Trabalho Prático 3
Segunda, Maio 7, 2018

1 Introdução

Neste trabalho você irá escrever um parser para Cool. O trabalho utilizará duas ferramentas: o gerador de parser bison e um pacote de manipulação de árvores. A saída do seu parser será uma árvore sintática abstrata (Abstract Syntax Tree/AST). Você irá construir esta AST usando ações semânticas do gerador de parser.

Você irá precisar consultar a estrutura sintática de Cool, que pode ser encontrada na figura 1 do manual do Cool. A documentação para o pacote de manipulação de árvores está no "Tour of Cool Support Code". Esse pacote será usado neste e em futuros trabalhos práticos.

Há muita informação nesta especificação, e você irá utilizar a grande maioria delas para escrever um parser que funcione. *Por favor leia a especificação por inteira*

2 Arquivos e diretórios

No Moodle podem ser encontrados os arquivos necessários para o trabalho em "Arquivos TP", na pasta assignments/PA3.

O arquivo mais importante nesta pasta é o cool.y. Este arquivo contém o ponta-pé inicial de um parser para Cool. A seção de declarações está quase completa, mas será necessário adicionar declarações adicionais de tipos para os novos não-terminais que você introduzir. Nós já colocamos os nomes e as declarações de tipos para os terminais. Você pode precisar também adicionar declarações de precedência. A seção de regras, entretanto, está praticamente incompleta. Nós provemos algumas partes de algumas regras. Não é necessário modificar este código para ter uma solução correta, mas, caso queira, pode alterar. Todavia, não assuma que quaquer regra em particular esteja completa.

3 Testando o Parser

Você executará seu parser utilizando myparser, um comando shell que 'integra' o parser com o scanner. Perceba que myparser recebe uma flag -p para depurar o parser; ao usar esta flag várias informações sobre o que o parser está fazendo irão ser exibidas na saída padrao `stdout`. O Bison produz uma saída legível das tabelas de parsing LALR(1) no arquivo cool.output. Verificar este arquivo pode ser útil para depurar o seu parser.

Você deve executar

```
make
```

para obter as outras fases do compilador e

```
make parser
```

para compilar o parser.

O script mycoolc corresponde ao coolc utilizando o parser que vocês criaram. Se seu parser estiver correto, o comportamento do mycoolc deveria ser similar ao do coolc.

Seu parser será avaliado utilizando o analizador léxico utilizado pelo coolc.

4 Saída do Parser

Suas ações semânticas devem construir uma AST. A raíz (e somente a raíz) da AST deve ser do tipo `program`. Para programas que forem analisados com sucesso, a saída do `parser` deve ser uma listagem da AST.

Para programas com erros, a saída são as mensagens de erro do parser. Nós fornecemos uma rotina de exibição de erros, que printa as mensagens de erro em um formato padrão; por favor não modifique-a. Você não deve invocar esta rotina diretamente nas suas ações semânticas; bison irá invocar ela automaticamente quando um problema for identificado.

Você não precisa lidar com programas que contenham mais de um arquivo.

5 Erros

Você deve usar o pseudo-não-terminal `error` para lidar com erros. O propósito de `error` é permitir que o parser continue após antecipar um erro. Veja a documentação do bison para mais detalhes sobre como utilizá-lo corretamente.

Seu parser deve ser capaz de recuperar das seguintes situações:

- Se há um erro em uma definição de classe, mas a classe é finalizada corretamente, o parser deve ser capaz de processar o resto do arquivo.
- Similarmente, o parser deve ser capaz de recuperar de erros em features (indo para a próxima feature), `let` bindings (indo para a próxima variável), e expressões dentro de um bloco {} (indo para a próxima expressão).

Não se preocupe muito com o número da linha que irá aparecer nas mensagens de erro geradas pelo seu parser. Se seu parser está funcionando corretamente, o número da linha geralmente será onde o erro ocorreu. Para construções errôneas em múltiplas linhas, o número da linha provavelmente vai ser o da última linha da construção.

6 O pacote Tree

Há uma discussão extensa do pacote de árvores para ASTs em Cool em *Tour of Cool Support Code*.

7 Considerações

- Você pode usar declarações de precedência, mas apenas para expressões. Não use declarações de precedências cegamente (não responda a um conflito de shift-reduce na sua gramática adicionando regras de precedências até que os conflitos desapareçam).
- A construção `let` de Cool introduz uma ambiguidade na linguagem. O manual resolve a ambiguidade dizendo que a expressão `let` se extende o máximo possível para a direita. A ambiguidade pode acabar aparecendo no seu parser como um conflito shift-reduce envolvendo produções para `let`.

  Uma possível solução é usar uma funcionalidade do bison que permite que precedência seja associada com produções (e não somente operadores).

- Você deve declarar bison "types" para seus não-terminais e terminais que tiverem atributos. Por exemplo, em cool.y está a declaração:

  ```
  %type <program> program
  ```

  Esta declaração diz que o não-terminal `program` tem tipo `<program>`. O uso da palavra "type" é enganador; o que ela realmente significa é que o atributo para o não-terminal `program` é salvo no membro `program` da `union` declarada em cool.y, que tem tipo Program.

  Ao especificar o tipo

  ```
  %type <member_name> X Y Z ...
  ```

  você instrui o **bison** que os atributos dos não-terminais (ou terminais) `X`, `Y`, e `Z` tem um tipo apropriado para o membro `member_name` da union.

  Todos os membros da union e seus respectivos tipos possuem nomes similares por padrão. É uma coincidência no exemplo acima que o não-terminal `program` tem o mesmo nome que um membro da union.

- É crítico que você declare os tipos corretos para os atributos dos símbolos da gramática; o contrário praticamente garante que seu parser não irá funcionar. Você não precisa declarar tipos para os símbolos da sua gramática que não possuam atributos.

- O verificador de tipos do g++ pode gerar warnings se você usar os construtores das árvores com os parâmetros com tipos incorretos. Se você ignorar essas warnings, seu programa pode crashar quando o construtor perceber que está sendo usado incorretamente. O bison também pode gerar warnings se detectar erros de tipo. Tenha cuidado com essas warnings. Não se surpreenda se seu programa crashar inesperadamente nesses casos.

- Executar o bison com o cool.y inicial fornecido irá produzir algumas warnings sobre "useless noterminals" e "useless rules". Isso acontece porque alguns não-terminais e regras nunca serão utilizados. Porém, essas warnings não deveriam estar presentes no parser final.

8 Submissão

Deverá ser enviado somente o arquivo cool.y