the existence of a transition `gc_free1` between two states `s` and `t` requires a node `n` to be `white` in `s` and `free` in `t`, but our specification does not dictate which particular node `n` among the white nodes must be picked out to be freed. This is left to the implementor.

Transitions vs labelled transitions

A label is associated with each transition, say `fr` for `gc_free`. Let `label` be the set of the nine labels. We inductively define a relation

`R :label→state →state→Prop`

and write $\texttt{s} \xrightarrow{l} \texttt{t}$ for `(R l s t)`. The relation is defined in such a way that $\texttt{s} \xrightarrow{fr} \texttt{t}$ stands for `(gc_free s t)`. Now $\texttt{s} \xrightarrow{step} \texttt{t}$ means that `t` is derived from `s` by any of the nine transitions. Thus, we obtain an instance of the abstract model for labelled transition systems, introduced in [3].

Reachability, accessibility

Finally, we define as follows the relations `reachable` and `accessible` over the nodes.

Definition 2 Let `h` be a heap and `n` be a node. The relation "to be reachable" in `h` from `n` is defined inductively by the two following conditions:
- in `h`, `n` is reachable from itself
- every node is reachable from `n` provided it has a father in `h` that is also reachable from `n`.

Definition 3 A node is `accessible` in a heap `h` if and only if it is reachable from the root `rt`.

When it is clear from the context, we leave the heap implicit in these accessibility notions.

We can now tackle the verification of the algorithm, starting with the establishment of the safety property.

3 The Safety Property

The safety property states that nothing bad will ever happen, and more precisely, that the garbage collector only collects unreachable objects. One must prove that the formula *"any free node is inaccessible"* continuously holds on all the *runs*.

Lemma 4 (garbage collector safety) The formula defined on all states `s` by "in `s`, all free nodes are inaccessible" is safe.

The lemma is obtained as a corollary of the following stronger result:

Lemma 5 Let `I` be the conjunction of the five properties below.

1. `RtGreyOrBlack`: The root is grey or black
2. `SweepNoGrey`: If the control is `sweep`, there are no grey nodes
3. `NoEdgeBlackWhite`: There is no edge from a black node to a white node
4. `AccNotFree`: If a node is accessible, then it is not free
5. `NoGreyAccBlack`: If there are no grey nodes, then all accessible nodes are black

`I` is a safe property.

In this statement, an identifier in teletype characters is associated with each of the five properties. In the sequel, it denotes either the state formula or the corresponding lifted stream formula, depending on the context. Let us give a sketch of the proof.

Proof The proof follows from theorem 2 (*safety*) in [3]. One can easily be convinced that I holds on the initial state. As far as invariance is concerned, the first three properties can be treated independently from each other. Their proofs are performed by case analysis on the nine possible transitions and do not present any theoretical difficulty.

However, we cannot prove directly that statements 4 and 5 are maintained by all transitions. One establishes the following two weaker results.

Lemma 6 Let `s` and `t` be two states such that $\texttt{s} \xrightarrow{step} \texttt{t}$. If `s` satisfies I, then `t` satisfies property `AccNotFree` (condition 4).

Lemma 7 Let `s` and `t` be two states such that $\texttt{s} \xrightarrow{step} \texttt{t}$. If `s` satisfies I, then `t` satisfies property `NoGreyAccBlack` (condition 5).

The proofs of these lemmas require the following intermediate result which is proved by case analysis on all the possible transitions.

Lemma 8 Let `s` and `t` be two states such that $\texttt{s} \xrightarrow{step} \texttt{t}$. Every node which is accessible in state `t` is also accessible in state `s` (except for the allocated cell in the case of the `alloc` transition).

Proof of lemma 6. Let `s` and `t` be two states, let us consider a transition such that $\texttt{s} \xrightarrow{l} \texttt{t}$ and let us assume that `s` satisfies I. If a free node in `t` was already free in `s`, it was not accessible in `s` by hypothesis. Therefore, from lemma 8, it is not accessible in `t` (in the exceptional case of an alloc transition, the allocated node is not `free` in `t`). So a problem is raised only when a white node `n` in state `s` becomes free in state `t`. This happens in three cases:

- for `gc_call`, when there are no grey nodes in `s`
- for `gc_free`, which is enabled only if there are no grey nodes in `s`
- for `gc_free1`, which is enabled only if the control in `s` is `sweep`. But then, by the hypothesis that `s` satisfies I, and specially condition 2 in I, one can conclude that there are no grey nodes in `s`.

In the three cases, there are no grey nodes in `s` and as `s` satisfies `NoGreyAccBlack` (condition 5), all accessible nodes in `s` are black. Thus `n` is not accessible in `s` and so, by lemma 8, it is not accessible in `t`.

Proof of lemma 7. Let `s` and `t` be two states, let us consider a transition such that $\texttt{s} \xrightarrow{l} \texttt{t}$ and let us assume that `s` satisfies I. When the transition $\xrightarrow{l}$ does not blacken any grey node in `s`, and if there are no grey nodes in `t`, then there are no grey nodes

in s. So in s, all accessible nodes are black. By lemma 8, an accessible node (say n) in t is accessible in s (except if it is a newly allocated cell, in which case n is black in t). Then n is black in s and it remains black in t. As a matter of fact, the only transition that can change the color of a black node, is gc_call, when re-initializing the marking. But in that case, the root is grey in t, which is inconsistent with the fact that there are no grey nodes in this state.

