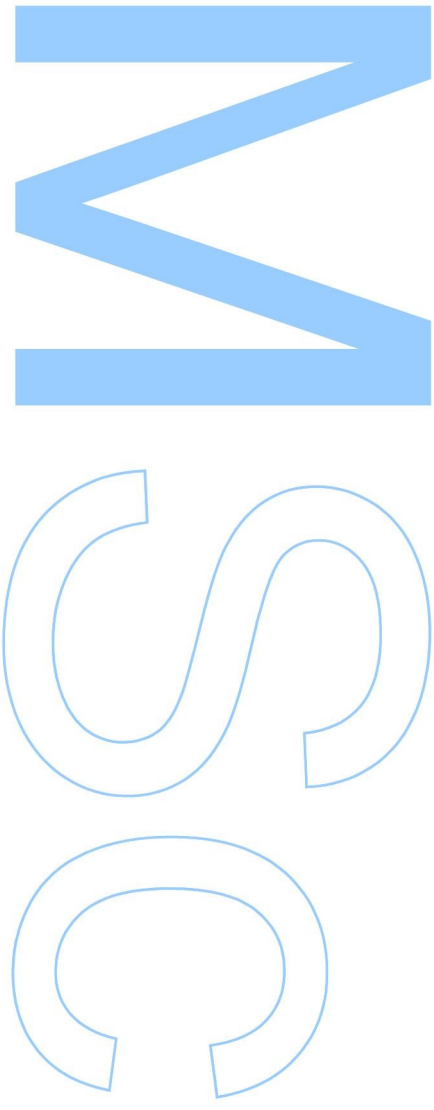
Um ambiente para programação de Haskell em Android

Daniel Josué Domingues Magalhães
Mestrado em Ciências de Computadores
Departamento de Ciências de Computadores
2016

Orientador
Prof. Doutor Pedro Vasconcelos, FCUP



Todas as correções determinadas pelo júri, e só essas, foram efetuadas.

O Presidente do Júri,

Porto, / /

Abstract

This dissertation describes the development of a native application that evaluate *Haskell* expressions in the *Android* system. To achieve this goal we have used the *Hugs* interpreter, a functional programming system based on *Haskell*, which allows evaluating expressions natively, therefore, without recurring to an online service.

In order to use the Hugs system to perform expression evaluation, it was first necessary to perform a background study of this system to determine the best approach to adapt it, by building an *interface* adapted to the mobile system for the *Hugs* interpreter. This study allowed the interconnection between the existing *Hugs* system, written in C, with the interface, developed in Java and XML.

Keywords: Android; Android SDK; Android NDK; Haskell; Hugs; Gofer; Java; XML.

Resumo

Esta dissertação descreve o desenvolvimento de uma aplicação nativa para avaliar expressões em *Haskell* no sistema *Android*. Para atingir esse objetivo, utilizámos o interpretador *Hugs*, um sistema de programação funcional baseado no *Haskell*, que permite avaliar expressões de modo nativo e, desta forma, sem necessitar de aceder a um serviço *online*.

Para utilizarmos um sistema já existente para efectuar a avaliação das expressões, foi necessário um estudo prévio a este sistema para determinar a melhor forma de adaptá-lo, construindo uma *interface* adaptada a um sistema móvel para o interpretador *Hugs*. Este estudo permitiu a interligação do sistema *Hugs*, construído em C, com a restante aplicação, implementada em Java e XML, resultando na aplicação que desenvolvemos.

Palavras-chave: Android; Android SDK; Android NDK; Haskell; Hugs; Gofer; Java; XML.

Agradecimentos

Gostaria de agradecer ao meu orientador, Pedro Baltazar Vasconcelos, por ter sugerido o tema da dissertação, por me ter guiado na elaboração da mesma, tanto no desenvolvimento da aplicação como na redação da dissertação.

