Uma Linguagem de Domínio Específico para Programação de Memórias Transacionais em Java

Marcos Gonçalves Echevarria[1], **André Rauber Du Bois**[1]

[1]Programa de Pós-Graduação em Informática – Mestrado em Ciência da Computação
Universidade Católica de Pelotas (UCPEL) – Pelotas – RS – Brazil

{marcosge,dubois}@ucpel.tche.br

1. Introdução

As arquiteturas multi-core influem diretamente no desenvolvimento de software porque para melhor aproveitar esses processadores os desenvolvedores devem paralelizar seus programas. Programas concorrentes para máquinas multi-core são geralmente implementados usando threads que se comunicam através de uma memória compartilhada. Quando múltiplas threads acessam uma mesma faixa de endereços de memória é possível que alguma intercalação de suas instruções resulte em uma execução inválida. Para evitar essas execuções inválidas, as linguagens fornecem mecanismos de sincronização como por exemplo bloqueios (*locks*), mas sincronizações baseadas em bloqueios possuem algumas armadilhas que dificultam a programação e são propensas a erros [Lee 2006, Jones 2007].

Memórias transacionais fornecem um novo modelo de controle de concorrência que não apresenta as mesmas dificuldades encontradas no uso de bloqueios. Ela traz para a programação concorrente os conceitos de controle de concorrência usados a décadas pela comunidade de banco de dados. Construções em linguagens transacionais são fáceis de serem usadas e podem gerar programas fortemente escaláveis [Adl-Tabatabai et al. 2006].

Este trabalho tem como objetivo apresentar a linguagem CMTJava, uma linguagem de domínio específico para programação de memórias transacionais em Java. Essa linguagem foi criada visando facilitar a programação de máquinas multi-core e sua grande vantagem é que o sistema de tipos da linguagem garante que transações serão executadas apenas através da primitva `atomic`.

2. A linguagem

Com o objetivo de apresentar um simples programa usando a linguagem CMTJava, segue abaixo o código de uma classe `Conta`, a qual possui um atributo `saldo`.

```
class Conta implements TObject{
   private Double saldo;
}
```

CMTJava provê uma abstração de objetos transacionais (interface `TMObject`). TMObject é usada como um alerta para o compilador, a partir desse alerta são gerados automaticamente métodos especiais de `get` e `set` para os atributos da classe. Para a classe `Conta` são gerados os seguintes métodos:

```
STM<Void> setSaldo(Double saldo);
STM<Double> getSaldo();
```

Esses métodos retornam ações como resultado. Uma ação transacional é uma ação que, quando executada, irá produzir um efeito desejado. No caso do método `getSaldo()`, `STM<Double>` é uma ação transacional que retorna como resultado um `Double`.

Ações transacionais podem ser combinadas para gerar novas ações usando um bloco `STM`:

```
public STM<Void> deposito(Double v)
 return STM{ Double s = getSaldo();
             setSaldo(s + v);};
}
```

Para executar uma transação, a primitiva `atomic` deve ser usada:

```
public <A> A atomic(STM<A> t);
```

A primitiva `atomic` executa usando *sincronização otimista*: transações são executadas em paralelo sem o uso de bloqueios, mas uma transação só é efetivada se nenhuma outra transação concorrente modificou uma área de memória em comum. O sistema de tipos da linguagem CMTJava garante que ações transacionais somente serão executadas através da primitiva `atomic`. A linguagem CMTJava também oferece as primitivas `retry` e `orElse` para compor transações sequencialmente ou como alternativas [Harris et al. 2008].

3. Implementação da Linguagem

Ações transacionais podem ser implementadas através do conceito de mônadas [Wadler 1995]. Uma mônada é usada para descrever computações que podem ser combinadas gerando novas computações. Por esta razão, mônadas são freqüentemente usadas na implementação de linguagens de domínio específico. As operações básicas de uma mônada usam funções de alta ordem, dessa maneira, para implementar CMTJava foi usada uma extensão Java que suporta *closures* [jav 2008]. A linguagem CMTJava define um conjunto de regras de tradução que transforma blocos `STM` em operações monádicas usando closures. Todo o suporte a transações é implementado com código Java puro. Outro detalhe importante é que mesmo o sistema sendo desenvolvido em Java, as idéias apresentadas podem ser implementadas em qualquer outra linguagem orientada a objetos que suporte closures, como por exemplo C#.

4. Conclusões

A linguagem CMTJava é a primeira extensão Java para transações que suporta a construção `OrElse`. A idéia foi adaptada da linguagem STM Haskell [Harris et al. 2008] para um contexto de orientação a objetos. Vale ressaltar que este trabalho serve como motivação para a introdução de *closures* na linguagem Java.

Referências

(2008). Java Closures. WWW page, http://www.javac.info/.

Adl-Tabatabai, A.-R., Kozyrakis, C., and Saha, B. (2006). Unlocking concurrency. *ACM Queue*, 4(10):24–33.

Harris, T., Marlow, S., Jones, S. L. P., and Herlihy, M. (2008). Composable memory transactions. *Commun. ACM*, 51(8):91–100.

Jones, S. P. (2007). Beautiful concurrency. In Oram, A. and Wilson, G., editors, *Beautiful Code*, pages 385–406. O'Reilly & Associates, Inc., Sebastopol, CA 95472. ch. 24.

Lee (2006). The problem with threads. *COMPUTER: IEEE Computer*, 39.

Wadler, P. (1995). Monads for functional programming. In Jeuring, J. and Meijer, E., editors, *Advanced Functional Programming*, volume 925 of *LNCS*. Springer-Verlag.