The transitions that blacken a grey node (that is mark_node or, in certain cases, gc_call) are more difficult. The proof requires the hypothesis that s satisfies conditions 1, 2, 3, and 4 and is performed by induction on the accessibility of the node n.

The whole proof consists of approximatively one hundred and fifty lemmas.

4 The Liveness Property

The proof of the liveness property is awkward. We will try to emphasize the main points as clearly as possible.

Classically, the termination proof relies on the fact that a "measure" on the set of states strictly decreases when one applies a transition for which a fairness requirement is made. The "measure" we have chosen here is the sum of the number of the white nodes and the number of the grey nodes in a state. So we have to do meticulous work to prove that there is always a finite number of white nodes, of grey nodes, and of black nodes and to calculate their number accurately in each state of a run. For this reason we were led to axiomatize some notions of finite subset cardinality adapted to our problem.

4.1 Axiomatizing Finite Sets

In [11], G.Huet and G.Kahn define the cardinal of the finite subsets of a set A. The subsets are represented by their characteristic predicates. Their axiomatization is based on a relation cardinal of type (A→Prop)→nat→Prop, inductively defined as follows:

1. (cardinal ∅ 0)
2. Let Q be a subset of A and a_0 an element of A such that $a_0 \notin Q$. Then for all natural numbers nb, (cardinal Q nb)→(cardinal $Q \cup \{a_0\}$ (nb+1)).

In the first condition, ∅ denotes the empty subset of A and is axiomatized as an inductive type with no constructor. The union of two subsets is also inductively defined.

Our definition is a bit more concrete in the sense that the characteristic predicate is decomposed into a computational part and a logical part. More precisely, let us consider two sets A and B and let P be a predicate on B.

For all programs M:A→B the cardinal of the subset {a∈A| (P (M a))} is specified by a relation card of type :(B→Prop)→(A→B)→nat→Prop. This relation is defined inductively by the two conditions below:

1. (∀a∈A ¬(P (M a)))→(card P M 0)
2. Let M_1 and M_2 be two functions of type A→B and let a_0 be an element of type A such that (P (M_2 a_0)) and ¬(P (M_1 a_0)). Moreover if M_1 and M_2 coincide over A-{a_0}, then for all natural numbers nb :
(card P M_1 nb)→(card P M_2 (nb+1))

An extra hypothesis (∃b:B)¬(P b) is assumed for the relation to be non empty.

Being less abstract and closer to the informal notion of counting objects some attributes of which satisfy a certain predicate, our definition, although less elegant, is easier to handle in the framework of our study. However, we can prove that these two approaches are equivalent provided that we assume the extensionality axiom on the subsets of A (and this is the drawback of the abstract point of view). This axiom is set as follows:

For all subsets Q_1 *and* Q_2 *of* A, $(Q_1 \subseteq Q_2) \wedge (Q_2 \subseteq Q_1) \rightarrow Q_1 = Q_2$

where = denotes the Leibniz equality.

Having set this axiom, we can then prove the next two results (where P∘M stands for the composition of P and M):

- (∀nb:nat) (card P M nb) ↔ (cardinal (P∘M) nb)
- For all subset Q of A,(∀nb:nat)(cardinal Q nb) ↔ (card Q λx.x nb)

The second proposition is just a corollary of the first one.

The proofs of both implications in the first result are performed by induction. The extensionality hypothesis is needed quite naturally to prove the equivalence. As a matter of fact, let us consider the example of the null cardinality. With our definition it corresponds to all the predicates (P∘M) that are not satisfied anywhere. Identifying all these terms with the empty set considered in [11] requires the extensionality condition. This axiom is also applied during the induction step of the reciprocal implication, to prove that a subset Q of A is of the form P∘M.

We establish some classical results such as the unicity of the cardinal as well as the following inclusion related proposition:

Lemma 9 Let P and Q be two predicates over B and let M and N be two functions of type A→B. Assuming that (∀a:A) (Q (N a))→ (P (M a)), for all natural numbers nb such that (card P M nb) there exists a natural number nb' such that nb'≤nb and (card Q N nb').

We now have at our disposal the logical tools for counting the marked nodes.
In [11], G.Huet and G.Kahn define the cardinal of the finite subsets of a set A. The subsets are represented by their characteristic predicates. Their axiomatization is based on a relation cardinal of type (A→Prop)→nat→Prop, inductively defined as follows:

1. (cardinal ∅ 0)
2. Let Q be a subset of A and a_0 an element of A such that $a_0 \notin$Q. Then for all natural numbers nb, (cardinal Q nb)→(cardinal Q∪{a_0} (nb+1)).

In the first condition, ∅ denotes the empty subset of A and is axiomatized as an inductive type with no constructor. The union of two subsets is also inductively defined.

Our definition is a bit more concrete in the sense that the characteristic predicate is decomposed into a computational part and a logical part. More precisely, let us consider two sets A and B and let P be a predicate on B.

For all programs M:A→B the cardinal of the subset {a∈A| (P (M a))} is specified by a relation card of type :(B→Prop)→(A→B)→nat→Prop. This relation is defined inductively by the two conditions below:

1. (∀a∈A ¬(P (M a)))→(card P M 0)
2. Let M_1 and M_2 be two functions of type A→B and let a_0 be an element of type A such that (P (M_2 a_0)) and ¬(P (M_1 a_0)). Moreover if M_1 and M_2 coincide over A-{a_0}, then for all natural numbers nb :
(card P M_1 nb)→(card P M_2 (nb+1))

An extra hypothesis (∃b:B)¬(P b) is assumed for the relation to be non empty.