Aos colegas Ricardo Cruz, Rui D'Orey, Joana Teixeira, Nuno Fernandes, Carla Lages e Tiago Moutinho, pelo auxílio efetuado na elaboração desta tese de mestrado, nomeadamente, a nível da composição da dissertação.

E por último, e de forma a não me esquecer ninguém em particular, um agradecimento para a minha família e meus amigos, pelo apoio demonstrado durante este período.

Conteúdo

Lista de Figuras

Lista de Códigos

Capítulo 1

Introdução

Neste trabalho pretendemos desenvolver uma *interface Android* para o interpretador *Hugs* que permita avaliar expressões em linguagem *Haskell*, de maneira interativa, rápida e eficiente.

Esta aplicação tem de ser nativa, isto é, não necessitar de qualquer ligação a sistemas externos para produzir resultados, funcionar em modo *read-eval-print-loop*, com possibilidades de repetir resultados anteriores e ser configurável em relação aos limites de *output* e *computação*.

Esta dissertação assume que o leitor tenha conhecimentos elementares de programação, mais especificamente em processamento de linguagens, linguagens imperativas, funcionais e declarativas.

1.1 Motivação

Nestes últimos tempos verificamos um aumento de popularidade e avanços técnicos em *smartphones* e *tablets* [1, 2]. Estes têm capacidade muito superior a PCs mais antigos, podendo-se mesmo considerar os *smartphones* “PCs de bolso”, sendo inclusivé mais potentes do que supercomputadores de ha 15 anos atrás [3].

O poder de processamento dos *smartphones* da atualidade permite-nos fazer cálculos complexos e, desse modo, executar programas mais exigentes. Em alternativa, existem aplicações móveis que são uma *interface* para servidores que executam a computação pretendida, um modelo de arquitectura que requer ligação de rede. Estas ligações padecem de disponibilidade, com possíveis custos para o utilizador, latência no serviço e tempo de espera associado à utilização do servidor.

Para um iniciado em programação, existem diversas opções para começar a experimentar vários tipos de linguagens de programação, em sistemas como o computador pessoal. Muitas linguagens de programação têm implementações nestes sistemas, desde as mais populares como Java ou C (ambas linguagens imperativas) [4, 5] até às linguagens de outros âmbitos, como as linguagens funcionais (*Haskell* ou *OCaml*) (tabela 2.2).

A nossa principal motivação nesta dissertação foi desenvolver um ambiente para experimentar programação em *Haskell* capaz de correr num *smarphone ou tablet Android*.

1.2 Objectivos de Trabalho

Esta dissertação teve como objetivo interligar um interpretador *Haskell* existente, com uma *interface* para *smartphone*, concretizado com uma aplicação nativa no sistema *Android*. Esta aplicação será desenvolvida para obter um desempenho adequado ao sistema no qual está a ser implementado, mais especificamente, um ecrã táctil com pequenas dimensões, ausência de teclado físico e rato. Funcionará interativamente, num processo chamado de “read-eval-print-loop”, que fará repetitivamente avaliações às expressões inseridas pelo utilizador, imprimindo os resultados obtidos na aplicação.

Ao desenvolvermos esta aplicação, tivemos em conta alguns pontos que gostaríamos de ver concretizados:

- Deverá ser uma aplicação intuitiva, funcional e de fácil utilização, rápida na iniciação e avaliação de expressões, sem quebras ou latência, e com resultados adequados;
- Deverá conter um editor de texto de “input”, um campo de “output” que possa funcionar como histórico e uma barra com opções de funcionamento;
- Em vez de reescrever um interpretador de *Haskell*, esta aplicação terá de fazer a ligação com o sistema *Hugs*.

Em termos práticos, o nosso trabalho residiu em três fases: primeiro, estudar o sistema *Hugs*, um interpretador de *Haskell open-source* [6], de forma a poder utilizá-lo na aplicação, adaptando-o; segundo, criar uma ligação entre o *Hugs* e uma aplicação *front-end* em Java. Para tal, iremos utilizar o *API server Hugs*, um API que permite ao *Hugs* funcionar num fluxo de pedido/resposta; por último, desenvolver uma aplicação em *Android* com os objectivos já enumerados.

