7
Conclusão

Relativamente às abstrações mais comuns encontradas em linguagens de programação, finalizadores e referências fracas têm um uso mais restrito, concentrado sobretudo em aplicações de grande porte ou complexidade, com um alto nível de utilização de memória. Para este conjunto de aplicações tais dispositivos têm especial importância na medida em que constituem-se em soluções eficazes, e por vezes únicas, para tratar uma série recorrente de problemas.

Finalizadores e referências fracas permitem que qualquer aplicação interaja dinamicamente com o coletor de lixo, alterando o processo básico de coleta, ou simplesmente obtendo e consumindo informações sobre mudanças na conectividade de objetos. Apesar de estarem associados a conceitos relativamente simples, são mecanismos inerentemente complexos, e possuem semânticas que dependem das características específicas do coletor de lixo, bem como de algumas decisões básicas de implementação sobre as quais não existe muito consenso. Em decorrência disso o suporte a finalizadores e referências fracas varia consideravelmente entre as linguagens de programação mais conhecidas.

Conforme discutimos, em sistemas que empregam coletores de lixo baseados em rastreamento a utilização de finalizadores introduz linhas de execução concorrentes. Isso impõe a necessidade de compreender e tratar adequadamente requisitos de sincronização mais extensos e complexos, não apenas durante o processo de desenvolvimento de aplicações, mas também na própria implementação da linguagem considerada. Em particular, a execução de finalizadores em threads que detêm locks pode levar à violação de mecanismos básicos de sincronização. Como consequência de otimizações frequentemente efetuadas por compiladores, alguns objetos podem ser finalizados prematuramente, criando condições de corrida inesperadas.

Em sistemas que empregam contagem de referências, problemas de sincronização podem ser evitados através de uma análise cuidadosa do fluxo de execução do programa, além da restrição ao uso de mecanismos explícitos de sincronização

em finalizadores. Essas práticas contudo nem sempre são razoáveis, ou mesmo possíveis. A invocação assíncrona de finalizadores constitui uma solução alternativa, mas que implica nos mesmos problemas e requisitos de sincronização inerentes a sistemas que utilizam coletores de lixo baseados em rastreamento.

Mesmo quando implementados corretamente, finalizadores podem impor custos de desempenho não desprezíveis. Além disso, o indeterminismo comumente associado a sua invocação, incluindo a falta de garantias quanto a ordem de execução e quanto a invocação ao término da aplicação, restringem a sua utilização em algumas situações importantes. Essa limitação pode ser parcialmente contornada em sistemas que oferecem suporte à invocação explícita do coletor de lixo, e à execução antecipada de finalizadores associados à objetos considerados inacessíveis. Independentemente de qualquer outro fator, a ordem de execução de finalizadores, como demonstramos, pode ser controlada explicitamente pelo programa cliente através do uso de uma estrutura de dados estática.

Vimos ainda que, de forma geral, referências fracas são mecanismos mais simples e mais expressivos do que finalizadores. Quando associadas a um dispositivo de notificação podem ser empregadas inclusive para finalizar objetos. A finalização através de callbacks usualmente apresenta duas vantagens em relação a mecanismos de finalização mais tradicionais baseados em classes:

- O objeto finalizável pode ser desacoplado da rotina de finalização, evitando atrasos na reciclagem da memória.
- O objeto não deixa de ser acessível antes de ser finalizado, garantindo um maior controle da aplicação.

Filas de notificação oferecem estes mesmos benefícios, e ainda evitam muitos dos problemas de sincronização associados a execução implícita de finalizadores e callbacks. O programa cliente pode esperar por condições específicas para executar as ações associadas à finalização de um objeto. A escalonação dessas rotinas acontece sempre de maneira explícita.

Implementações em que o coletor insere o objeto fracamente referenciado na fila de notificação, ao invés da referência fraca, proporcionam um controle mais amplo sobre a dinâmica de coleta e finalização. Por outro lado restauram o acoplamento entre objetos e finalizadores, e limitam a automatização do processo de finalização.

Apesar das vantagens acima, implementações simples de referências fracas não são suficientemente expressivas para tornar o suporte a finalizadores completamente desnecessário. Isso decorre de duas razões básicas:

- A finalização através de referências fracas implica no uso de uma estrutura externa ao objeto para armazenar as informações necessárias à finalização, o que em linguagens orientadas a objetos, fere o princípio básico de encapsulamento de dados.

- Em alguns casos é essencial que o finalizador tenha acesso ao objeto finalizado.

O problema de quebra de encapsulamento pode ser atenuado, mas não tem como ser inteiramente evitado. Contudo, em linguagens não orientadas a objetos esse problema não é tão relevante, e a finalização baseada em coleções constitui um mecanismo até mais natural para gerenciar o ciclo de vida de objetos.

