Trabalho Final INF1022 2021-2

Edward Haeusler
Monitor Bernardo Alkmim

10 de novembro de 2021

1 Introdução

Conforme você deve ter visto na lista 9, é muito complicado lidar com erros que dizem apenas **Encontrei um erro**. Pensando como o usuário/programador da linguagem em si, como que você conserta um programa se o erro que você recebe é simplesmente esse?

Qualquer compilador *profissional*, digamos, tem um tratamento interno de erros bem esperto, para dar ao usuário um feedback útil para corrigir seus programas e adequá-los à gramática da linguagem que ele está utilizando. Muitos dos erros ocorrem a nível de sintaxe, e são pegos justamente pelo analisador sintático (sem mencionar ainda erros que ocorrem a nível léxico).[1]

Em [1],[2] Traver reúne 8 características principais que uma boa mensagem de erro de compilador deve conter:

1. Clarity and brevity (aesthetic and minimalist design, recognition rather than recall)
2. Specificity (recognition rather than recall; help user recognize, diagnose and recover from errors)
3. Context-insensitivity (consistency and standards)
4. Locality (flexibility and efficiency of use)
5. Proper phrasing (match between system and the real world)
6. Consistency (consistency and standards)
7. Suitable visual design (aesthetic and minimalist design; error prevention)
8. Extensible help (help and documentation)

Nesse mesmo artigo o autor explica com exemplos e comparações de mensagens de erro como essas propriedades podem ser aplicadas - especialmente no apêndice.

[1] Existem outros tipos de erro, mas esses são os que nos interessam no momento.

[2] O artigo é de fácil acesso e pode ser encontrado na íntegra online, bastando buscar pelo nome.

2 Enunciado

Faça (sozinho ou em dupla) um analisador sintático para a linguagem Provol-One, definida pela gramática a seguir, criando mensagens de erro úteis para o usuário, indicando **onde** ocorreu o erro dele, além de indicar o **contexto do erro**, ou seja, indique o que a gramática esperava como próximo token e o que foi recebido no lugar, além de indicar qual parte da gramática que encontrou o erro - foi um erro na hora de atribuir algo? De fazer uma comparação? - e qual o nível do erro - é léxico? Sintático? Semântico?[3] O artigo [1] pode ajudar a compreender melhor como uma mensagem de erro deve parecer.

Seu analisador deve compilar programas em Provol-One para uma linguagem à sua escolha, seja C, Python, Lua etc. Com isso, temos: em caso de aceitação do programa pelo compilador, a saída deve ser um programa equivalente em outra linguagem - como Provol-One é um fragmento pequeno englobado por todas essas linguagens, a compilação em si será bem direta. Em caso de não aceitação, a saída deve ser uma mensagem de erro bem elaborada.

A linguagem Provol-One possui somente os comandos de repetição indefinida (Enquanto X Faça C Fim) e atribuição. Provol-One tem somente tipo numérico inteiro não negativo.[4] Deve-se usar um gerador de analisador sintático que implemente o método LaLR(1) ou outro ascendente, por exemplo, Yacc/Lex, Bison/Flex, JavaCC (que usa o descendente LL(1)), etc.

2.1 Desenvolvimento

A sintaxe da linguagem Provol-One é dada pela gramática abaixo:

$$
\begin{array}{lcl}
program & \longrightarrow & ENTRADA\ varlist\ SAIDA\ varlist\ cmds\ FIM \\
varlist & \longrightarrow & id\ varlist \mid id \\
cmds & \longrightarrow & cmd\ cmds \mid cmd \\
cmd & \longrightarrow & ENQUANTO\ id\ FACA\ cmds\ FIM \\
cmd & \longrightarrow & id = id \mid INC(id) \mid ZERA(id)
\end{array}
$$