1.3 Modo de Funcionamento

Nesta secção, iremos demonstrar ao leitor uma simples sessão da nossa aplicação.

Ao iniciar a aplicação, será apresentada um *toolbar*, um histórico, uma caixa de *input* e um botão de avaliação (figura 1.1).

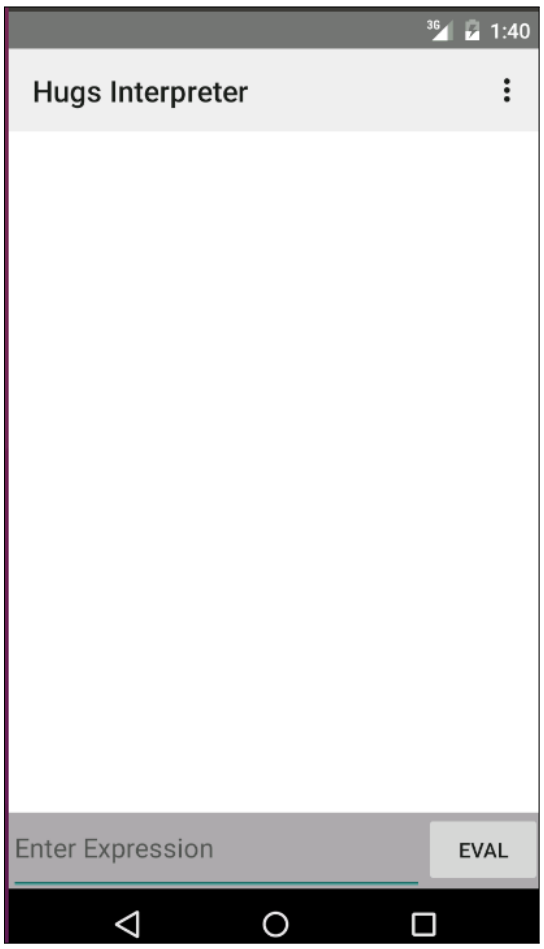


Figura 1.1: Ecrã inicial

O utilizador pode então inserir uma expressão na caixa de *input* e pedir para avaliá-la. Este pedido pode também ser realizado pelo botão *enter* do teclado (virtual ou físico). Dentro desta, pode colocar vários tipos de expressões como demonstrado nas figuras 1.2 e 1.3.

