Investigando erros de compilação de alunos aprendendo Haskell*

Boldizs´ar N´emeth
E ¨otv¨os Lor´and University
nboldi@elte.hu

Eunjong Choi
choi@is.naist.jp

Erina Makihara
makihara.erina.lx0@is.naist.jp Instituto de Ciência e Tecnologia de Nara

Hajimu Iida
iida@itc.naist.jp

Embora a programação funcional seja uma maneira eficiente de expressar software complexo, as linguagens de programação funcional têm uma curva de aprendizado acentuada. Haskell pode ser difícil de aprender para alunos que só foram apresentados à programação imperativa. É importante buscar métodos e ferramentas que possam diminuir a dificuldade de aprendizagem da programação funcional. Encontrar métodos para ajudar os alunos requer entender os erros que os alunos cometem enquanto aprendem Haskell.

Existem vários estudos anteriores revelando dados sobre erros do compilador Haskell, mas eles não concentram-se na análise dos erros do compilador ou estudam apenas um certo tipo de erros do compilador.

Este estudo investiga erros de compilação de alunos iniciantes em Haskell e faz sugestões sobre como sua eficiência de aprendizado pode ser melhorada. Ao contrário dos estudos anteriores, nos concentramos em descobrir a raiz dos problemas com as soluções dos alunos, analisando amostras de seus envios.

1. Introdução

A linguagem de programação Haskell [1] é uma linguagem de programação avançada e puramente funcional com sistema de tipo estático e avaliação preguiçosa. Graças ao sistema de tipo expressivo de Haskell, muitos erros de programação são capturados pelo compilador. Foi relatado que linguagens de programação com sistemas de tipos fortes são mais resistentes a bugs [2].

Independentemente dos benefícios de usar Haskell, é considerada uma linguagem de programação difícil de aprender. As causas dessa crença podem estar relacionadas à sua verificação estrita de tipo, natureza funcional (difícil de entender para alunos que foram introduzidos apenas a linguagens de programação imperativas) e avaliação preguiçosa não intuitiva.

Nossa motivação é ajudar os alunos a superar o desafio de entender e resolver os erros do compilador. Para fazer isso, devemos entender que tipo de erros eles cometem. Este estudo investiga erros de programação que resultam em um erro de compilador do primeiro ano de estudantes de bacharelado da E ¨otv¨os Lor´and University durante um curso de Haskell.

Uma vez que os erros do compilador que surgem nas primeiras etapas do processo de compilação, nomeadamente erros léxicos e sintáticos, são mais fáceis de detectar, os dados quantitativos sobre esses erros são mais amplamente cobertos [3]. Nossa pesquisa se concentra em erros complexos encontrados em estágios posteriores da compilação. Nós nos concentramos nos erros encontrados com mais frequência pelos alunos, para fornecer soluções úteis para seus problemas.

ÿEste trabalho foi parcialmente financiado por uma bolsa de mobilidade do primeiro autor no Nara Institute of Science and Technology no âmbito do Erasmus Mundus Action 2 Project TEAM Technologies for Information and Communication Technologies, financiado pela Comissão Europeia. Esta publicação reflete apenas a visão dos autores, e a Comissão não pode ser responsabilizada por qualquer uso que possa ser feito das informações nela contidas.

P. Achten, H. Miller (Eds.):
Trends in Functional Programming in Education 2018 (TFPIE'18)
EPTCS 295, 2019, pp. 52–64, doi:10.4204/EPTCS.295.4

O objetivo deste estudo é analisar os erros de programação que os alunos cometem durante o aprendizado de Haskell. Para resolver a questão de por que Haskell é difícil de aprender, temos que focar nos obstáculos que os alunos encontram ao aprender o idioma e como eles influenciam os erros que os alunos cometem.
Como o aprendizado de uma linguagem de programação só é possível por meio da prática, é melhor inspecionar os resultados dos alunos em um exercício prático, no qual eles devem colocar seus conhecimentos da linguagem no código-fonte real.

Estamos interessados em fornecer resultados sobre como os alunos podem ser ajudados durante o estudo da programação funcional. As informações que encontramos nos erros de compilação dos alunos podem ser aplicadas em muitos caminhos.

- Os dados analisados são um feedback valioso para os educadores. Os resultados da nossa investigação podem ser aplicados a outros cursos com currículo semelhante. Ao identificar os tópicos mais difíceis para os alunos, é possível dedicar mais tempo a esses tópicos desafiadores ou adiá-los até que os alunos estejam mais familiarizados com os conceitos básicos da linguagem.
- Nossa pesquisa pode ajudar os desenvolvedores de compiladores a melhorar as mensagens de erro do compilador. Ao descrever quais erros são encontrados com mais frequência pelos alunos e quais deles são enganosos ou confusos para eles, podemos apontar mensagens de erro que podem ser melhoradas.
- Alguns dos problemas dos alunos podem ser superados com ferramentas mais sofisticadas. Nossa pesquisa pode mostrar quais problemas precisam de mais atenção e pode orientar pesquisas futuras sobre ferramentas automatizadas que, por exemplo, descrevam os erros do compilador para os alunos ou façam sugestões sobre como corrigi-los.

2. Trabalho relacionado

O processo de aprendizagem de programação funcional já foi examinado por diversos pesquisadores.

