Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Uma Comparação entre Formalismos Utilizados para Especificações Semânticas

Fernando Magno Quintão Pereira

Relatório Técnico do
Laboratório de Linguagens de Programação

LLP001/2002

Av. Antônio Carlos, 6627
31270-010 - Belo Horizonte - MG

26 de Agosto de 2003

1 Introdução

A Semântica Formal é uma área da Ciência da Computação que engloba diversos formalismos diferentes, embora todos eles possam ser utilizados para a especificação de linguagens de programação. Ainda que estes formalismos visem prover uma notação concisa e não ambígua para a realização de especificações formais, não se pode dizer que são equivalentes. Mais correto seria afirmar que são complementares.

Uma vez que os diversos formalismos que compõem a Semântica Formal não têm exatamente os mesmos objetivos, compará-los torna-se uma tarefa difícil e, em alguns casos, mesmo sem sentido. Entretanto, a fim de expor ao leitor uma rápida apreciação a respeito das mais importantes técnicas utilizadas e como cada uma delas se situa em relação às outras, neste documento serão comparados quatro formalismos, a saber: *Semântica Denotacional, Semântica Operacional Estruturada*, *Máquinas de Estado Abstratas* e *π-Cálculo*.

A comparação de quaisquer tipos de estruturas, ainda que estas não sejam completamente equivalentes, exige o estabelecimento de critérios de análise. Assim, serão adotados para fins de comparação os critérios vistos em [2], os quais são listados abaixo:

não ambigüidade A técnica em análise deve facilitar o desenvolvimento de descrições rigorosas que claramente especifiquem o comportamento da linguagem que se procura formalizar.

demonstração O formalismo deve ser sustentado por conceitos matemáticos precisos que permitam a demonstração de propriedades dos programas de maneira irrefutável.

prototipagem Deve ser possível a obtenção de programas executáveis a partir da descrição formal de uma linguagem.

reusabilidade A técnica deve permitir o reúso de partes comuns de uma especificação entre diferentes linguagens. Idealmente, o formalismo deveria conter uma biblioteca de construções semânticas básicas que o desenvolvedor de linguagens poderia adicionar ou remover de maneira simples. Por exemplo, dado que expressões aritméticas são utilizadas em múltiplas linguagens de modo similar, a definição semântica de tais construtos poderia estar incluída em uma biblioteca para posterior reutilização.

modularidade O formalismo em questão deve permitir o desenvolvimento de especificações semânticas de forma incremental. A título de ilustração, caso seja adicionada a uma linguagem expressões aritméticas, outras de suas construções, como por exemplo o comando de atribuição, não deveriam ter de ser modificadas.

inteligibilidade Especificações semânticas devem poder ser lidas por pessoas com diferentes embasamentos. Isto porque o comportamento de uma linguagem de programação afeta todas as pessoas que intervêem no processo de desenvolvimento de software. Descrições semânticas que podem ser lidas apenas pelos seus autores não têm grande utilidade prática.

flexibilidade Existe uma grande variedade de linguagens e paradigmas de programação diferentes. A técnica de descrição semântica deve poder ser capaz de lidar com todos eles, e deve, também, permitir a investigação de novos tipos de linguagens via a combinação de conceitos já existentes.

aplicabilidade O formalismo em análise deve poder ser aplicado na descrição formal de linguagens de programação populares. De fato, pouco uso teria uma técnica que consegue descrever somentes pequenos exemplos de linguagens de programação que não se prestam a nenhum fim prático.

2 Comparação entre técnicas de especificação semântica

Nesta seção serão comparados quatro dentre as principais técnicas utilizadas para descrever formalmente linguagens de programação. São estas técnicas: Semântica Denotacional, Semântica Operacional Estruturada, Máquinas de Estado Abstratas e π-Cálculo.

