1
Introdução

"*Software is hard* ".(D.E. Knuth)

Após mais de 50 anos de estudos, durante os quais ocorreram evoluções consideráveis nas técnicas para o desenvolvimento de programas, assim como surgiram linguagens de programação de "mais" alto nível, ainda hoje é um consenso que o processo de programação não é trivial. As principais dificuldades estão relacionadas à ausência de segurança no que tange a confiabilidade dos programas, e pode ser testemunhada por alguns acidentes causados por falhas associadas a problemas apresentados em programas:

✠ Em Julho de 1996 - o foguete europeu Ariane-5 explodiu 40 segundos após o lançamento de seu vôo inaugural; isso ocorreu devido a falta de um tratador de exceção ($overflow$) para o caso de o número ponto-flutuante ultrapassar 2^{15} (valores representados por 16 bits). Essa ausência fez com que o programa entrasse em colapso.[1]

✠ Recentemente, Janeiro de 2004 - O robô americano *Spirit*, enviado à Marte com objetivo de fotografar este planeta, perdeu contato com a Terra devido a problemas de gerenciamento de memória.[2]

Além desses, pode-se citar problemas em sistemas bancários, vírus que invadem os sistemas por suas falhas, etc.

Mas, por que ocorrem essas falhas nos programas?

A resposta é que, apesar da informática ser considerada uma ciência exata [9], o processo de desenvolvimento de programas, em sua maior parte, apresenta um caráter experimental.

PUC-Rio - Certificação Digital Nº 9924916/CB

Do ponto de vista histórico, os fundamentos da informática foram desenvolvidos por matemáticos, que podem ser considerados como os primeiros programadores, e além disso, a programação pode ser entendida como uma manipulação formal de símbolos muito semelhante às operações matemáticas.

Apesar disso, o processo de desenvolvimento de programas possui um caráter experimental, baseado mais em estudos de casos do que em raciocínio sobre os problemas. Isso se deve ao fato de que na maioria dos casos é fácil verificar o resultado de um programa desta forma e os testes aparentam ser um método mais rápido de desenvolvimento. Mas sabe-se que isso não passa de ilusão, visto que, o domínio de um programa é, quase sempre, infinito.

Os matemáticos se convencem da validade de seus trabalhos pelo rigor em que sua argumentação é realizada e por sua verificação. E esse processo é realizado através de provas, que servem como uma forma de comunicação, onde a partir de dados verdadeiros se tenta extrair novos fatos.

Considerando, por exemplo, os trabalhos de Knuth [32, 33, 35], Floyd[7] e Hoare[9], que fornecem programas e suas provas com o mesmo rigor das provas matemáticas, o trabalho de Kolmogorov [38], que apresenta a identificação de provas com algoritmos, e o trabalho de Howard[13], que propõe um isomorfismo entre as regras de inferência em Dedução Natural e as regras de formação de termos em λ-calculus, pode-se concluir que existe pelo menos um elo unindo a matemática ao processo de desenvolvimento de programas.

Esse elo pode ser responsável pela construção de um mundo quase perfeito para a informática, pois se for possível automatizar a construção desse elo, será amenizado o problema da ausência de segurança em relação à confiabilidade do programa, mas também estarão sendo resolvidos os problemas relacionados ao custo de desenvolvimento, manutenção e adaptação dos programas [22].

Existe uma área da computação denominada Métodos Formais que pesquisa maneiras de formar e construir esse elo, entre a matemática e o desenvolvimento de programas, e mesmo no caso de não conseguir encontrá-lo, visa propor ao menos algumas alternativas que amenizem os problemas

PUC-Rio - Certificação Digital Nº 9924916/CB

citados anteriomente.

Muitas pesquisas têm sido realizadas nesta área, voltadas para a verificação da correção de um programa que se baseia na especificação matemática do programa, seguido da sua prova de correção, e que são, inclusive, utilizados industrialmente como KIV[3], VDM [43], SPIN[20], SMV[4], etc.

Além dessas pesquisas, que partem de um programa já desenvolvido, existem outras que propõem a construção automática de um programa correto a partir das especificações dos problemas utilizando lógica [12, 17, 21, 23, 25, 28]. Este método, denominado síntese de programas, evita o duplo trabalho no processo de desenvolvimento de programas: a implementação do sistema e a verificação do programa, visto que ambos podem ser considerados como o mesmo processo de programação em diferentes níveis de formalização.