Finalizadores baseados em classes podem ressuscitar objetos utilizando um critério de decisão qualquer. Referências fracas ordinárias, ao contrário, não permitem esse tipo de controle sobre o processo de coleta. Como vimos porém, algumas extensões simples tornam possível tratar esse problema com relativa facilidade. Se o controle da coleta de objetos for baseado na disponibilidade de memória, uma situação relativamente comum, basta introduzir diferentes gradações de referências fracas. Caches implementados com referências suaves atendem perfeitamente este tipo de requisito: preservam o objeto referenciado apenas enquanto a memória disponível for abundante. Se o controle da coleta de objetos for baseado em um critério mais específico, a linguagem pode ser estendida para permitir a parametrização do critério de limpeza de referências fracas, como acontece por exemplo em SmartEiffel.

A definição do critério empregado para efetuar a coleta de referências fracas para as quais foram registrados finalizadores é uma decisão semântica importante. Pode ser interessante impedir a coleta destas referências até que os respectivos finalizadores sejam executados. Isso permite que a finalização de um objeto não dependa da conectividade de um outro objeto, evitando complicações adicionais na estruturação do programa. Para que seja possível modificar ou cancelar a finalização de qualquer objeto, o sistema deve oferecer algum tipo de chamada para acessar as referências fracas acopladas a finalizadores. Isso evita ainda que referências fracas tornem-se temporariamente inacessíveis (zumbis).

Referência fracas podem ser usadas também na implementação de coleções, como conjuntos e tabelas fracas. Estas estruturas de dados são úteis em diversos tipos de aplicação, e em algumas linguagens constituem a única forma de referência fraca existente.

Para tratar adequadamente ciclos de referências internas, tabelas fracas com valores fortes devem obedecer a uma semântica análoga a de ephemerons: a conectividade do valor é definida em função da conectividade da chave. Essa relação não é simples, e para ser incorporada pela linguagem demanda uma modificação específica na implementação do coletor de lixo.

Com o objetivo de identificar com precisão e esclarecer muitas das questões associadas à especificação e à implementação de finalizadores e referências fracas, como uma das partes mais importantes deste trabalho desenvolvemos um modelo abstrato capaz de representar o estado e a dinâmica da memória em um linguagem de programação convencional. Através deste modelo conseguimos descrever a semântica de finalizadores e referências fracas de forma bem abrangente, e demonstramos como estes mecanismos interferem na semântica básica de uma linguagem de programação. Ambos são capazes de tornar acessíveis novamente objetos que não são mais referenciados pelo programa cliente, criando algumas dificuldades semânticas relevantes. A ressurreição de objetos depende da dinâmica de execução do coletor de lixo, que em sistemas baseados em rastreamento, não é determinística.

Mesmo diante da possibilidade de ressurreição, conseguimos demonstrar formalmente que é possível definir uma semântica correta para finalizadores e referências fracas, isto é, uma semântica que não gera ponteiros quebrados e nem leva a vazamentos de memória.

7.1 Contribuições

Neste trabalho tentamos abordar de forma bem abrangente os conceitos de finalizadores e referências fracas, preenchendo um hiato existente na literatura acadêmica. Buscamos também desenvolver uma referência capaz de facilitar o uso e futuras implementações de mecanismos associados a essas abstrações. Resumidamente, as principais contribuições deste trabalho são:

- Efetuamos um amplo levantamento sobre como é feito o suporte a finalizadores e a referências fracas nas principais linguagens de programação usadas atualmente, comparando as características mais importantes de cada interface, e discutindo as suas vantagens e desvantagens.

- Identificamos e analisamos as questões mais significativas relacionadas a semântica, ao uso e a implementação de finalizadores e referências fracas.

Como já ressaltamos, alguns aspectos da semântica de finalizadores foram abordados por outros autores de maneira informal, mas sem a mesma extensão que conseguimos alcançar aqui. A semântica de referências fracas contudo só foi discutida em um número reduzido de artigos, e mesmo assim de forma relativamente limitada. Uma parcela significativa dos tópicos que exploramos não foi previamente considerada.

- Desenvolvemos um modelo abstrato capaz de expressar os principais conceitos relacionados ao gerenciamento de memória em linguagens de programação convencionais. Usando este modelo especificamos formalmente as semânticas de finalizadores, referências fracas, e dos principais mecanismos associados a referências fracas. Com esta formalização conseguimos identificar ainda várias questões semânticas que precisam ser cuidadosamente consideradas na implementação de linguagens que contam com tais facilidades. Não conhecemos nenhum trabalho anterior a este que tenha abordado finalizadores e referências fracas do ponto de vista formal.

- Demonstramos como finalizadores e referências fracas interferem na semântica básica de uma linguagem de programação, e provamos formalmente que esses mecanismos podem ser definidos de forma a garantir a correção da linguagem.

- Apresentamos um algoritmo simples que permite efetuar a finalização ordenada de finalizadores em coletores baseados em rastreamento, mesmo na presença de ciclos.

Um ponto importante que não abordamos nesta tese é como o sistema de tipos de uma linguagem de programação é afetado pela introdução de referências fracas. Assim, como uma linha futura de trabalho seria interessante estender o modelo formal apresentado aqui com um sistema de tipos, e concomitantemente, tentar desenvolver um algoritmo para a inferência de tipos nesta linguagem.