Vérification formelle d'une implémentation d'un gestionnaire de mémoire pour un compilateur certifié

Tahina Ramananandro
École normale supérieure

Xavier Leroy
Projet GALLIUM, INRIA Rocquencourt

19 février – 27 juillet 2007

Plan

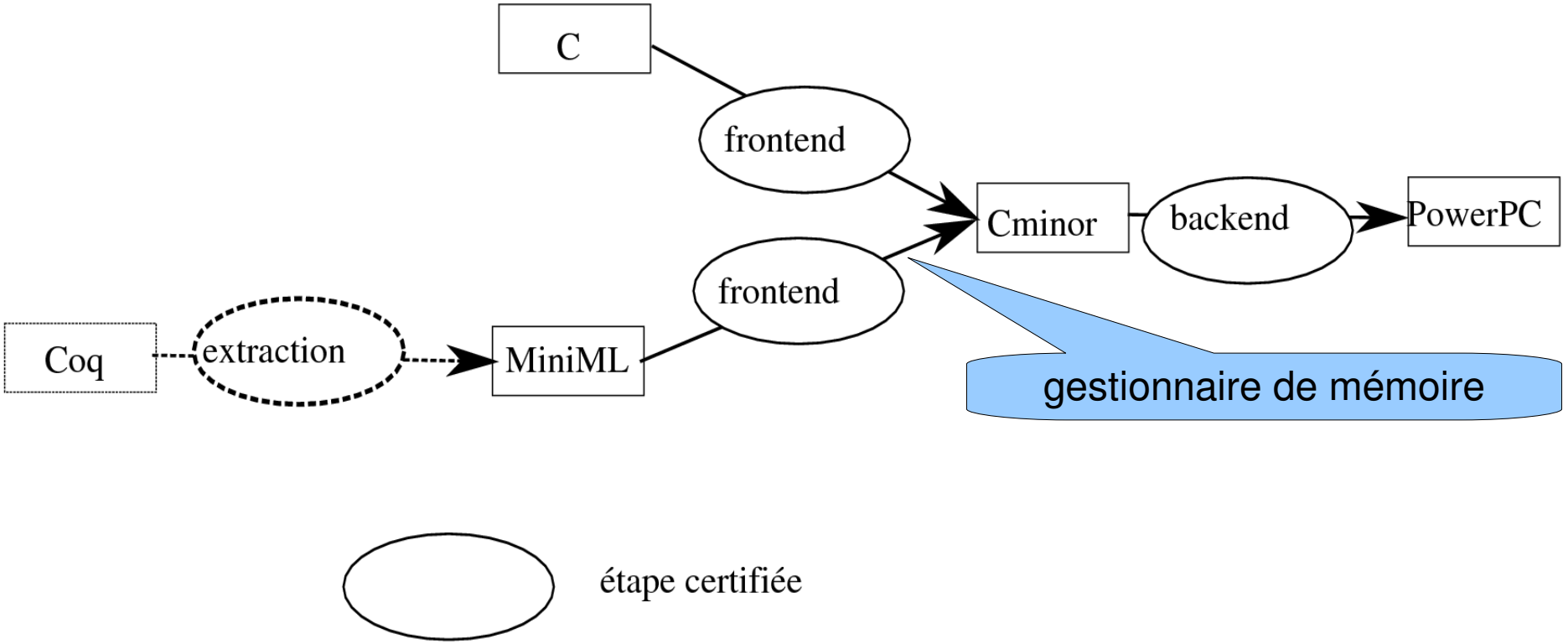
- Contexte
- Modélisation de la mémoire
- Preuve du marquage
- Balayage et allocation
- Conclusion

Plan

- Contexte
  - Le compilateur CompCert
  - Gestionnaire de mémoire
  - GC
- Modélisation de la mémoire
- Preuve du marquage
- Nettoyage et allocation
- Conclusion