A fim de melhor ilustrar as diferenças entre as diversas técnicas de especificação formal apresentadas neste relatório, todas elas serão utilizadas para modelar a atribuição de valor à variável. Comandos de atribuição são uma constante em linguagens do paradigma imperativo. Informalmente, a construção `x := 1` significa armazenar na posição denotada pela variável `x` o valor 1.

2.1 Semântica Denotacional

Em Semântica Denotacional, o comportamento de um programa é descrito por meio de *denotações*. Denotações são funções matemáticas que modelam o significado de construções sintáticas de uma linguagem de programação. Esse formalismo, que se baseia em λ-cálculo, foi inicialmente proposto por C. Strachey em 1967 [5]. Dois anos depois, em 1969, D. Scott e C. Strachey estabeleceram as bases teóricas desta técnica usando a teoria dos domínios.

Em Semântica Formal, o significado de cada construto que caracteriza uma linguagem de programação é descrito via equações semânticas. Assim, expressões que, em última instância, denotam números, poderiam ser mapeadas por uma função do tipo $E : Expr \mapsto State \mapsto Number$. Comandos, por outro lado, deveriam ser mapeados por uma função do tipo $C : Comm \mapsto (State \mapsto State)$. Tal função quer dizer que a execução de um comando denota uma transformação de estado. Assim, o comando de atribuição poderia ser descrito pelas seguintes funções:

$$E[Const\ n] = n$$

$$C[x := e]\varsigma = upd\ \varsigma\ x\ (E[e]\varsigma)$$

O principal problema de Semântica Denotacional se refere ao seu caráter pouco modular e pouco reutilizável. Outros formalismos, como as semânticas monádica e de Ação foram propostas a fim de resolver o problema da modularidade. Por outro lado, embora a Semântica Denotacional seja tradicionalmente reconhecida como pouco modular, a Linguagem Script [1] permite que especificações em Semântica Denotacional sejam elaboradas segundo métodos muito semelhantes aos empregados na construção de softwares orientados por objetos. Outro ponto em que a Semântica Denotacional se mostra deficiênte é quanto às possibilidades de prototipagem. De fato, não existem muitos sistemas que possibilitem a obtenção de interpretadores a partir de descrições denotacionais.

O grande apelo sobre a Semântica Denotacional é o seu rigor matemático, uma vez que está baseada em uma teoria já bem consolidada, conhecida como teoria dos domínios. Tal rigor torna esse tipo de semântica adequado à demonstração de propriedades. Além de ser adequada às provas formais, a Semântica Denotacional é flexível, pois permite a modelagem de linguagens provenientes de diferentes paradigmas, como o funcional ou o imperativo. Esse formalismo já foi aplicado com sucesso para descrever diversas linguagens de grande porte, como Prolog, C ou XPath, o que demonstra ser ele

bastante aplicável e poderoso, embora a Semântica Denotacional não se mostre abrangente o suficiente para descrever sistemas concorrentes, atualmente tão comuns, dada a rápida evolução das tecnologias de comunicação.

2.2 Semântica Operacional Estruturada

A técnica conhecida por Semântica Operacional Estruturada foi apresentada por G. Plotkin [4] com o intuito de especificar as transições elementares de um programa via regras de inferência, as quais são definidas por indução sobre a estrutura que se procura descrever. Ao contrário da Semântica Denotacional, a Semântica Operacional procura descrever de modo claro também os estados intermediários que caracterizam uma computação, e não somente o resultado final que advém da execução de um programa sobre sua entrada. Devido a esta particularidade, esse tipo de formalismo é também conhecido como *semântica passo-a-passo*, uma vez que cada passo da execução de um programa pode ser descrito.