Todas as variáveis são do tipo número natural (inteiro não-negativo). O comando $Inc(id)$ incrementa em um o conteúdo da variável id, $Zera(id)$ faz o conteúdo da variável id ser 0 (zero), $id = id$ copia o conteúdo da variável a direita na variável da esquerda, i.e. é um comando de atribuição. Os booleanos, usados nos comandos *Enquanto* e de desvio condicional ($Se - Entao$ e $Se - Entao - Senao$), são relacionados aos valores numéricos na forma: $falso$ é 0 (zero) e *verdadeiro* é qualquer valor diferente de zero. O comando de repetição definido $Faca\ id\ vezes\ <cmds>\ FIM$ repete a execução de *cmds* o número de vezes que for o valor de id. Este valor é constante e avaliado no ínicio da execução do comando $Faca$. Assim, um trecho de código na forma $Faca\ X\ vezes\ X = X + 3$ será executado o mesmo número de vezes que for

[3] Tratar erros semânticos (basicamente checagem de tipos) é opcional, mas muito bem-vindo.

[4] Caso queira brincar com tipos, uma sugestão é expandir essa parte.

o valor de X no início da execução do $Faca$. Por exemplo, se X tiver falor 5, este será o número de repetições de *cmds* para o trecho acima. Quando X tem valor zero, nenhuma execução de *cmds* é realizada. Se houver necessidade de fazer pequenas modificações na sintaxe de Provol-One, fiquem à vontade.

Um exemplo de programa em Provol-One é:

```
Program ENTRADA X, Y
        SAIDA  Z
        Z=Y
        ENQUANTO X FACA
           INC(Z)
        FIM
```

Como há possibilidades diferentes de erros a serem analisados, espera-se que você teste componentes diferentes da gramática para mostrar a robustez do seu sistema de mensagens de erro. Testes de aceitação também são bem-vindos.

2.2 Observação

Essa linguagem é bem restrita. Caso queira, pode expandir a gramática para permitir comandos adicionais ou demais operações que julgar interessantes. Sugerimos a inclusão dos comandos de seleção (if-then e if-then-else) comando de repetição definida (Faça cmds X vezes) etc.

Caso escolha expandir a linguagem, deixe explícitas as alterações realizadas bem como o modo que foram tratadas no relatório do trabalho.

3 Entrega

Você deve entregar um arquivo contendo seu relatório, no qual estarão descritos seu trabalho - o que foi implementado, como foi implementado, o que funciona, o que não funciona, quais os testes utilizados etc. - e quaisquer outras informações que você considere relevantes, como a maneira de executá-lo. Caso utilize alguma fonte na sua pesquisa do trabalho, indique também.

Além disso, entregue um .zip com os casos de teste utilizados.

Por fim, entregue seu analisador sintático - no caso de utilizar Flex e Bison, os arquivos .l e .y - bem como quaisquer outros arquivos/módulos auxiliares que tenha criado para a execução do trabalho.

Indique nome e matrícula de **ambos os componentes da dupla**, se for o caso.

Mande email indicando qual o dia e horário que vai querer preencher para apresentar o trabalho.

3.1 DATA DE ENTREGA

Mande seus arquivos no local apropriado no EAD até o dia 05-12-2021, às 23h59.

4 Dicas

- Espelhe-se nos compiladores que você já utiliza para seus programas em linguagens de programação usuais - o que eles geralmente apresentam nas mensagens de erro? O que você gostaria de acrescentar a eles?
- Comece pequeno: garanta que seu analisador funciona para um fragmento da linguagem, depois vá incrementando.
- É bom planejar uma suíte de testes com alguns testes de aceitação (e bastantes de não aceitação), organizados pela funcionalidade que eles testam. Com isso em mãos, passe a fazer testes mais elaborados que misturam funcionalidades já testadas - às vezes você não prevê como elas vão interagir.

Referências

[1] V. Javier Traver. On compiler error messages: What they say and what they mean. *Adv. in Hum.-Comp. Int.*, 2010, January 2010.