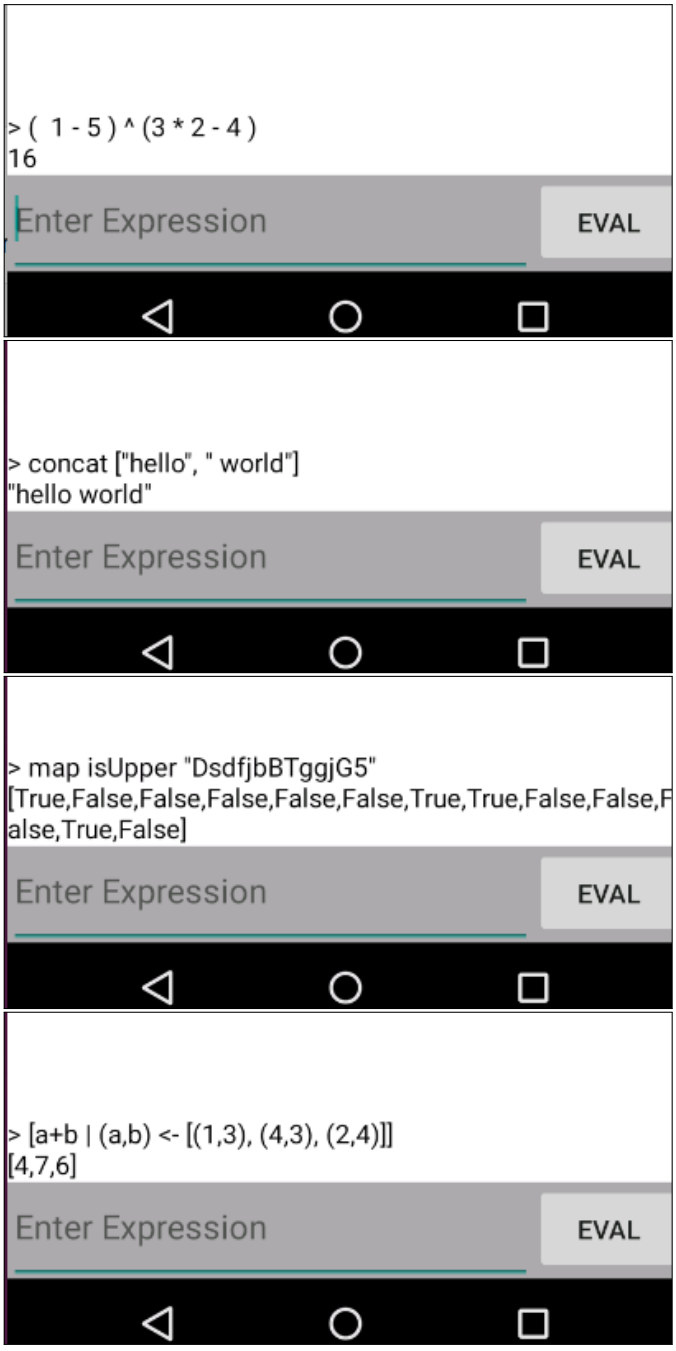


Figura 1.2: Sessão na aplicação

Figura 1.3: Sessão na aplicação (continuação)

Se pretender recuperar o resultado, pode selecioná-lo com toque, e automaticamente o resultado passa para a caixa de *input*, sendo possível editá-lo e reavaliá-lo. O utilizador pode ainda modificar alguns parâmetros do interpretador para alterar limites de computacionais (figura 1.5).

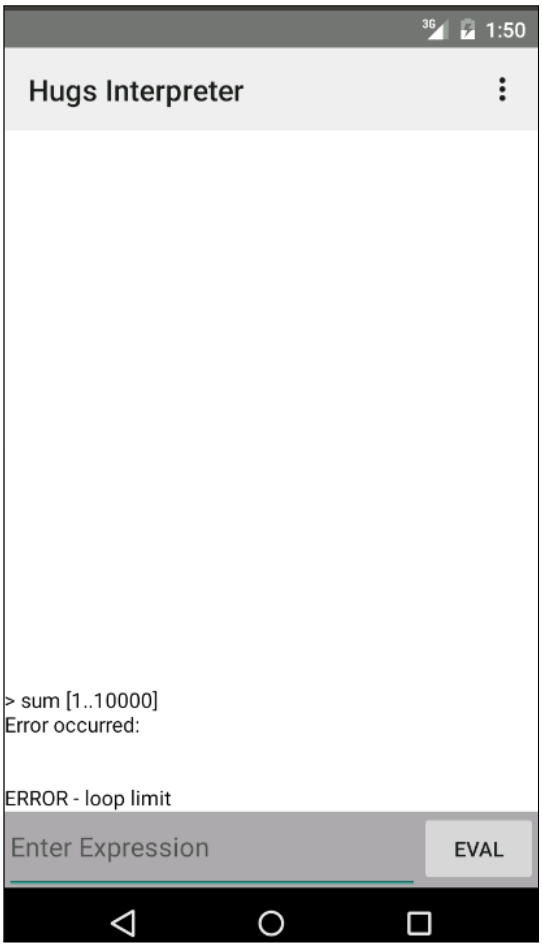


Figura 1.4: Exemplo de uma expressão não avaliada

Para evitar que a aplicação bloqueie, esta irá abortar uma computação demasiada longa, isto é, que ultrapasse o limite imposto. O limite poderá ser modificado nas opções da *toolbar*. Na figura 1.4 dá-se o exemplo de uma expressão cuja computação é demasiado longa e é interrompida com uma mensagem de erro, sendo que na próxima figura 1.5 é modificado o parâmetro de limite e a mesma expressão já é avaliada.