Being less abstract and closer to the informal notion of counting objects some attributes of which satisfy a certain predicate, our definition, although less elegant, is easier to handle in the framework of our study. However, we can prove that these two approaches are equivalent provided that we assume the extensionality axiom on the subsets of A (and this is the drawback of the abstract point of view). This axiom is set as follows:

For all subsets Q_1 *and* Q_2 *of* A, $(Q_1 \subseteq Q_2) \wedge (Q_2 \subseteq Q_1) \rightarrow Q_1 = Q_2$

where = denotes the Leibniz equality.

Having set this axiom, we can then prove the next two results (where P∘M stands for the composition of P and M):

- (∀nb:nat) (card P M nb) ↔ (cardinal (P∘M) nb)
- For all subset Q of A,(∀nb:nat)(cardinal Q nb) ↔ (card Q λx.x nb)

The second proposition is just a corollary of the first one.

The proofs of both implications in the first result are performed by induction. The extensionality hypothesis is needed quite naturally to prove the equivalence. As a matter of fact, let us consider the example of the null cardinality. With our definition it corresponds to all the predicates (P∘M) that are not satisfied anywhere. Identifying all these terms with the empty set considered in [11] requires the extensionality condition. This axiom is also applied during the induction step of the reciprocal implication, to prove that a subset Q of A is of the form P∘M.

We establish some classical results such as the unicity of the cardinal as well as the following inclusion related proposition:

Lemma 10 Let P and Q be two predicates over B and let M and N be two functions of type A→B. Assuming that (∀a:A) (Q (N a))→ (P (M a)), for all natural numbers nb such that (card P M nb) there exists a natural number nb' such that nb'≤nb and (card Q N nb').

We now have at our disposal the logical tools for counting the marked nodes.

4.2 Counting the marked nodes

We define a relation card_color in such a way that (card_color c m nb) expresses that there are nb nodes of color c in marking m. We simply use the definition of card in section 4.1, in which P is instantiated by λc' :color. (c'= c) and M by m. A relation measure is inductively defined so that (measure m nb) holds if and only if, for the marking m, the sum of the white nodes and the grey nodes is equal to nb.

Then we prove 4x9=36 lemmas that express how the numbers of grey, white, and black nodes of a state evolve as well as the measure of its marking, when each of the nine transitions is applied.

Some of the proofs are tedious and we do not go into details. Let us just summarize the results.

Lemma 11 Let us consider a state, the marking of which makes w nodes white, g nodes grey, b nodes black, let m = w + g and let us apply a transition.

- **White nodes**
  - w is decremented by gc_free and gc_free1
  - w is left unchanged or is decremented by add_edge
  - w takes the value b-1 when the colors are re-initialized
  - w takes the value w-nb when blackening a node and greying its nb white children
  - w is left unchanged by remove_edge, alloc, gc_stop, gc_end.

- **Grey nodes**
  - g is left unchanged or is incremented by add_edge
  - g takes the value 1 when the colors are re_initialized
  - g takes the value g+nb-1 when blackening a node and greying its nb white

children
– g is left unchanged by remove_edge, alloc, gc_stop, gc_end, gc_free, gc_free1.

- **Black nodes**
  – b is incremented by alloc and by a marking transition that blackens a node and greys its nb white children
  – b takes the value 0 when the colors are re_initialized
  – b is left unchanged by add_edge, remove_edge, gc_stop, gc_end, gc_free, gc_free1.

- **Measure**
  – m is decremented by gc_free, gc_free1, mark_node and, if m≠0, by gc_call
  – m is left unchanged by the rest of the transitions.

Fairness These statements about the measure make it clear why we require a fairness condition for the labels corresponding to the transitions gc_free, gc_free1, gc_call, and mark_node: they are the transitions that decrease the measure. We then introduce a predicate fair that is satisfied exactly by these four transitions.

Lemma 12 (Existence of the measure) The finiteness of the sets of the nodes of color c, c∈{grey, white, black} is a *safe* property. Therefore, the existence of the measure is a safe property.

This is straightforward from lemma 11.

4.3 Termination

The aim of this paragraph is to prove that, for all the *runs* satisfying the justice process condition fairstr defined in [3] (*a stream σ is a fair stream iff infinitely often on σ, if a fair transition is enabled, then a fair transition is taken*) the measure, and therefore the number of white nodes and the number of grey nodes, becomes null infinitely often. For that, we have to prove some intermediate lemmas.

Let us recall the invariant stated in condition 2 of lemma 5: in a state s in which the control is sweep, there are no grey nodes. This is denoted by: (SweepNoGrey s).

Lemma 13 A fair transition is enabled on every state that satisfies SweepNoGrey and where the measure exists and is strictly positive.

The proof is performed by case analysis on the control of the state.