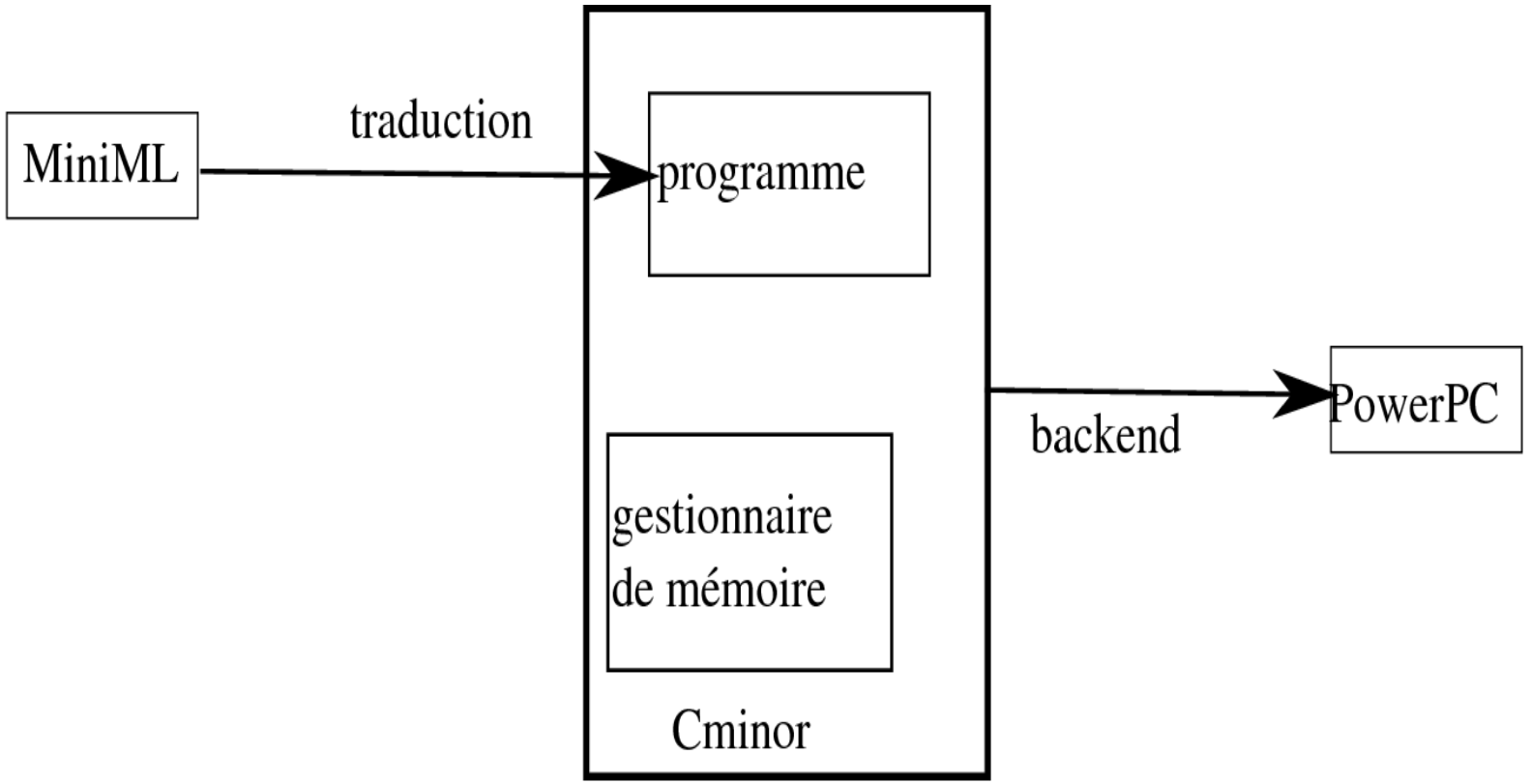
Le compilateur CompCert (1/2)

- Compilateur certifié
  - Sémantique code objet = sémantique code source
  - Développé en utilisant l'assistant de preuve Coq
  - Deux langages d'entrée : C (impératif) et MiniML (fonctionnel)
  - Un langage de sortie : assembleur PowerPC