Há três categorias básicas de síntese de programas:

◇ Síntese construtiva de programas (*proof-as-program*)[15]: Neste método, a derivação de um programa é entendida como um processo de prova que constroi implicitamente um algoritmo. Este se baseia em uma prova construtiva e em um processo para extrair o programa da prova.

◇ Síntese transformacional ou Síntese de programas dedutivos [19]: Este método utiliza técnicas de reescrita, que serão executadas através de um conjunto de equivalências entre os predicados e os axiomas, com o objetivo de transformar uma especificação, que não é necessariamente executável, em um programa.

◇ Síntese baseada em conhecimento ou Síntese de programas indutivos [22]: Neste método existe uma cooperação entre o desenvolvedor e o sistema de síntese. O primeiro guia o processo de derivação através de estratégias de decisão, o que requer o entendimento do programa e do conhecimento de suas soluções. O sistema de síntese fornece o raciocício formal para garantir a correção do programa gerado [19].

Apesar de existirem vários métodos de síntese de programas, este trabalho apresenta um processo de síntese construtiva de programas cuja

PUC-Rio - Certificação Digital Nº 9924916/CB

motivação se segue na próxima seção.

1.1 Motivação

Considerando que:

1. - desenvolver um programa e provar que ele é correto são somente dois aspectos do mesmo problema [7],

2. - a prova para uma aplicação pode ser considerada como uma solução (construtiva) para um problema [14], e

3. - o programa pode ser extraído a partir de uma prova (construtiva) da existência de uma solução para o problema correspondente [17];

utilizando provas formais - como um método de raciocínio em relação à especificação - e fornecendo um processo de extração do programa que preserve a semântica da prova, consegue-se construir, a partir de uma especificação matemática, um programa que é correto por construção.

Dado que a especificação do problema e as regras dedutivas fornecem informações sobre as estruturas do algoritmo e a argumentação a respeito das propriedades, existem muitos cálculos da lógica formal para representar essas informações adequadamente; por exemplo: ITT (*Intuitionist type theory*) e GPT (*General problem theory*) [14, 24, 39]).
Como é utilizado o GPT no processo de síntese proposto, segue uma breve apresentação sobre o mesmo.

A descrição de um problema em lógica de predicados pode ser vista dentro do escopo do GPT, que é capaz de descrever os dados de entrada, os de saída e a relação entre eles. Esta teoria considera o problema como estruturas matemáticas, nas quais as soluções podem ser tratadas com exatidão e fornecem uma estrutura para o estudo de algumas situações a partir das quais os problemas são resolvidos. Entretanto, estas informações não são suficientes para assegurar a existência de um método que resolva o problema. Se, além da especificação em lógica de predicados e da sentença que descreve o problema ser um teorema da especificação, for obtida uma prova construtiva para o teorema, será possível compreendê-la não só como uma transcrição sintática, mas também como uma descrição de um dado

PUC-Rio - Certificação Digital Nº 9924916/CB

objeto, em outras palavras, a descrição de um algoritmo [42].

O isomorfismo de Curry-Howard (C.H.) associa as regras de inferência em dedução natural para lógica intuicionista (utilizada na prova) às regras de formação de λ-termos (comandos em uma linguagem de programação), de tal modo que uma prova de σ (uma fórmula) pode ser vista como um λ-termo do tipo σ. Desta forma, pode-se dizer que a prova possui uma interpretação computacional, ou seja, ela pode ser entendida como um conjunto de comandos em uma linguagem de programação, ou seja, um programa [13].

Este isomorfismo fornece a base para o processo de construção de programas a partir de provas, o que é geralmente denominado "extração do conteúdo computacional de uma prova". Ele extrai uma função que relaciona as entradas com as saídas específicas do programa. As entradas e as saídas do programa refletem a aplicação das regras de inferência utilizadas para obter a fórmula.

Pelo isomorfismo de C.H., as variáveis de entrada e de saída do programa são representadas, respectivamente, pelas variáveis quantificadas pelos quantificadores universal e existencial. Portanto, o teorema da especificação deve ser da forma $\forall x_1 \ldots \forall x_n \exists y P(x_1, \ldots, x_n, y)$.

Existem muitas propostas para o processo de síntese construtiva de programas que utilizam a lógica construtiva. Estes sintetizadores de programas, em sua grande parte, especificam os problemas em ITT, utilizam como sistema dedutivo: o cálculo de seqüentes [12, 17, 28] ou mecanismo de reescrita [21], e seus programas são gerados em linguagem de programação lógica ou funcional.