Lemma 14 Let σ be a stream such that (fairstr σ) and such that SweepNoGrey continuously holds on σ. Then σ satisfies (infinitely_often P), where the stream formula P is defined by:
$\lambda\tau$:stream.((∃nb:nat) (measure (mk τ_0) nb)∧nb≠0) → ($\tau_0 \xrightarrow{fair} \tau_1$)

Proof We apply a general temporal lemma that results straightforwardly from the monotonicity of the operator `infinitely_often` (see [3], section 4.2) and that states that for all stream formulas `P`, `Q`, and `R`:
`((∀σ:stream)(□R σ) → (Q σ) → (P σ)) →`
`((∀σ:stream)(□R σ)→(infinitely_often Q σ)→(infinitely_often P σ))`
In this case:
– `R` will be instantiated by `SweepNoGrey`,
– `Q` by *"if a fair transition is enabled on the head of the stream, then a fair transition is taken"* (from the definition of `fairstr`),
– `P` by the formula `P` defined in the statement.

Proving lemma 14 now simply amounts to proving that `(P σ)` holds for all σ satisfying the following conditions :

– `(□SweepNoGrey σ)`
– if a fair transition is enabled in σ_0, then a fair transition is taken.

This follows immediately from lemma 13.

The next lemma aims at establishing the conditions under which termination theorem (theorem 12) in [3] can be applied.

Lemma 15 Let σ be a trace satisfying $\Diamond$`P`. If in the state σ_0 the measure exists and has a positive value `v`, it remains positive until eventually it takes a value less than `v` or it becomes null.

The predicate `P` under consideration is that defined in lemma 14. The proof is performed by induction on the term `(◇P σ)` and relies on the fact that if a fair transition is taken in a state where the measure is positive, the measure strictly decreases (lemma 11).

Lemma 16 Let σ be a trace such that `(fairstr σ)` and such that the invariant property `I` of lemma 5 continuously holds on σ. Then, eventually the measure becomes null and therefore so do the number of grey nodes and the number of white nodes.

Proof As σ always satisfies `SweepNoGrey`, one can apply lemma 14 and thus lemma 15. Then, the result follows from theorem 12 (termination theorem) in [3] and relies on the well-foundedness of the strict ordering on the natural numbers.

Lemma 17 Let σ be a run such that `(fairstr σ)`. Then, infinitely often the measure becomes null and therefore so does the number of grey nodes and white nodes.

Proof The proof is done by applying lemma 5 in [3]. Then, the result follows immediately from the previous lemma and the fact that `I` is a *safe* property by lemma 5.

4.4 Liveness

We introduce ten families of state formulas indexed by a node `n`. They are defined on a state `s` as follows. :

(a n s) : `n` is not accessible

(b n s) : `n` is free or
- `n` is not free and
- `n` is not accessible and
- the number of grey nodes is null

(c n s) : – `n` is white and
- `n` is not accessible and
- the number of grey nodes is null

(d n s) : `n` is free

(e n s) : – `n` is black and
- `n` is not accessible and
- the number of grey nodes is null
- the number of white nodes is not null

(f n s) : – `n` is black and
- `n` is not accessible and
- the number of grey nodes is null

(g n s) : – `n` is black and
- `n` is not accessible and
- the measure is null

(h n s) : – `n` is white and
- `n` is not accessible and
- the root is grey, and
- all the nodes, except the root, are white or free.

(k n s) : – `n` is white and
- `n` is not accessible and
- and all the ancestors of `n` are free or white,
- the number of grey nodes is not null.

(k' n s) : – `n` is white and
- `n` is not accessible and
- and all the ancestors of `n` are free or white
- the number of grey nodes is null.

Let us denote by the corresponding capital letter the lifted stream formulas. For instance, `(A n` σ`)` stands for `(a n` σ_0`)`.

Lemma 18 summarizes the intermediate steps necessary to establish the final liveness result.

Lemma 18 Let `n` be a node and σ a *trace* such that (`□I` σ) where I is the invariant conjunction introduced in lemma 5.

1. If σ is such that eventually the number of grey nodes becomes null, then `(A n` σ`)`$\rightarrow$`((A n)`$\mathcal{U}$`(B n)` σ`)`.
2. If σ is such that eventually the number of white nodes becomes null, then `(C n` σ`)`$\rightarrow$`((C n)`$\mathcal{U}$`(D n)` σ`)`.
3. If σ is such that eventually the number of white nodes becomes null, then `(C n` σ`)`$\rightarrow$ `(`$\Diamond$`(D n)` σ`)`.
4. If σ is such that the number of white nodes is always finite and such that it eventually becomes null, then `(E n` σ`)`$\rightarrow$`((E n)`$\mathcal{U}$`(G n)` σ`)`.
5. If σ is such that the number of white nodes is always finite and such that it eventually becomes null, then `(F n` σ`)`$\rightarrow$`((F n)`$\mathcal{U}$`(G n)` σ`)`.
6. If σ is such that infinitely often a fair step is taken then `(G n` σ`)`$\rightarrow$`((G n)`$\mathcal{U}$`(H n)` σ`)`.
7. If σ is such that the number of grey nodes is always finite and such that it eventually becomes null, then `(K n` σ`)`$\rightarrow$`((K n)`$\mathcal{U}$`(K' n)` σ`)`.
8. If σ is such that the number of grey nodes is always finite and such that it eventually becomes null, then `(K n` σ`)`$\rightarrow$`((K n)`$\mathcal{U}$`(C n)` σ`)`.
9. If σ is such that the number of grey nodes eventually becomes null and infinitely often the number of white nodes becomes null, then `(K n` σ`)`$\rightarrow$`(`$\Diamond$`(D n)` σ`)`.