Singer e Archibald coletaram dados de um curso online sobre Haskell e identificaram alguns padrões comuns de erros sintáticos [3]. No entanto, os dados apresentados no artigo descrevem apenas as frequências de erros lexicais e sintáticos. Os erros encontrados em etapas posteriores da compilação não são analisados. Eles mencionam sua intenção de realizar uma análise em nível de tipo nos dados, mas até hoje não publicaram tais resultados. Eles concluem que ferramentas de programação mais ricas seriam benéficas para seus alunos.

Heeren et ai. apresentam um compilador Haskell alternativo chamado Helium [4], que é direcionado especificamente para estudantes que estão aprendendo a linguagem. Ele suporta um subconjunto da linguagem Haskell e fornece mensagens de erro melhores do que outros compiladores Haskell e correções automáticas para os alunos. O artigo deles contém uma análise dos tipos de erros ao longo de sete semanas. Sua abordagem de categorização de erros é semelhante à nossa, seus resultados sugerem que a principal causa de erros do compilador são erros relacionados ao tipo. No entanto, eles não oferecem mais detalhamento das categorias de erro.

Gerdes, e outros. descrevem um tutor interativo para Haskell [5]. Ele oferece suporte ao desenvolvimento interativo com lacunas e fornece conselhos específicos, combinando a solução do aluno com os exemplos anotados. Orientar o aluno a resolver o exercício passo a passo tem uma vantagem sobre simplesmente dar-lhe tarefas para fazer.

Pettit e Gee relatam em seu artigo que simplesmente criar mensagens de erro mais úteis e personalizadas para certos erros comuns do programador não tornará os alunos menos propensos a repetir os mesmos erros [6], mas seus resultados não são conclusivos. Em um artigo recente sobre ferramentas de análise estática, Barik argumenta que essas ferramentas devem explicar as etapas que executaram [7]. Um estudo com rastreamento ocular sustenta que a compreensão das mensagens de erro é vital para o sucesso de uma tarefa de programação e tão difícil quanto a leitura do código-fonte [8].

3 Método de análise

3.1 O conjunto de dados analisado

Investigamos erros de alunos em um curso de um semestre sobre programação funcional. Mais de 120 alunos do primeiro ano de graduação da E ¨otv¨os Lor´and University participaram do curso. Eles só têm experiência básica de programação imperativa em uma linguagem de programação estaticamente tipada. O curso de programação funcional consiste em 12 palestras e sessões práticas onde os alunos usam Haskell para resolver exercícios por cerca de 60-70 minutos por vez, com trabalhos de casa e tarefas entre as aulas. Os exercícios para as sessões práticas são curtos, normalmente exigindo que o aluno escreva a implementação de uma função (1-5 linhas) dada a assinatura de tipo. Para as sessões práticas, os alunos são divididos em grupos de 20. Dentre as 12 semanas do curso, analisamos nove semanas (da segunda à décima) em que a frequência foi alta o suficiente para oferecer uma amostra representativa.

O currículo do curso abrange os fundamentos da linguagem Haskell. As sessões práticas começam com aritmética inteira e booleana. Depois disso, as listas são introduzidas incluindo a sintaxe de compreensão de lista. Isso é seguido por definições de função, correspondência de padrão com foco na recursão e uso de guardas de padrão. No final do semestre, as funções de ordem superior são introduzidas, concentrando-se nas funções de processamento de listas, como a dobragem. A programação precisa desses tópicos variou entre os diferentes grupos.

O conjunto de dados usado neste estudo é registrado em um site interativo (versão em inglês acessível [9]) que os alunos usam para resolver exercícios e tarefas. Não apenas os exercícios concluídos são registrados, mas também cada passo que o aluno deu em direção à apresentação final.

Para facilitar o início dos exercícios, os alunos enviaram fragmentos de código em vez de módulos Haskell completos. O restante do módulo foi gerado para eles, incluindo assinaturas de tipo para as funções que eles tiveram que escrever. Isso é um pouco limitante, por exemplo, os alunos não puderam adicionar uma nova declaração de importação ao módulo, mas fornecemos a eles uma versão modificada do Haskell Prelude padrão (as partes acessíveis automaticamente da biblioteca padrão) e módulos adicionais onde o exercícios necessários, incluindo-os.

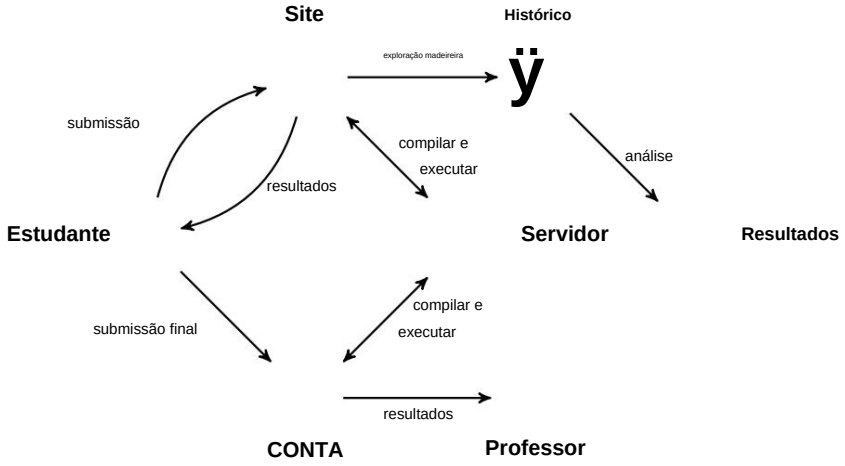
O site não requer a criação de contas para usá-lo. Como os alunos usam computadores diferentes cada vez que acessam o site, a falta de contas impossibilita o acompanhamento do desempenho de cada aluno ao longo do semestre. Felizmente, não está no escopo de nossa pesquisa. Aproveitando a forma como o site mantém as sessões dos alunos, as interações de um único aluno com o sistema podem ser lembradas durante uma única aula. Os timestamps das interações são registrados, estamos analisando quanto tempo leva para os alunos corrigirem determinados erros. Estamos usando as sequências de interações com registro de data e hora para analisar as reações dos alunos às mensagens de erro. O conjunto de dados é anônimo, nenhuma informação pessoal está presente.