A fim de modelar o comando de atribuição em Semântica Denotacional, duas regras de inferência são necessárias. Uma delas modela o fato de, antes de a atribuição ser processada, a expressão atribuída dever ser avaliada até que uma posterior avaliação não seja possível. A segunda regra de inferência modela a modificação do estado que decorre de uma atribuição de valor à variável.

$$\frac{\langle E, s\rangle \rightarrow \langle E', s'\rangle}{\langle I \ := \ E, s\rangle \rightarrow \langle I \ := \ E', s'\rangle}$$

$$\langle I \ := \ v, (m, i, o)\rangle \rightarrow (m[v/I], i, o)$$

A Semântica Operacional Estruturada tradicional não permite o desenvolvimento de especificações caracterizadas por modularidade e reusabilidade. Além disso, o caráter operacional do formalismo dificulta a verificação de propriedades [2]. Por outro lado, a construção de interpretadores para descrições operacionais não é um problema difícil. É importante dizer também que esta técnica já foi aplicada para descrever diversas linguagens complexas e populares de diferentes paradigmas.

2.3 Máquinas de Estado Abstratas

Máquinas de Estado Abstratas, ou ASM, definem um algoritmo usando uma abstração do ambiente no qual tal algoritmo será executado e um conjunto de regras de transição definidas sobre aquele ambiente. Este formalismo foi originalmente chamado *Álgebra Evolutiva*, tendo sido inicialmente descrito por Ian Gurevich [3] e posteriormente desenvolvido por Egon Börger. Máquinas de Estado abstratas podem ser consideradas como uma evolução da Semântica Operacional que funciona como uma nível de abstração mais próximo do algoritmo que está sendo analisado.

A modelagem do comando de atribuição via ASM pode ser feita com uma regra de inferência que utiliza a função dinâmica $loc : Var \rightarrow Value$. Nesta regra, *task* representa a próxima tarefa que deve ser executada.

> if *task* is *var* then
> *val(task)* := *loc(var)*
> *proceed*

Linguagens muito populares e complexas, como Java, por exemplo, já foram descritas com esse formalismo. A especificação via ASM define uma árvore que representa a sintaxe abstrata do programa. A avaliação semântica consiste em percorrer esta árvore. A cada nodo, uma dada ação é executada e então o fluxo de controle segue para a próxima tarefa. O estado de uma computação é representado via funções dinâmicas. Tais funções podem ser modificadas durante o processo de execução de uma especificação.

Dentre os quatro formalismos apresentados neste relatório, ASM é o mais geral, pois pode ser utilizado para especificar algoritmos, linguagens de diferentes paradigmas e até sistemas distribuídos. As especificações ASM são altamente modulares e dotadas de grande aplicabilidade, já tendo sido, por exemplo, utilizadas para descrever a Linguagem Java. Esse formalismo, por utilizar uma notação próxima da que é encontra na maior parte das linguagens imperativas, o torna legível até mesmo para programadores que não têm grande conhecimento a respeito de semântica formal. Além disso, existem diversos sistemas capazes de executar especificações baseadas em Máquinas de Estado Abstratas.

Por outro lado, as descrições baseadas em ASM não são tão sucintas quanto as que podem ser encontradas nos outros paradigmas, o que pode dificultar um pouco a prova de propriedades matemáticas das construções que estão sendo analisadas.

2.4 π-Cálculo

π-Cálculo é um modelo matemático para especificação semântica onde processos interagem por meio do envio de canais de comunicação entre si. Assim como no λ-Cálculo todas as entidades utilizadas são funções, em π-Cálculo todas as expressões representam processos. Segundo esse formalismo, o passo básico de computação é a transferência de uma mensagem que representa um canal de comunicação de um processo para outro. Devido a esta mecânica baseada em trocas de mensagens, o π-Cálculo é adequado para modelar sistemas onde o acesso aos recursos distribuídos varia com o tempo.