Proof

- **Statement 1**.

  We first prove that for all states `s` and `t` such that `s`$\xrightarrow{step}$`t`, any node that is not free and not accessible in `s` remains not accessible in `t`. The proof does not follow directly from lemma 8 since `alloc` transition needs a special handling. A case analysis is necessary. Nevertheless, the other eight cases follow directly from those considered in lemma 8.

  The proof of statement 1 is performed by induction on the hypothesis that, eventually, the number of grey nodes becomes null.

  Base case. In this case, in state σ_0, the number of grey nodes is null. Obviously `(b n` σ_0`)` holds and then the goal is proved.

  Induction Step. Assuming that the statement holds for a trace σ, we have to prove that it holds for a trace `(cons s` σ`)`. The proof is done by considering the cases in which `n` is free or not free in `s`. In the former case, we have `(b n s)` and therefore, `((A n)`$\mathcal{U}$`(B n) (cons s` σ`))`holds. In the latter, since `(cons s` σ`)` is a trace:

– either σ_0 = s, and then since n is not accessible in s by hypothesis, neither it is in σ_0, and then, (a n σ_0) holds.
– or s $\xrightarrow{step}$ σ_0, and then we know from the preliminary result that n is not accessible in σ_0, and then, (a n σ_0) holds .
In both cases the proof is achieved by applying the induction hypothesis on σ. We obtain ((A n)$\mathcal{U}$(B n) σ), and thus we deduce :
((A n)$\mathcal{U}$(B n) (cons s σ)) by the hypothesis (A n (cons s σ)).

- **Statement 2**.

  We first prove by case analysis on all the possible transitions that for all nodes n, for all states s and t such that s$\xrightarrow{step}$t, if s satisfies the predicates SeewpNoGrey and NoGreyAccBlack defined in lemma 5, then:
  (c n s)$\rightarrow$(c n t)$\vee$(d n t).

  Then, we prove that all traces σ on which SweepNoGrey and NoGreyAccBlack continuously hold, satisfy statement 2. This is done by induction on the hypothesis stating that eventually the number of white nodes becomes null.

  Base case. We have to prove (C n σ)$\rightarrow$((C n)$\mathcal{U}$(D n) σ) under the hypothesis that there are no white nodes in the head of σ. This hypothesis and (C n σ) are contradictory.

  Induction step. Assuming that the statement holds for a trace σ, we have to prove that it holds for a trace (cons s σ). The proof is done by considering separately the cases in which n is free or not free in σ_0. In the former case, (D n σ) holds and since (C n (cons s σ)) holds by hypothesis, we have proved ((C n)$\mathcal{U}$(D n) (cons s σ)). In the latter, since (cons s σ) is a trace:
  – either σ_0 = s. In this case, as (c n σ_0) holds, we can conclude by applying the induction hypothesis to σ
  – or s $\xrightarrow{step}$ σ_0, and then we know that (c n σ_0)$\vee$(d n σ_0) holds. If the condition (c n σ_0)is satisfied, the proof is achieved by applying the induction hypothesis. If (d n σ_0) holds, we can deduce ((C n)$\mathcal{U}$(D n) (cons s σ)).

- **Statement 3**. This is an immediate corollary of statement 2.

- **Statement 4**.

  We first prove by case analysis on all the possible transitions that for all nodes n, for all states s and t such that s$\xrightarrow{step}$t, if s satisfies the predicates SweepNoGrey and NoGreyAccBlack then: (e n s)$\rightarrow$(f n t).

  We now prove that all the traces σ on which SweepNoGrey and NoGreyAccBlack continuously hold, satisfy statement 4. This is done by induction on the hypothesis stating that eventually the number of white nodes becomes null.

  Base case. We have to prove that (E n σ)$\rightarrow$((E n)$\mathcal{U}$(G n) σ) under the hypothesis that there are no white nodes in the head of σ. This hypothesis and

(E n σ) are contradictory.

Induction step. Assuming that the statement holds for a trace σ, we have to prove that it holds for a trace (cons s σ). The proof is done by considering the cases in which the number w of white nodes in s is null or not. In the former case, (G n (cons s σ)) holds and thus ((E n)$\mathcal{U}$(G n) (cons s σ)) also holds. In the latter, since (cons s σ) is a trace:
– either σ_0 = s, then by the hypothesis (E n (cons s σ)), we deduce (E n σ), and we can conclude by applying the induction hypothesis to σ.
– or s $\xrightarrow{step}$ σ_0, and then we know that (f n σ_0) holds. If the number of white nodes of σ_0 is not zero, then (E n σ) holds and the proof is achieved by applying the induction hypothesis. If it is null, from (G n σ) we can deduce ((E n)$\mathcal{U}$(G n) (cons s σ)).

- **Statement 5**. It follows straightforwardly from statement 4 by noticing that (F n σ) is equivalent to (E n σ)$\vee$(G n σ).

- **Statement 6**.

  We prove the following preliminary results.
  – (a): For all nodes n, for all states s and t such that s$\xrightarrow{step}$t, if s satisfies the predicate NoGreyAccBlack then: (g n s)$\rightarrow$(g n s)$\vee$(h n t).
  – (b): Let s and t be two states and assume that the measure of s equals 0. Then, if s $\xrightarrow{fair}$t, the marking of t is obtained by re-initializing the marking of s.
  – (c): Let s and t be two states such that s $\xrightarrow{step}$ t and such that the marking of t is obtained by re-initializing the marking of s. Then:
  (g n s)$\rightarrow$(h n t).