A infraestrutura usada para ensinar Haskell inclui um ambiente de teste automatizado. As submissões dos alunos foram compiladas usando o Glasgow Haskell Compiler [10] (GHC), versão 8.0.2. O sistema sabe a quais exercícios fazem os envios associados. Nem todos os exercícios são testados, mas erros de compilação são relatados para cada um. As respostas automatizadas sobre os erros de compilação e os resultados dos testes do servidor também são registrados.

A Figura 1 apresenta o cenário em que os alunos experimentaram e apresentaram suas soluções. Eles estão usando o site interativo para desenvolver suas soluções para os exercícios. Seus envios são compilados e executados, comparando a saída com os resultados esperados. Cada compilação é uma solução separada, portanto, um aluno pode facilmente gerar de 30 a 60 soluções durante uma única sessão. Os arquivos de log gerados pelo site interativo são analisados neste artigo. Os alunos enviam suas submissões finais como trabalhos de casa e

assinaturas para um sistema diferente (BEAD), mas isso não faz parte do nosso conjunto de dados. O site registra os envios dos alunos e as respostas do servidor (os resultados da compilação e execução da solução).

Figura 1: Coleta de dados para o estudo



3.2 Etapas da investigação

Para resumir nossa análise do conjunto de dados extraído, apresentamos em quatro etapas:

1. Analisamos como a taxa de sucesso das submissões dos alunos muda ao longo do semestre.
2. Atribuímos erros do compilador em categorias com base nas mensagens de erro e verificamos como a proporção de os erros nessas categorias mudam ao longo do semestre.
3. Ao analisar as sequências de envios dos alunos, medimos quanto tempo leva para o alunos para corrigir diferentes tipos de erros do compilador.
4. Amostramos todo o conjunto de dados para determinar as causas principais dos erros de nível superior mais comuns.

Em alguns casos, as mensagens de erro do GHC não revelam a causa real do erro. Também amostramos os erros do compilador para detectar e categorizar as causas principais. Percebemos que a causa de algumas das mensagens de erro das fases posteriores das compilações se originam de erros sintáticos por parte dos alunos. Nós rastreamos como a frequência dos problemas causados sintaticamente muda ao longo do semestre.

Como comparação com nosso conjunto de dados, executamos a análise no conjunto de dados usado por Singer e Archibald em seu artigo Functional Baby Talk (conjunto de dados FBT a seguir) [3]. O conjunto de dados deles é coletado durante um curso online, então complementa nosso conjunto de dados que é coletado durante uma aula universitária tradicional.

4 Resultado da investigação

A Tabela 1 e a Figura 2 mostram a frequência de envios de alunos resultando em erros de compilador, erros de tempo de execução, bem como resultados errados e resultados corretos. (“Não testado” significa que a solução compilou, mas não há nenhum teste associado ao exercício. “Correto” significa que a solução passou nos testes do exercício.) Os resultados foram obtidos classificando mecanicamente a saída do compilador. Na Tabela 1 podemos observar que o número de envios teve seu pico na última semana do semestre. No entanto, a Figura 2 mostra,

Figura 2: Resultados das soluções dos alunos

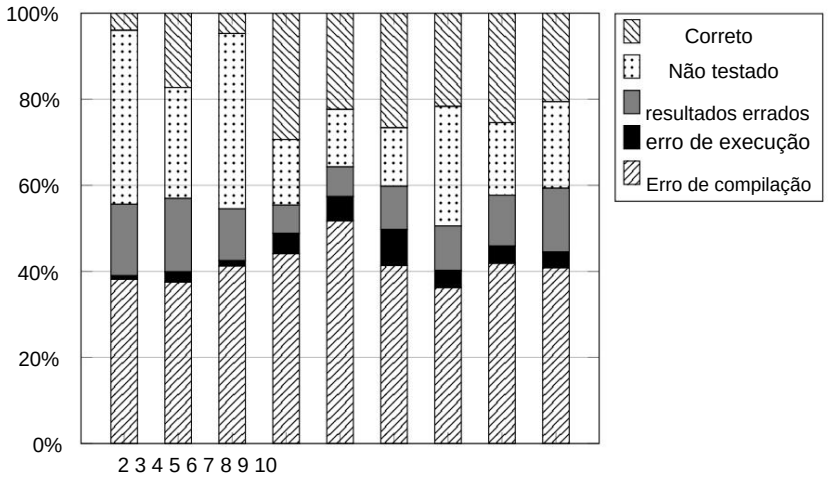


Tabela 1: Resultados das soluções dos alunos

Semana	Compilar erros	Tempo de execução erro	Non ter minação	resultados errados	Correto	Não é um teste
2	2.195	36	11	954	228	2.324
3	1.513	81	15	688	696	1.040
4	1.423	36	6	415	163	1.405
	743	79		110	494	257
5 6	1.116	121	1 0	149	482	288
7	930	188	1	225	598	307
	1.610	178	0	458	961	1.235
8 9	2.801	264	4	790	1.701	1.127
10	4.534	396	15	1.641	2.279	2.236

que a proporção de envios bem-sucedidos realmente não muda durante o curso do semestre. Isso implica que a dificuldade dos exercícios é compensada pelo aumento da proficiência do aluno em Haskell.