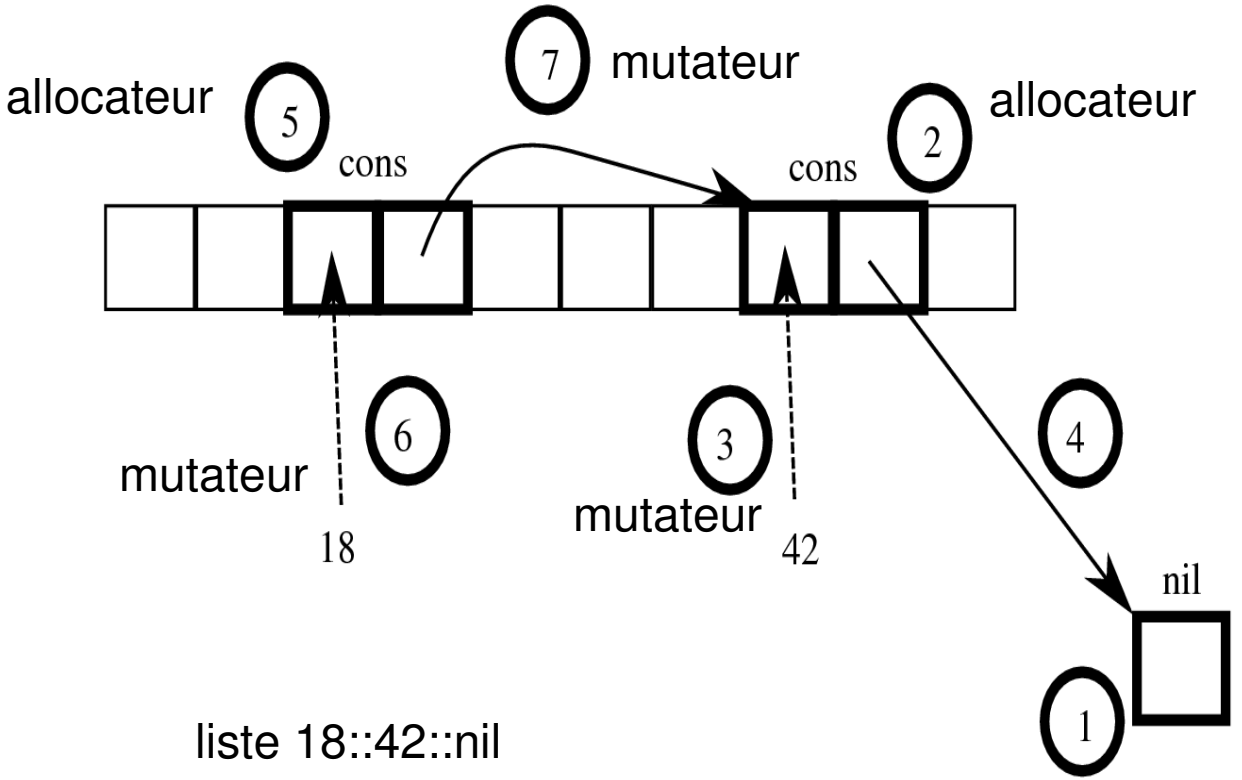
Le compilateur CompCert (2/2)

- Gestionnaire de mémoire = code Cminor à rajouter au programme MiniML compilé



Gestionnaire de mémoire

- Allocateur et mutateur



GC (1/3)

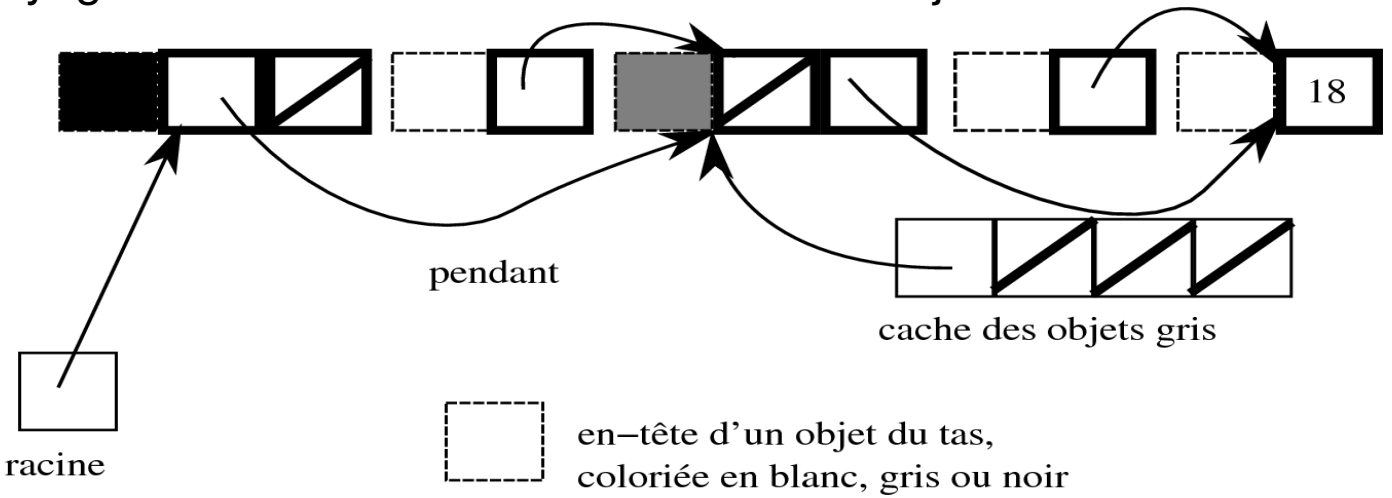
- Mémoire finie
- Pas de libération a priori
  - Objets inaccessibles encore en mémoire
  - Comment les retrouver et les libérer ?
- Adjoindre un GC (garbage collector, ramasse-miettes) à l'allocateur
  - L'allocateur essaie de trouver un espace libre
  - Si échec, le GC libère les espaces occupés par les objets inaccessibles