Ao invés de ilustrar o π-Cálculo com a especificação do comando de atribuição, será mostrado como o envio de um canal de comunicação a por um canal b pode ser descrito. Neste caso, o processo que envia a por meio de b é $\bar{b}a.S$. O cliente que recebe um canal via b e depois o utiliza para enviar dados é descrito por $b(c).\bar{c}d.P$.

$$\bar{b}a.S \rightarrow b(c).\bar{c}d.P$$

Dentre as técnicas de especificação analisadas, o π-Cálculo é a melhor quando o objetivo é a modelagem de sistemas distribuídos. Este formalismo provê uma notação clara e concisa para especificar a troca de mensagens entre processos e já foi utilizado para a modelagem de diversos sistemas distribuídos diferentes, por exemplo, baseados em objetos distribuídos ou em troca de mensagens.

O π-Cálculo possui um caráter muito específico, contudo. Esta técnica pode ser utilizada para modelar conceitos de linguagens de programação como valores numéricos ou boleanos, entretanto, como este não é o objetivo primordial do formalismo, as definições acabam se tornando muito complexas e pouco legíveis.

3 Conclusão

A Semântica Formal é uma importante área de estudos e seu domínio contribui grandemente para a formação de bons profissionais da informática. O conhecimento de semântica possibilita ao programador maior entendimento acerca da estrutura de linguagens de programação, a sua principal ferramenta de trabalho. Além disso, a formalização contribui para que especificações sejam menos propensas a erros que doutro modo não poderiam ser encontrados.

Por fim, formalismos contribuem para que descrições de de uma linguagem de programação possam ser transmitidas de maneira não ambígua entre os desenvolvedores da mesma.

Não é necessário que programadores conheçam a fundo os conceitos que fundamentam quaisquer dos formalismos citados neste relatório para que, ainda assim, possam desenvolver sistemas de computação de maneira clara e bem estruturada. O conhecimento semântico da linguagem de programação utilizada, contudo, facilita o desenvolvimento de aplicações, uma vez que questões como a maneira pela qual parâmetros são passados para funções ou como são avaliadas as expressões aritméticas sempre estão presentes durante o exercício da programação.

Outra, dentre as principais vantagens da formalização, é a possibilidade de que erros, em uma especificação, possam ser mais facilmente encontrados e removidos. Embora custosas, provas de corretude são necessárias em sistemas críticos, como os responsáveis pelo tráfego em aeroportos ou pelas movimentações financeiras em bancos. Para que propriedades possam ser provadas, é necessário que as estruturas sob análise possam ser descritas segundo algum formalismo bem fundamentado.

Finalmente, é importante dizer que descrições semânticas de uma linguagem de programação permitem que os principais construtos desta linguagem sejam definidos de maneira precisa. Um formalismo que permita especificações não ambiguias é necessário para que os desenvolvedores da linguagem tenham como transmitir suas intenções de modo preciso. Neste caso, a linguagem natural não se mostra adequada, uma vez que possibilita diferentes interpretações.

Devido às razões acima citadas, torna-se claro o importante papel que os formalismos possuem no contexto da Ciência da Computação. Embora não seja impressindível para a formação de um bom profissional, o conhecimento em semântica tem emprego seguro em situações nas quais especificações claras, concisas e não ambíguas sejam necessárias.

Referências

[1] Bigonha, Roberto da Silva. *Script 2.1, An Object Oriented Language for Denotational Semantics.* Relatório Técnico 0xx/00. Departamento de Ciência da Computação, Universidade Federal de Minas Gerais, Julho, 2000.

[2] Labra, J. E. *Reusable Semantic Specifications of Programming Languages*. VI Simpósio Brasileiro de Linguagens de Programação. Maio, 2002.

[3] Gurevich. Y. *Evolving Algebra 1993: Lipari Guide*. Specification and Validation Methods. Oxford University Press, 1995.

[4] Gordon D. Plotkin. *A Structural Approach to Operational Semantics*. Technical Report, Aarhus University, Aarhus, Denmark, 1981.

[5] Strachey C. *Fundamental Concepts in Programming Languages*. Higher Order and Symbolic Computation, Abril, 2000.