  The proof of statement 6 is done by induction on the fact that eventually, a fairstep is taken.

  Base case. In this case, we have σ_0 $\xrightarrow{fair}$ σ_1. Moreover, as (g n σ_0)holds, the measure in σ_0 is null. It follows from (b) that the marking of σ_1 is obtained by re-initializing the marking of σ_0 and thus, (h n σ_1) holds from (c). Thus we have proved ((G n)$\mathcal{U}$(H n) σ).

  Induction step. Assuming that the statement holds for a trace σ, we have to prove that it holds for a trace (cons s σ). As (cons s σ) is a trace, either s= σ_0 or s $\xrightarrow{step}$ σ_0. In the first case, we conclude by applying the induction hypothesis, since we know that (g n σ_0) is satisfied. In the second case, from (a) either (g n σ_0) holds and we apply the induction hypothesis, or (h n σ_0) holds and as by hypothesis we have (g n s), we deduce ((G n)$\mathcal{U}$(H n)(cons s σ)).

- **Statement 7**. We define a predicate p by :
  (p n s) : – n is white and
  – n is not accessible and

– and all the ancestors of n are free or white.

Let us notice that, if gr denotes the number of grey nodes in state s:
– (k n s)↔(p n s)∧(gr≠0)
– (k' n s)↔(p n s)∧(gr=0).

We first prove by case analysis on all the possible transitions that for all nodes n, for all states s and t such that $s \xrightarrow{step} t$, if s satisfies the predicates SweepNoGrey and NoGreyAccBlack: (k n s)→(p n t).

Now, the proof proceeds by induction on the fact that the number of grey nodes eventually becomes null.

Base case. In this case, the number of grey nodes on σ_0 is null and by hypothesis (k n σ_0) holds. These statements are contradictory.

Induction step. Assuming that the statement holds for a trace σ, we have to prove that it holds for a trace (cons s σ). Since (cons s σ) is a trace:
– either σ_0 = s, then we have (K n σ), and we can conclude by applying the induction hypothesis to σ
– or $s \xrightarrow{step} \sigma_0$, and then (p n σ_0) holds. Let gr be the number of grey nodes in σ_0. If gr is null, then (k' n σ_0) holds and thus (cons s σ) satisfies (K n)$\mathcal{U}$(K' n). If gr is not null, we can conclude by applying the induction hypothesis.

- **Statement 8**. It follows from the fact that obviously, for all streams σ, (K' n σ)→(C n σ).
- **Statement 9**. By applying statement 8 we deduce that σ satisfies the predicate (K n)$\mathcal{U}$(C n). The proof is then performed by induction on the term ((K n)$\mathcal{U}$(C n) σ) and by applying statement 3.

The final liveness theorem states that provided the set of fair transitions is not forgotten forever, every node that becomes inaccessible will eventually be freed.

Theorem 19 (Liveness) Let σ be a run such that infinitely often a fair transition is taken. For all nodes n the following property always holds: if n becomes inaccessible, then eventually it will be freed.

Proof By lemma 5 in [3], it is sufficient to prove that for all the traces σ satisfying the fairness condition and all the properties that have been proved to be safe, if n is not accessible in σ_0, then it will eventually be freed.

By statement 1 in lemma 18, assuming that n is not accessible, we have a term h1:((A n)$\mathcal{U}$(B n) σ). We then do an induction on h1. The induction step is immediate. Let us consider the base case.

Base case 1. We have to prove that under the hypothesis `h:(B n` σ`)` the node `n` will be eventually freed. By `h`, in the state σ_0 either `n` is free, and then the goal is proved, or `n` is not free and it is not accessible and the number of grey nodes is null. Therefore, `n` is white or black.

- If `n` is white, `(C n` σ`)` holds. The result follows from the statement 3 of lemma 18.
- If `n` is black, then `(F n` σ`)` holds. By statement 5 in lemma 18, we deduce `h2:((F n)`$\mathcal{U}$`(G n)` σ`)`.
  Let us do an induction on `h2`. The induction step is easy.

  Base case 2. We have to prove that under the hypothesis `h': (G n` σ`)`, eventually the node `n` will be freed. Applying statement 6 in lemma 18, we deduce `h3: ((G n)`$\mathcal{U}$`(H n)` σ`)`.
  We do an induction on `h3`. The induction step is easy.

  Base case 3. We have to prove that under the hypothesis `h'': (H n` σ`)`, eventually the node `n` will be freed. As `(H n` σ`)` implies `(K n` σ`)`, we can deduce the result from the statement 9 of lemma 18.

This completes the garbage collector certification process.