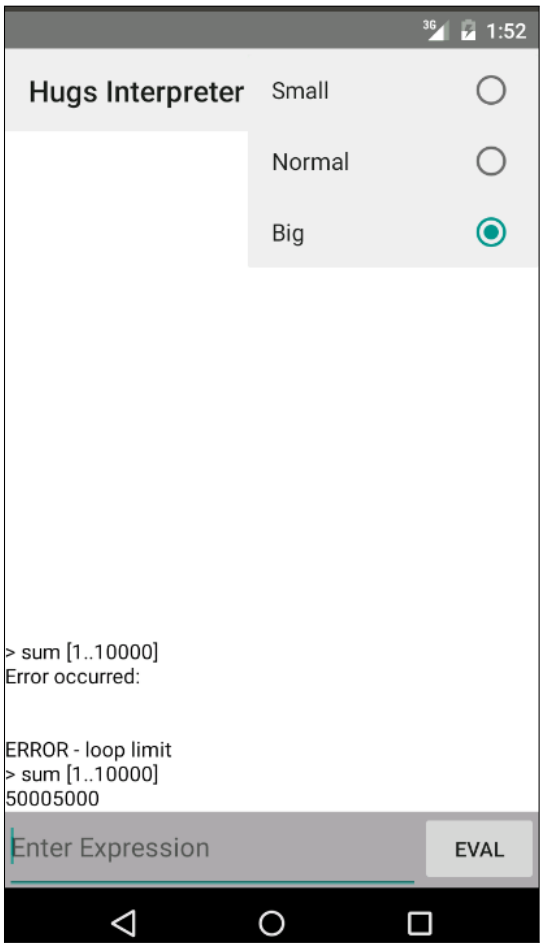


Figura 1.5: Expressão avaliada após mudança das opções

1.4 Estrutura

Esta dissertação está dividida em três capítulos:

Capítulo 2 (Estado da Arte), no qual serão revistas a linguagem *Haskell*, o interpretador *Hugs* e o sistema *Android*;

Capítulo 3 (Desenvolvimento), no qual serão documentadas as decisões relativamente à aplicação a nível de arquitectura, os desafios e o desenvolvimento da aplicação;

Capítulo 4 (Conclusão) no qual abordaremos os objectivos iniciais e os atingidos e faremos um balanço final sobre os resultados obtidos. Mencionaremos também o que poderá ser feito para melhorar a nossa aplicação em trabalhos futuros.

Capítulo 2

Estado da Arte

A nossa aplicação irá permitir programar *Haskell*, e portanto iremos debruçar-nos sobre algumas particularidades desta linguagem. Utilizámos o sistema *Hugs* como interpretador de *Haskell* e teremos neste capitulo a devida preocupação de o introduzir em termos históricos, enquadrando-o com o seu contexto, ilustrando os seus principais mecanismos. Iremos também esplanar o sistema operativo *Android*, bem como as ferramentas utilizadas para a elaboração da nossa aplicação como é o caso do SDK e NDK. Esta análise irá ser importante para melhor compreender as técnicas usadas, e irá também permitir tomar decisões durante o desenvolvimento da nossa aplicação.

2.1 Linguagem Haskell

A nossa aplicação irá permitir programar expressões em *Haskell*, sendo relevante estudar a linguagem. De seguida iremos contextualizá-la a nível histórico, demonstrando quais suas características e os ambientes onde pode ser desenvolvida.