No conjunto de dados FBT, a taxa de envios corretos também é constante durante o curso, mas é muito maior (70% - 80% contra nossos 50% - 60%). Assumimos que isso é causado pelas diferentes dificuldades e estruturas dos exercícios. A atividade está diminuindo durante o curso, atingindo o pico na segunda semana e diminuindo constantemente para cerca de 5% desse pico na última semana no conjunto de dados FBT.
Comparado a isso, em nosso conjunto de dados, os picos de atividade nas últimas semanas, quando os alunos estão aprendendo para o exames.

Com base na fase de compilação em que são relatados, agrupamos os erros do compilador em três categorias distintas: erros sintáticos (lexical e relacionados à sintaxe) são encontrados no primeiro estágio da compilação, enquanto erros relacionados a nomes (nomes ausentes, colisões de nomes) e erros de verificação de tipo (erros encontrados durante a verificação de tipo) encontrados no segundo estágio principal da compilação. Observe que erros em estágios anteriores podem ocultar erros em estágios posteriores. As frequências dessas categorias de erro podem ser vistas na Tabela 2.

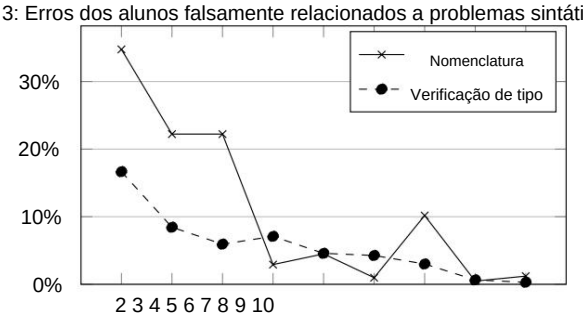
No entanto, descobrimos que nem todos os erros do compilador são categorizados corretamente por suas mensagens de erro. Devido à sintaxe simples e flexível de Haskell, alguns erros sintáticos que o aluno cometeu acidentalmente

Tabela 2: Categorias de erros dos alunos com base nas mensagens de erro

Semana	Analisar	Verificação do tipo de	
1	950	448	nomeação 971
2	507	458	694
3	627	324	536
4	228	289	301
	254	211	738
5 6	219	192	565
7	365	382	980
	594	631	1.729
8 9	982	1.045	2.851

resultaram em programas sintaticamente corretos. Nesses casos, o erro produziu um erro em um estágio posterior da compilação, produzindo uma mensagem de erro enganosa. Chamamos esses erros de falsos erros semânticos.

Figura 3: Erros dos alunos falsamente relacionados a problemas sintáticos



Para estimar a frequência de erros semânticos falsos, amostramos manualmente os erros dos estágios posteriores da compilação e os agrupamos em erros semânticos reais e erros semânticos falsos relacionados à sintaxe. Os resultados podem ser vistos na Figura 3 e na Tabela 3. É importante observar que a Tabela 2 contém os erros baseados apenas em sua mensagem de erro, os erros semânticos falsos não são removidos dos dados nela apresentados. Também é importante que a Tabela 1 sintetize as submissões enquanto a Tabela 2 e a Tabela 3 detalham o número de erros, pois mais de um erro pode aparecer em uma única submissão.

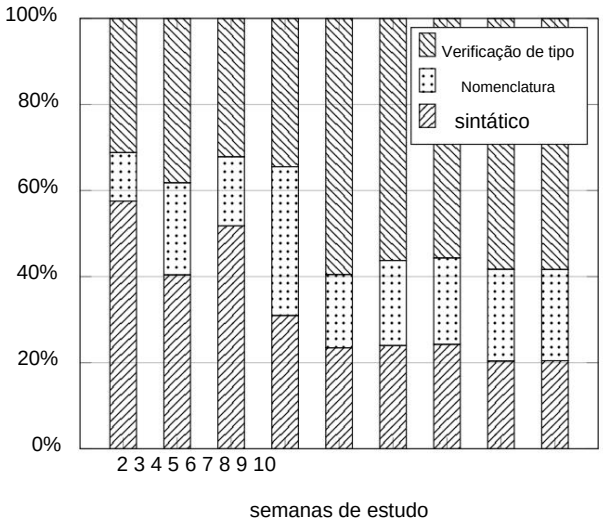
A Figura 4 mostra a frequência dos erros de compilação em diferentes estágios de compilação. Os resultados foram obtidos classificando mecanicamente a saída do compilador, mas erros semânticos falsos são reclassificados manualmente no resultado. Como pode ser visto nesta figura, erros relacionados a nomes e tipos são responsáveis por grande parte dos erros de compilação. Além disso, durante o semestre os erros relacionados ao tipo tornam-se ainda mais frequentes. Na semana 5, mais da metade das mensagens de erro são produzidas por erros de tipo.
Isto corresponde ao facto de na segunda parte do semestre serem introduzidos tipos mais complexos (funções de ordem superior, polimorfismo).

