VIII - Conclusão

8.1. Trabalhos Futuros

- Existem na literatura outros modelos categóricos de linguagens livre de contexto, como [Wells e Barr, 1988]. Seria interessante estender esse modelo para traduções entre linguagens de programação e comparar o resultado com o modelo apresentado nesta tese.
- O conceito de semântica de árvores de derivação pode ser utilizado em outras áreas relacionadas a linguagens de programação, tais como:
  - Responder a perguntas já abordadas, sem sucesso, por alguns pesquisadores, tais como:
    - O que significa uma linguagem expressar uma certa construção e outra não [Baliga e Schende, 1994];
    - O que é o nível (a expressividade) de uma linguagem [Halstead, 1977].
  - Ajudar a desenvolver critérios para a avaliação e desenvolvimento de padrões de projeto.
  - Avaliar projetos de sistemas, onde sua qualidade seja medida em função de sua adequação para mudanças.
- Este trabalho pode prosseguir em direções voltadas à prática, tais como:
  - Desenvolver ferramentas de suporte à construção de tradutores.
  - Aprofundar a análise comparativa de linguagens de programação.
  - Analisar ferramentas de tradução existentes.

- Trabalhos futuros também podem estender os modelos apresentados para traduções que dependem do contexto, por exemplo, de dados do ambiente. O exemplo 8.1 deixa claro esta necessidade.

Exemplo 8.1. Tradução de **record** variante.

Linguagem de entrada: Pascal.

Linguagem de saída: C.

Tradutores: P2C

Comentários: O P2C faz uma tradução para C padrão, onde as **structs** são obrigadas a ter um nome. A tradução é bastante intuitiva e não compromete a manutenibilidade. À primeira vista, parece preservar a semântica de árvores de derivação, mas este não é o caso. Considere a semântica da derivação ⟦`varaccess ⇒ fator.ID`⟧; não existe uma árvore na linguagem de saída com o mesmo significado. Este é um exemplo típico de tradução sensível ao contexto. O acesso a um campo do **record** é traduzido de forma diferente se fizer parte do **case**, pois este é traduzido para um **struct** que precisa ser nomeado.

PASCAL	C
`Type`	
`  Classe  = (Num, Str);`	`typedef enum {NUM,STR} CLASSE;`
`  Facts = record`	`typedef struct {`
`    Nome: string[10];`	`   CHAR nome[11];`
`    Case Tipo: Classe of`	`   CLASSE tipo;`
	`   union {`
`      Num: (N: real);`	`      DOUBLE n;`
`      Str: (S: string);`	`      STRING s;`
`  End;`	`   } u_n;`
`Var`	`} FACTS;`
`  Fato: Facts;`	`FACTS fato;`
`...`	`...`
	`p2c_init(NULL,2048,0);`
`  Fato.Nome := 'Alo';`	`strasg(fato.nome,"Alo",sizeof(fato.nome));`
`  Fato.Tipo := Num;`	`fato.tipo=NUM;`
`  Fato.S    := 'Oi';`	`strasg(fato.u_n.s,"Oi",sizeof(fato.u_n.s));`
`  Case (fato.Tipo) of`	`switch ((fato.tipo)) {`
`    Num:`	`     case NUM:`
`       fato.S := fato.S + ' !';`	`        strasg(fato.u_n.s,strsum(2,fato.u_n.s,"!"),sizeof(fato.u_n.s));`

	break;
Str:	case STR:
inc(fato.N);	(fato.u_n.n)++;
End;	break;
	}

Figura 8.1 Tradução de *records* variantes no P2C.

8.2. Contribuições

A partir da experiência adquirida na implementação de tradução entre linguagens de transformação usando ferramentas de tradução, ficou claro que o projeto das traduções exigia um considerável esforço criativo por parte dos projetistas e que não existia nenhum critério formal para ajudá-los a avaliar a qualidade dessas traduções. Como mencionado no capítulo 1, iniciamos o presente trabalho visando a executar um programa de dois pontos:

- Criar um modelo para tradução entre linguagens de programação.
- Utilizá-lo para formalizar nossa compreensão intuitiva do que é uma boa tradução.

O primeiro objetivo é atingido nos modelos de tradução apresentados no capítulo 6. Em particular são apresentados os seguintes resultados: um modelo categórico correto e completo para ETDS simples; e um modelo categórico correto e completo para ETDS em geral.

A utilização da teoria das categorias foi particularmente proveitosa para a criação dos diversos modelos de tradução que são homomorfismos entre estruturas diferentes (árvores de derivação, DAGs de derivação e, finalmente árvores de derivação não ordenadas). Estes homomorfismos surgem naturalmente como os morfismos das categorias apresentadas.

Quanto ao segundo objetivo, concentramo-nos na investigação de uma boa tradução no que diz respeito à *manutenibilidade*. Este objetivo é atingido no capítulo 4, quando introduzimos o conceito de semântica de árvores; no capítulo 5, quando

apresentamos argumentos da relação deste conceito com a manutenibilidade dos programas, e no capítulo 6, quando utilizamos o modelo de tradução entre linguagens de programação para formalizar a preservação da semântica de árvores.

Além de atingir esses objetivos, acreditamos que a técnica de semântica de árvores traz um maior entendimento da semântica de linguagens de programação e da forma como a intenção do programador está relacionada com a estrutura sintática de um programa. Este entendimento pode contribuir na abordagem de outras questões na área de linguagens de programação (mencionadas nas seções 5.3 e 8.1), que transcendem os objetivos imediatos desta tese.