Considerando esses fatos, foi realizado o trabalho [26] que propõe um sintetizador construtivo, no qual o programa em linguagem imperativa (similar ao *Pascal*) é gerado a partir de uma prova utilizando dedução natural, evitando a conversão que é usada nos trabalhos afins encontrados na literatura, visto que, o isomorfismo C.H. é apresentado entre as provas em dedução natural e λ-termos. Neste método, as provas são apresentadas em lógica de predicados intuicionista poli-sortida e, utilizando o conceito de conteúdo semi-computacional de uma fórmula, é comprovado que o programa gerado é uma representação para a solução do problema especificado pelo teorema, na respectiva teoria que descreve os tipos de dados utilizados

PUC-Rio - Certificação Digital Nº 9924916/CB

pelo problema.

Entretanto, esse processo de síntese apresenta restrições no uso das regras de eliminação do quantificador existencial e de introdução da negação.

Este trabalho discute as restrições citadas, procurando, senão resolver, ao menos apresentar respostas parciais. Algumas alterações são propostas no método de síntese apresentado em [26] de forma a eliminar a restrição sobre a regra de eliminação do quantificador existencial através do conceito de modularização de provas e de programas [48, 27]. Em seguida, o conteúdo semi-computacional da regra do $\perp$ intuicionista é analisado durante a sua aplicação no processo de geração de programas. A análise é realizada observando o conteúdo computacional desta regra nas provas das fórmulas que representam as funções recursivas totais. Entretanto, para realizar essas provas, é necessário inserir no sistema formal a regra ω na sua versão computacional. Além da prova de correção, como efeito deste procedimento foi obtida a completude do método de síntese de programas para a aritmética de Heyting (HA). Concluindo, é abordada a possibilidade da utilização da regra da indução finita no lugar da regra ω computacional para as provas realizadas em HA com o princípio de reflexão.

Na seção a seguir é apresentada a organização deste trabalho.

1.2 Organização da Tese

No capítulo 2 apresentam-se os conceitos básicos necessários para a compreensão dos resultados. Ele está dividido em duas seções, a primeira que traz um resumo do trabalho sobre síntese construtiva de programas a partir do qual se originou essa tese [47], sendo esta seguida de uma breve explicação sobre a definição do predicado de provabilidade.

No capítulo 3, é dado prosseguimento ao estudo sobre síntese construtiva de programas, onde é proposto um processo que não possui a restrição sobre a regra de eliminação do existêncial, que agora pode ter um conteúdo computacional qualquer, sua aplicação era restrita às fórmulas utilizadas na regra da indução, sobre a qual sabe-se o seu conteúdo computacional. Este processo de síntese utiliza a noção de provas e programas modularizados. São apresentadas as modificações que devem ser realizadas no processo de

PUC-Rio - Certificação Digital Nº 9924916/CB

extração do conteúdo computacional apresentado no capítulo 2, bem como a prova de correção do sistema, mencionando apenas as regras que foram modificadas. As provas para as regras que não tiveram sua interpretação modificada estão presentes no apêndice.

No capítulo 4, é realizado um estudo sobre o conteúdo computacional[1] da regra do $\perp$ - que é iniciado com a possibilidade de que, para todo teorema da forma $\forall x_1 \ldots \forall x_n \exists y \alpha(x_1, \ldots, x_n, y)$, exista uma prova na qual a regra do $\perp$ sempre apresenta conteudo lógico - que é realizado a partir de conceitos de computabilidade (funções recursivas) e a utilização da regra ω. Considerando as provas[2] que utilizam a regra ω computacional, tem-se a prova que estas poderiam ser provadas com indução finitária mantendo a propriedade de possuir a regra do $\perp$ sem conteúdo computacional.

E, finalmente, após a apresentação dos estudos realizados, são apresentados no capítulo 5 alguns exemplos do processo de síntese de programas. As conclusões e as futuras questões a serem estudadas a partir deste trabalho estão presentes no capítulo 6.

Este trabalho também contém um apêndice que além da apresentação da prova de correção para as regras que não foram incluídas no capítulo 3, possui também a apresentação da prova do teorema apresentado por Parik em [11] e a prova do lema Kreisel e Wang em [6] que foram utilizadas no capítulo 4.

PUC-Rio - Certificação Digital Nº 9924916/CB

[1] No contexto da arimética de Heyting.
[2] Realizadas na aritmética de Heyting com o princípio de reflexão.