O conjunto de dados FBT forneceu resultados semelhantes aos nossos, mas sem erros sintáticos dominantes na primeira parte do curso. Assumimos que isso ocorre porque o curso foi mais curto e a sintaxe Haskell usada foi mais restrita do que nosso conjunto de dados. Curiosamente, no conjunto de dados FBT, os erros relacionados a nomes estão diminuindo e os erros de sintaxe estão aumentando. No entanto, a baixa atividade na última semana pode estar distorcendo os resultados. Resultados semelhantes são apresentados por Heeren et al. usando Hélio [4], com erros de tipo causando mais

Tabela 3: Erros dos alunos falsamente relacionados a problemas sintáticos

Semana	verificação de tipo erros	Erros de verificação de	Nomeando er erros	Erros falsos de nomeação 57
2	258	tipo falso 43	164	
3	225	19	180	40
4	202	12	135	30
5	141	10	137	4
6	240	11	111	5
7	211	9	102	1
8	266	8	167	17
9	308	2	209	
10	332	1	248	1 3

Figura 4: Categorias de erros dos alunos com base nas mensagens de erro

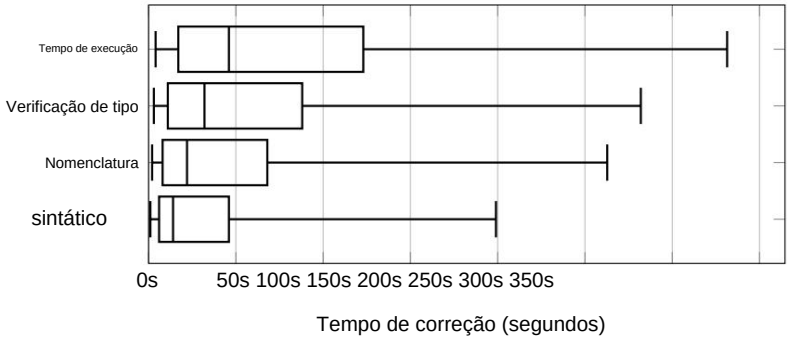


mais de 50% de todos os erros.

A Figura 5 mostra quanto tempo leva para os alunos corrigirem um tipo de erro de compilação como um gráfico de caixa. Medimos a diferença de tempo entre cada erro dado e a submissão que passa nessa fase de compilação. A medição é feita automaticamente a partir dos logs registrados e a figura mostra os resultados agregados de todo o semestre. Para erros de sintaxe, medimos o tempo até que todos os erros de sintaxe desapareçam, para erros relacionados a nomes e tipos, esperamos até o primeiro envio que não resulte em um erro de compilação. Optamos por não focar na eliminação de erros individuais, pois uma solução errada para um determinado erro pode resultar em um erro semelhante, mas um pouco diferente. No entanto, levamos em consideração todos os envios em uma sessão que resultam em um erro de compilação, não apenas o primeiro. Como comparação, incluímos também os tempos de correção de erros de tempo de execução (incluindo exceções de tempo de execução, não finalização e resultados incorretos). Somente as interações são incorporadas onde o erro foi eventualmente corrigido.

O primeiro autor, como instrutor de Haskell, percebeu que mensagens de erro complexas geralmente fazem com que os alunos parem de progredir por um período de tempo inesperado. Como os alunos estão tentando entender a mensagem de erro, eles não estão encontrando os problemas fáceis de detectar em seus envios. Geralmente esses problemas são resolvidos pelo instrutor que é questionado pelo aluno, ou percebe que o progresso do aluno

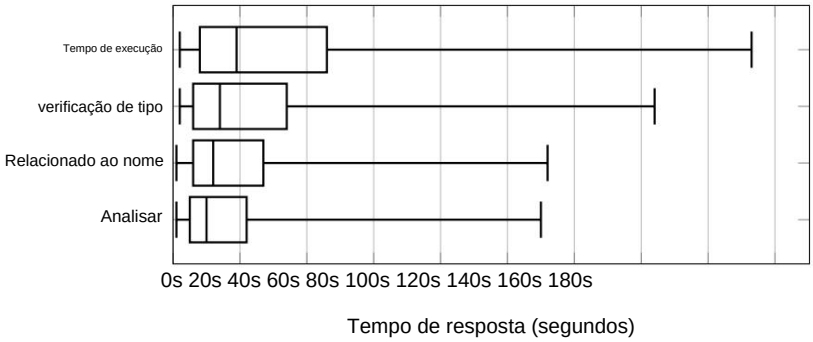
Figura 5: Tempos de correção de erros em diferentes categorias



é interrompido pelo problema.

Neste estudo, quantificamos esse fenômeno medindo quanto tempo leva para o aluno apresentar a próxima submissão após um erro. Como pode ser visto na Figura 6, o resultado é mais marcante do que no caso dos tempos de solução. O tempo que leva para os alunos encontrarem uma resposta para um erro de verificação de tipo próximo ao tempo que leva para criar uma nova solução para um problema de tempo de execução. Não é razoável dizer que entender as mensagens de erro de verificação de tipo é tão difícil para os alunos quanto depurar mentalmente o comportamento de tempo de execução de sua solução.