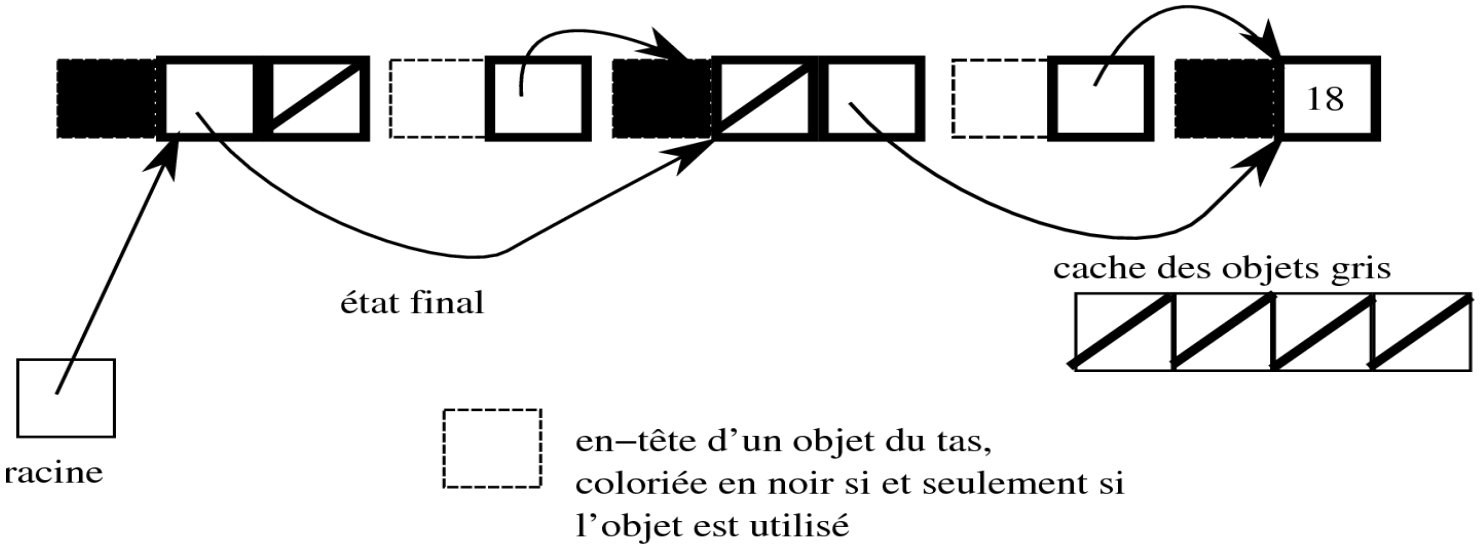
GC (2/3)

- Deux grandes familles
  - Mark and sweep
  - Stop and copy
- Problème : peuvent monopoliser le temps d'exécution
- Raffinements possibles
  - Incrémentaux
  - Concurrents

GC (3/3)

- Marquage 3 couleurs avec cache
- Nettoyage avec coalescence des blocs libres adjacents