5 Discussion

Fairness

As far as fairness is concerned, all the statements in lemma 18 have been proved under the weak fairness condition (called process justice in [13]) expressed by the predicate `fairstr`, except statement 6, for which strong fairness is required. As a matter of fact, a run that loops forever on a state whose measure is null and where the control is `mark` or `sweep`, meets the weak fairness condition since in such a state, no fair transition is enabled. However, it does not satisfy the liveness property, since in such a state, certain inaccessible nodes could be black. Strong fairness condition is then required to ensure that at the end of a cycle of garbage collection, a new cycle will be processed. Moreover, in the case of our algorithm, strong fairness is equivalent to requiring transition `gc_call` to be taken infinitely often. Indeed, we have proved formally that `gc_call` is the only fair transition that can be taken when the measure is null. Moreover all the non fair transitions leave the measure unchanged. As the measure becomes null infinitely often and as it remains null until eventually `gc_call` is taken, `gc_call` is taken infinitely often. One could also notice that once the gc is called, the other three transitions can occur at most a finite number of times, before the control returns to the mutator.

Axioms

The whole development contains exactly 3 axioms:

- the decidability of the equality on the set of nodes,
- the decidability of the equality on the set `A` on which we define the notion of cardinal (see section 4.1),
- the decidability of the fact that there exists a node having the color `c` in `m` for all markings `m` and for all colors `c`.

The first two hypotheses are required to achieve the proofs in intuitionistic logic and are satisfied by any "reasonable" set of nodes, such as a finite set or the set of natural numbers. The third one is used in a proof by cases on whether there exists a grey node or not. However, since we have established that the number `g` of grey nodes is always finite, we could have compared `g` with 0, and this can be done constructively.

Finite Sets

Our axiomatization of finite sets made it possible not to invoke the extensionality axiom. Moreover, in [11] the proof of the unicity of the cardinal uses the axiom of excluded middle whereas we only need the decidability of the equality on the set `A` whose elements are being counted. This is not the only advantage of our definition.

Let us recall that the term

`(card_color c m nb)`

expresses that there are `nb` nodes of color `c` in marking `m`. We define:

`(card_color` c_0`) := (card` λ`c.(c=`c_0`))`

This definition is parameterized by a color c_0 and is independent of any marking `m`. Such a parameterization is rather convenient and is not possible with the definition of `cardinal`, as defined in [11], since this notion is expressed by:

`(cardinal` λ`n:node.(m n)=`c_0`)`

which requires to explicitly refer to a marking `m`. The "computational" part `m` and the logical part "λ`c.(c=`c_0`)`" cannot be separated.

Let us also point out that the conclusion in the type of the second constructor of `card` is `(card P` M_2 `(nb+1))` whereas, for the predicate `cardinal`, it is `(cardinal` $Q \cup \{a_0\}$ `(nb+1))`. When applying such constructors to a goal, the variables `P` and M_2 can be instantiated easily whereas a term of the form $Q \cup \{a_0\}$ can only be unified with a term of the form `Q'`$\cup\{a\}$.

Size, Automatization

The development (without the part related to Linear Temporal Logic), consists of 10000 lines of code shared out among 40 separately-compiled sections. It contains approximatively 90 definitions and 450 lemmas, a third of which is related to the safety property, and the rest to the liveness property. Let us point out that this evaluation is purely indicative of the proof length since it strongly depends on the proof development style.

Certain parts of the proofs have been automatized, using a tactic that implements a Prolog-like resolution procedure. The drawback is that this hides the trace of the reasoning (even though it can be retrieved) and above all it slows down the compiling and the proof replaying, which is a real disadvantage since a slight modification in a file can require a lot of extra time.

6 Related Work

We have compared in sections 4.1 and 5, our axiomatization of finite sets with that of Huet and Kahn [11]. In [9] Goulbault-Larrecq proposes another nice axiomatization of finite sets in Coq, based on trees indexed by binary integers.

As far as mechanically verified garbage collectors are concerned, one of the most recent studies is that of Moreau and Duprat [14]. Their paper describes the verification of a distributed reference counting algorithm in Coq. No temporal logic is formalized: termination is expressed by the well-foundedness of a "successor" relation on the states.

Goguen, Brooksby, and Burstall [7] formulate in Coq the incremental tracing presented in [5]. First, a local presentation of the memory as a labelled pre-graph is given, then a model of incrementally traced memory is formalized. Considering the tracing actions between two such pre-graphs as unobservable τ-actions, a behavioral equivalence between the two models is established. The problem of incremental collecting is not tackled and no liveness property is established. Let us mention that bisimulations are not specified by means of co-inductive types.

Jackson ([12]) has verified a tricolor mark-and-sweep garbage collection algorithm in PVS. The algorithm is sequential and not interruptible: once a call to the GC has been made, this process runs until all the garbage is collected. The proof is much simpler and as far as the LTL formalization is concerned, executions are classically represented as functions on the natural numbers.

Russinoff [17] has verified safety and liveness properties of a concurrent garbage collector originally suggested by Ben-Ari [1]. This algorithm is simpler since it only uses two colors. The proofs are carried out with the Boyer-Moore theorem prover using a verification system for concurrent programs described in [16].

Havelund and Shankar [10] have verified a safety property of Ben-Ari's algorithm in PVS, using a refinement approach. The safety property is formulated as an abstract algorithm, and the proof is based on refinement mappings. No liveness properties are considered.

Gonthier proved the safety of a concurrent garbage collector used in Caml-light with the Larch Prover [8]. The state-transition system involves many transitions and the safety about 50 invariants. This is due to the fact that the all details of implementation are considered.