Figura 6: Tempos de resposta de erros em diferentes categorias



4.1 Inspeção das causas principais dos erros semânticos

Como a Figura 4 confirma que após aprender a sintaxe básica, os alunos cometem erros de compilação que são encontrados em estágios posteriores da compilação, focamos nesses erros. Vimos na Figura 3 que as mensagens de erro podem ser enganosas em alguns casos.

Para inspecionar ainda mais a natureza desses erros, a seguir, mostramos as cinco causas raiz mais frequentes para erros relacionados a nomes e verificação de tipos. Os resultados são feitos por amostragem dos erros de compilação da categoria dada. As causas principais dos erros são determinadas manualmente. Eles são relevantes com intervalo de confiança de 5% e nível de confiança de 0,95.

4.1.1 Erros relacionados ao nome

Erro global de nomes (23,8% dos erros relacionados a nomes) Como os alunos não estão familiarizados com o ambiente Haskell, os erros relacionados a nomes geralmente são causados por erros na referência a entidades da biblioteca padrão ou do ambiente de teste.

Veja a seguir um exemplo de erro ao fazer referência a uma definição importada. O aluno escreveu **Verdadeiro** e **Falso** em letras minúsculas.

```
isSingleton [ a] = verdadeiro
isSingleton _    = falso
```

Erro de nome de definição (14,2% de erros relacionados a nomes) Em outros casos, o problema não está no acesso a entidades globais, mas na grafia correta do nome da função definida. Nos exercícios, a assinatura tipográfica geralmente já é fornecida. Nesses casos, quando a assinatura de tipo e o nome da associação não correspondem, o compilador relata que não pode encontrar a associação que corresponde à assinatura de tipo presente.

Como exemplo desse erro, nesta submissão há um simples erro de digitação no nome da função definida. A assinatura de tipo gerada como parte do exercício é **isCircled :: Cell -> Bool.**

```
isCriculado x = rótulo x == Circulado
```

Um erro sintático (12,5% dos erros relacionados ao nome) Como foi demonstrado, erros anteriores de etapas posteriores da compilação resultam de erros sintáticos simples por parte dos alunos. Por exemplo, a expressão de lista **[10,9..-10]** não está correta, porque o operador .. e o sinal negativo do prefixo não estão separados, o compilador realmente procura por um operador chamado ..-, que não existe. Outras causas comuns de erros sintáticos são discutidas na Seção 5.

Erro de nome local (6,7% de erros relacionados a nomes) Vinculações de referência e variáveis definidas pelo aluno têm menos probabilidade de serem confundidas do que referências a partes do ambiente. Isso não é surpreendente, pois os alunos conhecem melhor o código-fonte que escreveram do que a biblioteca padrão.

O exemplo a seguir apresenta um erro ao referenciar um nome local definido pelo aluno. O o nome **n** está vinculado ao parâmetro da função, mas o aluno usa o nome **e** por engano.

```
elem n [] = False elem n (x:
xs ) | e == x =
  Verdadeiro | caso contrário
  = elem n xs
```

Provavelmente a maioria desses erros são apenas o resultado de esquecimento e rapidamente corrigidos.

Compreensão de lista incorreta (5,8% de erros relacionados a nomes) Os alunos parecem ter muitos problemas com a sintaxe correta de compreensão de lista. Essas estruturas sintáticas não estão presentes na maioria das linguagens de programação convencionais, mas são ensinadas no início da aula. Esta pode ser a razão pela qual os alunos estão lutando com este elemento de linguagem.

Em alguns casos fica claro que o aluno não compreende a estrutura desse elemento da linguagem do seguinte modo.

```
sumSquaresTo n = soma [ i|i < -[1..] *i ]
```

Em outros casos, como no exemplo a seguir, a causa do problema pode ter sido apenas um simples erro de digitação:

```
[n |m < -[1..] ,n < [1..m ]]
```

4.1.2 Erros relacionados ao tipo

Para erros relacionados ao tipo, a distinção entre causas raiz é mais difícil do que erros relacionados ao nome, pois os problemas são mais complexos. Em alguns casos só é possível determinar que tipo de modificações seriam necessárias para compilar a submissão do aluno.

Incompatibilidade de tipo de lista (20% de erros relacionados ao tipo) Um erro muito comum dos alunos é confundir tipos de lista e escalares. Em outros casos, listas de diferentes dimensões são a causa do problema. O erro raiz geralmente é uma aplicação de uma operação incorreta que resulta em erros de tipo difíceis de ler.

A seguir é descrito um exemplo desse tipo de problema. A tarefa era definir a função **cutRepeated** que retorna o prefixo mais longo da lista que não possui elementos idênticos adjacentes. No entanto, o aluno converteu a expressão **a:(cutRepeated_ x xs)** (do tipo **[a])** em uma lista de um elemento. A expressão resultante tem o tipo **[[a]]** que entra em conflito com o tipo da expressão **[a].**

```
cutRepeated_ a (x : xs ) | a == x
    = [a]
    | caso contrário = [ a :( cutRepeated_ x xs ) ]
```

O erro de tipo resultante é um pouco complicado e não descreve bem o problema:

```
Não é possível construir o tipo infinito: t ~ [t]. ( cutRepeated_ x xs ).
Na expressão: a :
```

Argumento de função ausente (8,6% dos erros relacionados ao tipo) O segundo erro relacionado ao tipo mais frequente dos alunos é esquecer de fornecer um dos argumentos para um aplicativo de função. Veja o exemplo do envio a seguir que está tentando implementar a operação **de contagem** usando a operação **de filtro .**

```
contagem fl = comprimento ( filtro l )
```

O aluno esqueceu de passar o argumento **f** para a função **de filtro .** A mensagem de erro do compilador informa ao usuário que o argumento da função length deve ser uma lista, mas como o aplicativo da função não está completo, é uma função. Há também uma mensagem de erro adicional porque o primeiro argumento da função está faltando e o parâmetro **l** está no lugar errado.