2.1.1 História

No Outono de 1987, surgiu a ideia no meio académico, em específico por *Hudak*, *Peyton Jones* e *Wadler*, de construir uma linguagem de programação funcional “lazy” com objectivo de ser livre, de acesso académico para investigação e com descrição formal, mas principalmente, conseguir unir as outras linguagens do mesmo paradigma numa só [7].

Em 1 de Abril de 1990 foi publicado o primeiro relatório da linguagem Haskell, versão 1.1, por *Hudak* e *Wadler*, que marcou o início formal da linguagem. Desde então, a linguagem teve várias novas versões, sendo a versão atual *Haskell 2010*.

Durante a fase inicial de vida da linguagem *Haskell*, a sua elaboração ficou a cargo de um comité composto por vários académicos. O objetivo era de definir o desenho de *Haskell* através da submissão e aprovação do comité de novas ideias. Após algum tempo, foi decidido criar uma versão padronizada à qual fosse possível implementar extensões por parte de módulos particulares. A esta versão chamaram de *Haskell 98*. Isto levou a que a linguagem *Haskell* obtivesse novas funcionalidades, como por exemplo uma mais vasta e rica biblioteca e também funcionar como linguagem de investigação, o que permitiu explorar novas formas de tratamento de sistemas de tipos e metaprogramação.

2.1.2 Características

Haskell é uma linguagem funcional, desenvolvida sobre algumas características importantes que iremos identificar: a primeira característica é a *lazy evaluation*, ou *call-by-need*, que consiste na ideia de avaliar expressões só em caso de serem utilizadas [8]. Estas expressões ficam "guardadas", evitam casos de avaliações duplicadas, o que permite que estas expressões avaliadas sejam partilhadas. Esta característica já tinha sido implementada em linguagens funcionais, como é o caso das linguagens Orwell e Miranda [7], entre outras.

Outra das características que definem a linguagem *Haskell* é a "pureza", que consiste na ideia de que nenhuma função produz efeitos colaterais, noção semelhante à da função em matemática [9]. O facto de uma linguagem ser *call-by-need* torna impraticável avaliar e produzir input/output como efeito colateral, e faz com que *Haskell* seja também uma linguagem pura. Esta consequência dificultou a introdução de um mecanismo de I/O em Haskell. Isto foi resolvido posteriormente, com a utilização do I/O monádico. O mecanismo monádico foi inventado por Wadler, e foi implementado em *Haskell*. A utilização de monádicos foi posteriormente adoptado em linguagens funcionais com *call-by-value* como Ocaml, Scala e F#.

Uma característica inovadora do *Haskell* em relação às restantes linguagens funcionais é o uso de classe de tipos. Esta característica permitiu resolver o problema de sobrecarga de operadores numéricos e igualdade. Em linguagens como SML e Miranda esse problema foi resolvido de forma "ad-hoc".

A noção de tipo de classes consiste em agrupar tipos em classes que suportem determinadas

aplicações, sendo possível através de instâncias, definir implementações especifícas. As classes de tipo tornaram-se uma solução uniforme para os problemas de *overloading* e melhoraram as abordagens existentes.

Outro dos objectivos originais da linguagem *Haskell*, era que teria uma definição formal do seu sistema de tipos e semântica. A sua motivação foi o facto de permitir usar técnicas matemáticas numa linguagem de programação. No entanto, mesmo após vários relatórios que descrevem o sistema de tipos de *Haskell*, a sua semântica nunca foi completamente descrita nem definida. Isto implica que *Haskell* não tenha (ainda) uma semântica formal completa (ao contrário, por exemplo, da SML). A razão para isto suceder foi o facto de que descrevê-la nunca foi uma missão urgente, tornando-se mesmo desnecessária. Apesar de não existir uma descrição formal, a semântica utilizada por *Haskell* é fortemente inspirada em modelos formais, em particular o cálculo-*lambda*.