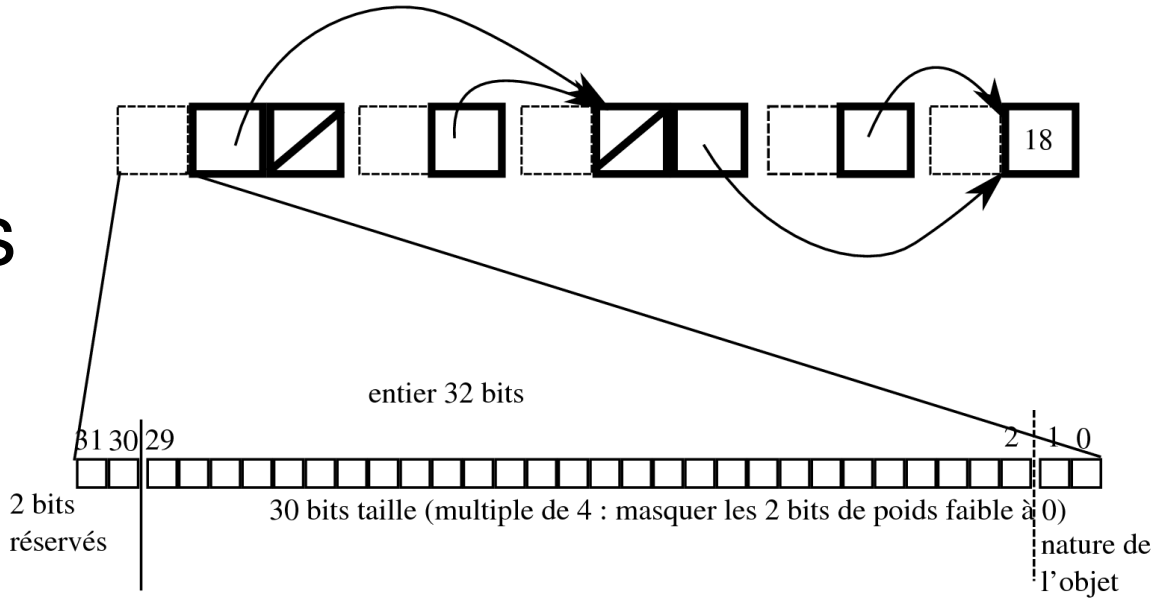
Plan

- Contexte
- Modélisation de la mémoire
  - Tas
  - Racines
  - Chemin
- Preuve du marquage
- Nettoyage et allocation
- Conclusion

Modélisation du tas

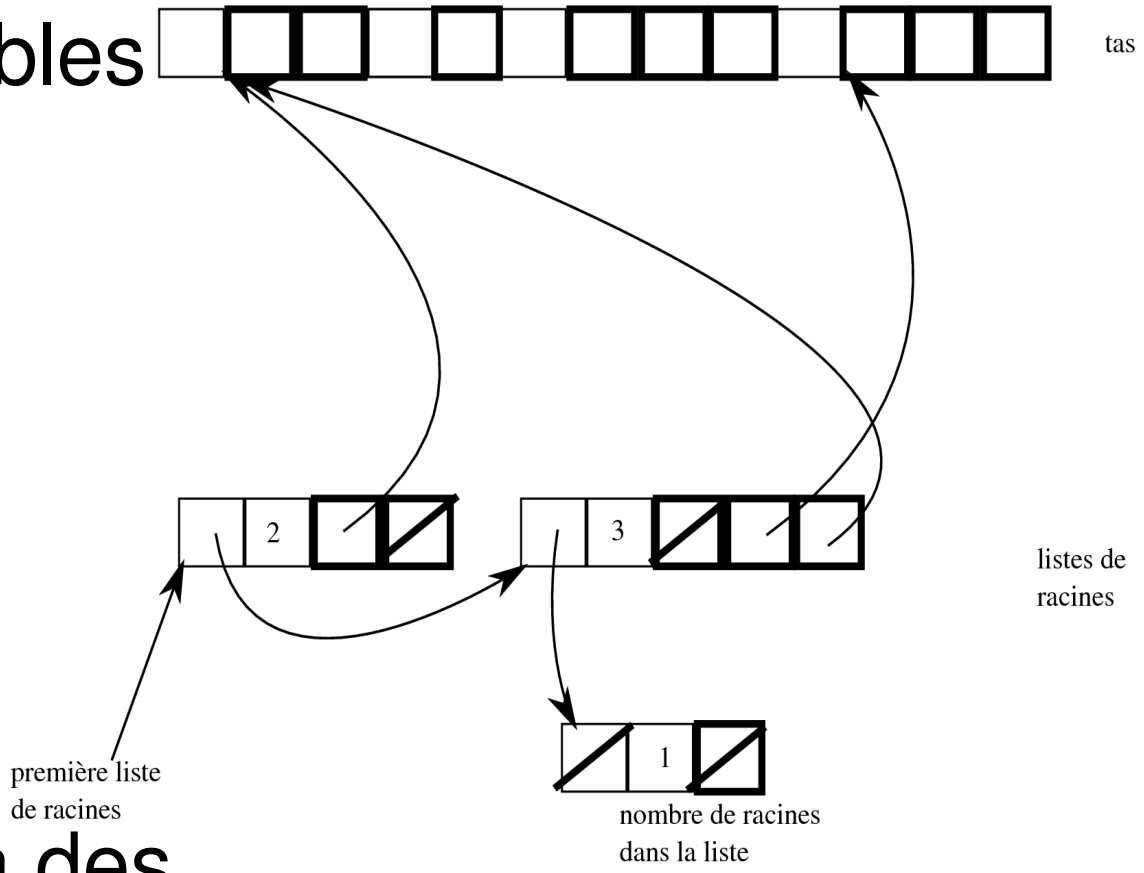
```
Inductive block_description_from_to (m : mem) (heap : block) : int → int →
list int → Prop :=
| block_description_end : ∀ he, block_description_from_to m heap he he nil
| block_description_block : ∀ he n, (n = he → False) →
∀ v, Some (Vint v) = load Mint32 m heap (signed n) →
(Size_header v ≠ zero → ∀ x, modulo_four x n → cmp Cle (add n four) x =
true →
cmp Cle x (add n (Size_header v)) = true →
valid_access m Mint32 heap (signed x)) →
cmp Cle n (add n three) = true →
cmp Cle (add n three) (add (add n (Size_header v)) three) = true →
cmp Cle (add (add n (Size_header v)) three) (sub he one) = true →
∀ l, block_description_from_to m heap (add (add n (Size_header v)) four) he
→
block_description_from_to m heap n he (add n four :: l).
```