Incompatibilidade de tipo simples (8,6% de erros relacionados a tipo) A confusão entre tipos simples, como **Integer, Int, Double, Char, String** ou **Bool** é frequentemente encontrada entre os envios dos alunos. Isso pode estar relacionado ao fato de Haskell não possuir conversões automáticas entre esses tipos de dados, enquanto conversões implícitas estão presentes em outras linguagens de programação.

Um exemplo clássico desse tipo de erro é um problema em que o aluno tenta usar a divisão / operador em integrais, onde só é aplicável a números fracionários em Haskell.

```
x ^ n
  | n ==0 | = 1
  estranho n = x * x ^(n -1)
  | caso contrário = quadrado x ^ (n/2)
```

O aluno deve usar o operador **div** em vez disso, que realiza a divisão inteira. Os operadores de exponenciação ^, **^^** e **** também estão causando problemas frequentes, porque os alunos tendem a confundi-los entre si.

Operação errada aplicada (8,4% de erros de digitação) Em muitos casos os alunos não tinham conhecimento das funções disponíveis e não encontravam a correta para aplicar. No exemplo a seguir, o aluno usou a função **concat** (que recolhe listas de listas em listas unidimensionais), em vez do suposto operador **++** que anexa uma lista à outra.

```
pedaços de _ [] = []
chunksOf nl@ (x: xs ) = [ take nl] 'concat' chunksOf n ( drop nl)
```

A versão corrigida deste código-fonte:

```
pedaços de _ [] = []
chunksOf nl@ (x: xs ) = [ take nl] ++ chunksOf n ( drop nl)
```

Parâmetros agrupados incorretamente (6,7% de erros relacionados ao tipo) Como em Haskell a aplicação da função é escrita como a justaposição da função e dos argumentos **(fx** significa aplicar a função **f** ao argumento x), pode ser confuso para os alunos que estão acostumados para idiomas onde a aplicação da função é sempre marcada por parênteses.

Pegue a função **cutRepeated,** que retorna o prefixo mais longo da lista que não possui elementos adjuntos idênticos:

```
cutRepeated ( x:y: xs ) | x == y = [x]
                | caso contrário = x:
                cutRepeated y:xs
```

O aluno provavelmente pretendia aplicar **cutRepeated** a uma lista **y:xs** (essa explicação é suportada pela falta de espaços em **y:xs),** mas como os operadores têm menor precedência do que a aplicação da função, o GHC interpretou o lado direito como **x: (cutRepeated y) : xs** que é a aplicação de uma função de processamento de lista a um valor escalar.

A solução é simplesmente colocar a expressão **y:xs** entre parênteses.

```
cutRepeated ( x:y: xs ) | x == y = [x]
                | caso contrário = x:
                cutRepeated (y : xs )
```

5 Discussões

Com base nos resultados da investigação, podemos dizer que os erros de digitação são os erros mais frequentes após algumas semanas de estudo de Haskell. Embora os erros relacionados a nomes venham principalmente de nomes com erros ortográficos, conforme relatado por estudos anteriores [4], os erros de tipo na verdade têm muitas causas diferentes. Esta pode ser a razão

por que os erros de digitação são persistentes nos envios dos alunos ao longo do semestre. Eles também são mais difíceis de corrigir do que erros de sintaxe ou relacionados a nomes.

Também descobrimos que os erros causados por enganos na compreensão da lista são numerosos. Uma vez que a sintaxe de compreensão de lista é apenas uma alternativa ao uso de funções de processamento de lista, talvez eles possam ser reagendados para um curso de Haskell mais avançado. Outro tópico que causou muitos problemas para os alunos foi o uso de listas com vários tipos de elementos e dimensões.

Do ponto de vista do projeto do compilador, é problemático que alguns dos erros sintáticos possam causar problemas nos estágios posteriores da compilação, com mensagens não relacionadas ao problema real. Algumas das instâncias desse problema frequentemente encontradas na solução do aluno:

- Escrever o sinal negativo não separado de outros símbolos, por exemplo **[0,-1..-10].**
- Confundir o uso de nomes e operadores, tentando usar operadores na forma de prefixo sem parênteses **foldl + 0 [1..10],** ou tentando usar nomes como operadores sem crases **a mod** b.
- Esquecer de colocar o operador de multiplicação explícito nas expressões: **3x + 4y.**

Sugerimos direcionar esses problemas com casos especiais de relatórios de erros. Isso envolveria alterar o compilador para reconhecer essas circunstâncias e responder com uma mensagem de erro mais informativa.
Os resultados deste estudo sugerem que as ferramentas para ajudar os alunos a identificar os problemas reais de compilação em seu código seriam muito benéficas para fins de aprendizado.