A linguagem que mais influenciou o *Haskell* foi a linguagem *Miranda* sendo que algumas das suas características encontrarmos em *Haskell*. *Miranda* já era uma linguagem estabelecida na altura mas o facto de ser proprietária e de não conter algumas características que foram implementadas em *Haskell*, motivou a criação de uma nova linguagem funcional.

Por último, iremos referir dois conhecidos compiladores de *Haskell*, o GHC e o NHC. O GHC (Glasgow Haskell Compiler), um projeto *open-source* começou em 1989 na Universidade de Glasgow, sendo dos primeiros compiladores de *Haskell* (estando disponibilizado desde que a linguagem teve o desenho inicial finalizado), escrito pela equipa composta por Cordelia Hall, Will Partain, and Peyton Jones.

A linguagem contém muitas das características de *Haskell*, suporta várias extensões e uma interface interativa *read/eval/print* (GHCi). O NHC, desenvolvido por Niklas Röjemo em Chalmers, em 1995, sendo que tinha como objetivo construir um compilador *Haskell* que fizesse *Bootstrap* num espaço de memória mais pequeno que GHC. Desde então já teve vários melhoramentos e optimizações, especialmente por Malcolm Wallace [7].

2.1.3 Ambientes de Programação

A nossa aplicação permitirá programar *Haskell*, mas não é a única no contexto de ambientes móveis. Numa visão abrangente dos vários dispositivos existentes, verificamos que quem quiser programar em *Haskell* num computador pessoal, tem como opções o sistema operativo Windows, Mac, Linux ou outros. Para esse efeito, existem implementações gratuitas que podem ser carregadas para Windows e Mac; além disso, são incluídas plataformas Haskell em várias distribuições de

Linux [10].

Para efeitos comparativos selecionamos algumas aplicações para programar em *smartphones* ou *tablets*. As aplicações para linguagem *Haskell* são: *Haskell Programming Language* para IOS, Try Haskell e Learn Haskell para *Android* (ambos utilizam o servidor *Try Haskell*), Ocaml Toplevel para programas em Ocaml (https://ocaml.org/) e Scheme Droid e Simple Scheme para programas em Scheme (http://www.schemers.org/). Estas aplicações têm em comum permitirem programar linguagens do paradigma funcional, daí a nossa escolha para esta comparação. A aplicação *Haskell Programming Language* é paga e não foi possível obter todos os dados requeridos para esta experiência. Foram obtidos os seguintes resultados ilustrados na tabela seguinte:

Aplicações	**SO**	**Expressões / Script**	**Offline**	**Limite Output**	**Histórico**	**Touch**	**Scroll**	**Exemplos**
Haskell Programming Language	IOS	Ambos	✘	—	✓	✘	✓	✘
Try Haskell	Android	Expressões	✘	✘	✓	✘	✓	✘
Learn Haskell	Android	Expressões	✘	✓	✓	✘	✓	✓
Ocaml Toplevel for Android	Android	Ambos	✓	✘	✓	✘	✓	✘
Scheme Droid	Android	Expressões	✓	✘	✘	✘	✓	✘
Simple Scheme	Android	Ambos	✓	✘	✘	✘	✘	✘

Tabela 2.1: Tabela de comparação de aplicações de programação

A tabela contém oito campos de comparação entre as seis aplicações escolhidas:

SO Qual o sistema operativo da aplicação.
Expressões/Script A aplicação só funciona com expressões, com scripts ou ambos.
Offline A aplicação funciona nativamente ou em modelo cliente/servidor.
Limite de Output A aplicação define o output.
Histórico A aplicação contém um histórico.
Touch A aplicação permite seleção usando toque do utilizador.
Scroll A aplicação permite movimento *Scroll*.
Exemplos A aplicação contém exemplos de ajuda para o utilizador.