Finally in [19] Verma, Goubault-Larrecq, Prasad, and Arun-Kumar implement and certify BDD's in Coq using reflection. From the proof of correctness of a BDD based tautology checker, a certified tautology-checker in O'Caml is extracted. Moreover, the algorithms presented in this work involve a mark-and-sweep garbage collector for which a safety property is formally established. This amounts to proving that some well-formedness condition on the current BDD is preserved by a call to the GC. The GC considered in this paper is sequential and is not incremental. No general temporal notions are introduced and no liveness considerations are tackled.

7 Conclusion

Proving the correctness of the algorithm we have presented is particularly tricky. The history of the proofs of the algorithms derived from the seminal work of Dijkstra, Lamport et al. [5] is recounted in [10] where the many logical traps and flaws in the *paper and pencil* proofs are described. Obviously, it is easy to skip reasoning steps in our proof and even to make errors when specifying the transitions (this is what we did and discovered later on when trying to prove liveness! Fortunately the mistake was slight and did not raise major problems in the proofs already done). For these reasons, a formal mechanized verification was imperative, especially in the context of software to be embedded on Smart Cards.

First, we were led to design logical tools such as a variant of the axiomatization of finite sets given in [11] and an embedding of linear temporal logic in the Calculus of Inductive Constructions [3]. This part of our work is general and therefore is re-usable.

Moreover we have specified the algorithm at an abstract level in order to cover several possible implementations. Consequently, our work applies to a whole class of concrete algorithms. This non trivial case study is a good illustration of the feasibility and the advantages of our methodology for treating elegantly the temporal aspects by means of inductions or co-inductions on temporal operators. Our approach also makes clear why and where specific fairness conditions are necessary. We think that this study can be useful for people interested in certifying garbage collectors and more generally in temporal reasoning.

8 Acknowledgements

This work was partially supported by action COLOR *Clé* between INRIA Sophia Antipolis, the Université de Provence, and Gemplus. The authors wish to thank the anonymous referees for their helpful comments on earlier drafts of this paper.

References

[1] Ben-Ari, M. Algorithms for On-the-fly Garbage Collection. *ACM Toplas*, 6, July 1984.

[2] Thierry Coquand and Gérard Huet. Constructions : A Higher Order Proof System for Mechanizing Mathematics. *EUROCAL 85, Linz Springer-Verlag LNCS 203*, 1985.

[3] Solange Coupet-Grimal. An Axiomatization of Linear Temporal Logic in the Calculus of Inductive Constructions. *The Journal of Logic and Computation*, 2002.

[4] Stéphan Courcambeck. Mécanismes Avancés de Gestion de Mémoire dans les Systèmes Ouverts pour Cartes à Puce. Technical report, Gemplus Researh Lab., BP 100- 13881 Gemenos Cedex France, 2000.

[5] Edsger W. Dijkstra, Leslie Lamport, A.J. Martin, C.S. Scholten, and E.F.M. Steffens. On-the-fly garbage collection: An exercise in cooperation. *Communications of the ACM*, 21(11):966–975, November 1978.

[6] Eduardo Giménez. *Un Calcul de Constructions Infinies et son application a la vérification de systèmes communicants*. Thèse d'université, Ecole Normale Supérieure de Lyon, December 1996.

[7] Healfdene Goguen, Richard Brooksby, and Rod Burstall. An Abstract Approach to Memory Management. In *Proceedings of TYPES*, volume 1956. Springer-Verlag, 1999.

[8] Georges Gonthier. Verifying the Safety of a Practical Concurrent Garbage Collector. In *Computer Aided Verification CAV'96*, number 1102 in LNCS, pages 462–465. Springer-Verlag, 1996.

[9] Jean Goubault-Larrecq. The Coq Standard Library. *http://pauillac.inria.fr/coq/library/MAPS/*.

[10] Klaus Havelund and Natarajan Shankar. A Mechanized Refinement Proof for a Garbage Collector. *Unpublished manuscript*, 1997.

[11] Gérard Huet and Gilles Kahn. The Coq Standard Library. *http://pauillac.inria.fr/coq/library/SETS/*, 1995.

[12] Paul B. Jackson. Verifying a Garbage Collection Algorithm. In *TPHOL'98 (11th International Conference on Theorem Proving in Higher Order Theory)*, volume 1479. Springer-Verlag, 1998. Series LCNS.

[13] Zohar Manna and Amir Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag, 1991.

[14] Luc Moreau and Jean Duprat. A Construction of Distributed Reference Counting. *Acta Informatica*, 37:563–595, 2001.

[15] Christine Paulin-Mohring. *Définitions Inductives en Théorie des Types d'Ordre Supérieur*. Habilitation à diriger les recherches, Université Claude Bernard Lyon I, December 1996.

[16] David M. Russinoff. A Verification System for Concurrent Programs Based on the Boyer-Moore Prover. *Formal Aspects of Computing*, 4:597–611, 1992.

[17] David M. Russinoff. A Mechanically Verified Incremental Garbage Collector. *Formal Aspects of Computing*, 6:359–390, 1994.

[18] The Coq Development Team. The Coq Proof Assistant Reference Manual – Version V7.1. Technical report, LogiCal Project-INRIA, 2001.

[19] Kumar Neeraj Verma, Jean Goubault-Larrecq, Sanjiva Prasad, and S. Arun-Kumar. Reflecting BDDs in Coq. In *6th Asian Computing Science Conference (ASIAN'2000)*, number 1961 in LNCS, pages 162–181. Springer-Verlag, 2000.