Ameaças à validade O estudo foi feito usando os dados de um curso de Haskell de uma universidade. É certo que o currículo seguido afetou os resultados deste estudo. Os resultados do conjunto de dados FBT foram usados para generalizar os resultados. O sistema que forneceu nosso conjunto de dados era anônimo, então há alguma chance de que pessoas de fora do curso também o tenham usado e seus envios também tenham entrado no conjunto de dados.

A categorização das diferentes causas de erros de compilação é baseada na experiência do primeiro autor como programador e instrutor de Haskell. Os resultados podem ser afetados por erros categorizados incorretamente.

6. conclusões

Neste artigo, estudamos os erros de compilação cometidos por alunos ao usar um site interativo durante o curso de Haskell para iniciantes. Analisamos os dados usando várias etapas.

Primeiro, examinamos as taxas gerais de sucesso e falha e descobrimos que a proporção de erros do compilador permanece o mesmo durante o semestre.

Em seguida, dividimos os erros do compilador em três grandes categorias: erros sintáticos, relacionados a nomes e erros de verificação de tipo. Os resultados mostram que no início do curso os erros sintáticos são os mais frequentes, mas ao longo do semestre os erros de verificação de tipo tornam-se a maior fonte de erros do compilador. Para tornar os resultados mais precisos do que seria possível usando apenas as mensagens de erro, fizemos uma amostra dos envios para identificar os erros introduzidos por mensagens de erro categorizadas incorretamente.

Também reconstruímos as sessões dos alunos e usamos os dados para medir quanto tempo leva para os alunos corrigirem diferentes erros. Usamos duas métricas para isso, o “tempo para corrigir uma categoria de problema” e o “tempo para responder”. A partir dos resultados, fica claro que os erros de tipo não são apenas os mais frequentes dos diferentes erros do compilador, mas levam mais tempo para serem corrigidos.

Por fim, aplicamos uma amostragem ao conjunto de mensagens de erro para agrupá-las de acordo com a raiz comum causas. Mostramos as causas raiz mais comuns para erros relacionados a nomes e verificação de tipo.

Trabalho futuro Após esta pesquisa, nosso próximo objetivo é projetar ferramentas de ajuda para os alunos. Isso significaria usar algumas ferramentas externas para ajudar os alunos a resolver erros de compilação ou evitá-los em primeiro lugar. Também esperamos analisar erros individuais com mais detalhes.

Referências

[1] Hudak, P., Peyton Jones, S., Wadler, P., Boutel, B., Fairbairn, J., Fasel, J., ... & Kieburtz, D. (1992). Relatório sobre a linguagem de programação Haskell: uma linguagem não estrita e puramente funcional versão 1.2. Avisos ACM SigPlan, 27(5), 1-164. doi:10.1145/130697.130699

[2] Ray, B., Posnett, D., Filkov, V., Devanbu, P. (2014, novembro). Um estudo em larga escala de linguagens de programação e qualidade de código no github. Nos Anais do 22º Simpósio Internacional ACM SIGSOFT sobre Fundamentos da Engenharia de Software (pp. 155-165). ACM. doi:10.1145/2635868.2635922

[3] Singer, J., & Archibald, B. Functional Baby Talk: Analysis of Code Fragments from Novice Haskell Programmers. In Proceedings TFPIE 2017. EPTCS 270, 2018, pp. 37-51. doi:10.4204/EPTCS.270.3

[4] Heeren, B., Leijen, D., & van IJzendoorn, A. (2003, agosto). Hélio, por aprender Haskell. Em Proceedings of the 2003 ACM SIGPLAN workshop on Haskell (pp. 62-71). ACM. doi:10.1145/871895.871902

[5] Gerdes, A., Heeren, B., Jeuring, J., & van Binsbergen, LT (2017). Ask-Elle: um tutor de programação adaptável para Haskell dando feedback automatizado. International Journal of Artificial Intelligence in Education, 27(1), 65-100. doi:10.1007/s40593-015-0080-x

[6] Pettit, RS, Homer, J., & Gee, R. (2017, março). As mensagens de erro do compilador aprimorado ajudam os alunos?: Resultados inconclusivos. Nos Anais do Simpósio Técnico ACM SIGCSE 2017 sobre Educação em Ciência da Computação (pp. 465-470). ACM. doi:10.1145/3017680.3017768

[7] Barik, T. (2016, novembro). Como as ferramentas de análise estática devem explicar as anomalias aos desenvolvedores? Em Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (pp. 1118-1120). ACM. doi:10.1145/2950290.2983968

[8] Barik, T., Smith, J., Lubick, K., Holmes, E., Feng, J., Murphy-Hill, E., & Parnin, C. (2017, maio). Os desenvolvedores leem as mensagens de erro do compilador? Em Proceedings of the 39th International Conference on Software Engineering (pp. 575-585). Imprensa IEEE. doi:10.1109/ICSE.2017.59

[9] Programação Funcional: Introdução a Haskell (BSc.). E ¨otv ¨os Lor´and University. **http://lambda.inf.elte.hu/Index_en.xml#introduction-to-haskell-bsc** Página acessada: (2019, abril). Link permanente: **https://perma.cc/L423-CFZG**

[10] Jones, SP, Hall, C., Hammond, K., Partain, W., & Wadler, P. (1993, julho). O compilador Glasgow Haskell: uma visão técnica. Em Proc. Conferência Técnica da Estrutura Conjunta do Reino Unido para Tecnologia da Informação (JFIT) (Vol. 93).