- En-tête et contenu
- Coq : entiers 32 bits
  - objets adjacents
  - conditions d'ordre



Modélisation des racines

- Objets racine = visibles dans le scope
  - Variables locales, globales, pile d'exécution
- Objets eux-mêmes dans le tas
- Pointeurs stockés à des endroits spéciaux
  - liste de listes de racines



Chemin dans la mémoire

- Quelle liste de racine, quelle racine, quels objets fils successivement
  - Objet utilisé = accessible à partir des racines
- Chemins doivent être préservés par le GC
  - mark and sweep : exactement
  - stop and copy : à isomorphisme près
- Idée : dissocier chemin et son interprétation
  - plusieurs interprétations possibles pour les raisonnements locaux ou les abstractions

Plan

- Contexte
- Modélisation de la mémoire
- Preuve du marquage
  - Présentation de la preuve
  - Code Cminor, algo concret Coq, algo abstrait
  - Preuve de l'algorithme abstrait
  - Abstraction de la mémoire
  - Preuve de l'abstraction
- Nettoyage et allocation
- Conclusion

Présentation de la preuve de marquage (1/2)

- Algorithme
  - Marquer en gris les objets racine
  - Tant qu'il y a des objets gris :
    - en prendre un, le marquer en noir
    - marquer en gris tous ses enfants non encore marqués
- Avec cache :
  - chaque objet gris empilé en cache
    - recherche d'un objet gris = dépiler le cache
  - cache borné
    - indicateur si le cache a été en overflow au moins une fois
    - dans ce cas, si cache vide, rechercher un objet gris dans le tas

Présentation de la preuve de marquage

- Montrer que l'algorithme marque en noir tous les objets accessibles et eux seulement
- Prouver ce théorème sur un algorithme abstrait
  - représentation abstraite de la mémoire
  - suppose les structures tas et racines fixées
- Puis prouver la relation d'abstraction pour chaque étape du marquage

Code Cminor, algo concret Coq et algo abstrait

- Trois niveaux d'abstraction pour les algorithmes
  - code Cminor
  - algorithme concret Coq, manipulant la mémoire
  - algorithme abstrait, sur des structures abstraites (graphes, fonctions...)
- Code Cminor et algorithmes concrets
  - Prouver qu'ils mènent à la même mémoire
  - Preuve facile mais Coq inadapté
- Algorithmes concrets et abstraits
  - Raisonnements bcp plus poussés

Preuve de l'algorithme abstrait

- Montrer l'invariant tricolore : aucun objet noir ne pointe vers un objet blanc
- Preuve courante dans les preuves de GC sur papier
  - Néglige les structures tas et racines considérées comme inchangées

Abstraction de la mémoire

- État abstrait :
  - liste des objets du tas et de leurs fils
  - fonction de coloration des objets du tas
  - liste des objets du cache
- État concret : mémoire
- Relation d'abstraction
  - tas bien formé
  - couleurs lues dans les en-têtes des objets du tas
  - cache lu comme une liste de pointeurs

Preuve de l'abstraction

- Prouver que la relation d'abstraction est conservée par chaque étape du marquage
  - Structures tas racines abstraites vers le même état
  - Couleurs et cache lus comme dans l'algo abstrait
- Briques de base :
  - marquage en gris (avec mise en cache)
  - marquage en noir (une seule opération)
- Autres opérations : boucles fondées sur ces briques de base
  - pas d'écriture en dehors de ces briques

Plan

- Contexte
- Modélisation de la mémoire
- Preuve du marquage
- Balayage et allocation
  - Algorithmes «duaux»
  - Problèmes de raisonnement
- Conclusion

Balayage et allocation : algos duaux

- Balayage avec coalescence des objets libres
  - À partir de la coloration obtenue par marquage, ajouter les objets blancs = libres à la liste des objets libres
  - si deux objets libres consécutifs, les fusionner
- Allocation
  - Parcourir la liste des objets libres jusqu'à trouver un objet de taille suffisante
  - Le scinder en deux
- Raisonnement dual fusion/scission

Problèmes de raisonnement

- Tailles variables -> coalescence
  - Maintenir les objets adjacents, le tas bien formé
  - Et les objets vides ? Bogues possibles !
- Problèmes arithmétiques
  - Ordre sur les adresses des objets du tas
  - Stockage des tailles dans les en-têtes
  - Entiers 32 bits, manque théorèmes en bibliothèque reliant ordre et représentation binaire machine (bit de signe)

Plan

- Contexte
- Modélisation de la mémoire
- Preuve du marquage
- Balayage et allocation
- Conclusion
  - Bilan
  - Travaux connexes
  - Perspectives

Bilan

	Marquage	Balayage et allocation
Algorithmes abstraits	Rédigés et prouvés	Approche différente
	Preuve d'abstraction terminée	
Algorithmes concrets	Rédigés intégralement	Preuves en cours
	Preuve de correction en cours	
Code Cminor	Préexistants et adaptés	Préexistants et adaptés

	Spécifications	Preuves
Librairies utilitaires	900	1500
Tas et racines	2200	2900
GC Mark & Sweep	2600	5900
TOTAL	**5700**	**10300**
Backend CompCert	*17600*	*15100*

Nombre de lignes de code Coq
(ordre de grandeur)

Problèmes rencontrés

- Coq
  - Validation des preuves longue voire boucle
  - Nécessite d'adapter les spécifications
    - structure de boucle générique au lieu de fonctions récursives sur des entiers naturels !
- Arithmétique des entiers 32 bits machine
  - Prouver les écritures et lectures à des endroits différents
  - Entiers 32 bits machine, peu de théorèmes reliant ordre et opérations
  - Pas d'automatisation (omega...)

Travaux connexes

- **Coupet-Grimal, Nouvet (2003)**
  - GC incrémental formalisé et prouvé en Coq
  - Algorithme abstrait
- **McCreight, Shao, Lin, Li (2007)**
  - Différents GC (mark & sweep, stop & copy, etc.) formalisés et prouvés en Coq
  - Langage machine : implémentation plus bas niveau
    - avec entiers naturels au lieu d'entiers machine
  - Interface GC/mutateur
- **Doligez (1995)**
  - GC concurrent (parallèle avec le mutateur)
  - Non Coq, mais TLA+
  - Algorithme abstrait

Perspectives

- Preuves balayage et allocation
  - améliorer/automatiser raisonnement entiers 32 bits
- Raisonnement sur les programmes en Cminor
  - Outils type Caduceus
  - Extraction depuis les algorithmes concrets Coq
    - fragment monadique (cf. Tolmach)
- Autres preuves
  - Stop & copy, GC incrémentaux, concurrents
  - Interface GC/mutateur
  - Autres composants des environnements d'exécution : threading, machines virtuelles...

Merci de votre attention !

- Pour toute question, merci de contacter :
  - Tahina Ramananandro
    Tahina.Ramananandro@normalesup.org
  - Xavier Leroy
    Xavier.Leroy@inria.fr
- Sources Coq bientôt disponibles sur le Web
  - http://www.eleves.ens.fr/~ramanana/travail/gc