Após testes realizados, salientamos os seguintes pontos: verifica-se que para a linguagem *Haskell*, não existe uma aplicação nativa para *Android*. Este tipo de aplicações que utilizam serviços externos (servidores), não permitem a elaboração de trabalhos em modo *offline* e são dependentes da latência e velocidade da ligação de rede e do funcionamento do servidor, mas têm como ponto forte o facto de ser possível realizar computações mais complexas comparativamente a uma aplicação nativa, que depende do sistema *Android* e capacidade do *Smartphone*. No entanto, o que verifica-

mos na prática é que ambas as aplicações (que utilizam o serviço `http://tryhaskell.org`) não permitem maior computação do que seria possível diretamente num *smartphone* típico (talvez para limitar requisitos de servidor)[1].

2.2 Sistema Hugs

Hugs (acrónimo para "Haskell User's *Gofer* System") é um sistema de programação funcional, baseado em *Haskell* desenvolvido por Mark P. Jones [11]. Evoluiu do sistema de programação funcional *Gofer* (acrónimo para "Good For Equational Reasoning"), mais propriamente da versão 2.3b [7].

Gofer é um interpretador, implementado em C, desenvolvido num PC de 8MHz 8086 com 640KB de memória, e desenhado para caber numa disquete de 360KB, para programar numa linguagem semelhante a *Haskell*. O seu funcionamento baseia-se num *read-eval-print-loop*, isto é, um ambiente de programação interativo na qual recebe um input, avalia-o e imprime o resultado.

Desde a sua primeira versão, assistimos a uma evolução no sistema *Gofer*, um interpretador para uma linguagem baseada em *Haskell* com o objetivo de investigação pessoal. O seu primeiro *design* foi um pequeno interpretador, que "funcionava como uma boa ferramenta para raciocínio equacional", como descreveu Mark Jones.

A nível de influências, realçamos as semelhanças entre a interface do sistema *Gofer* e a interface da linguagem Orwell, que foi uma inspiração, como o próprio autor indica. O sistema *Gofer* foi produzido para ser compilado em K&R C; apesar de já estarem disponíveis na altura compiladores ANSI C, o desenvolvimento foi em K&R C para facilitar a portabilidade em máquinas mais antigas.

2.2.1 História

O primeiro lançamento de *Gofer* foi em Setembro de 1991 sendo o seu autor Mark P. Jones. Na altura, Mark Jones era estudante de Doutoramento na Universidade de Oxford, e escreveu a sua dissertação com base no *Gofer*, apesar de inicialmente ter sido um projeto "escondido" do seu orientador. *Gofer* tornou-se uma base de teste para as suas experiências, em especial sobre classe de tipos, sendo o primeiro a implementar classe de tipos com múltiplos parâmetros, e adotou uma

[1] Experiência realizada com a avaliação da expressão "sum [1..100000]" que computa no sistema hugs e nas aplicações do "tryhaskell" e "sum [1..1000000]" que não computa em nenhuma.

variante da tradução por passagem de dicionários de Wandler e Blott [12].

Após o seu Doutoramento, Mark Jones continuou a fazer investigação para o seu projeto, no caso na Universidade de Yale em 1992. Nesta altura, adicionou suporte para construtor de classes, uma ideia que se baseava na parametrização de uma classe sobre um construtor de tipos, e produziu a primeira implementação de *do-notation* [13], uma alternativa para a sintaxe monádica, em 1994, mais tarde incluídas no *Haskell 98*. Ao modificar o *back-end* de *Gofer*, Mark Jones desenvolveu um compilador de *Gofer* para C, que foi uma base para a primeira implementação de classe de tipos [7] sem dicionários [14].

Após sair de Yale em 1994, Mark Jones decidiu modificar o *Gofer* para aceitar *Haskell*, pois devido às diferenças entre ambas, alguns programas de *Gofer* não funcionavam em *Haskell* e vice-versa. Desta forma, nascia o *Hugs*: *Haskell User's Gofer System*. O seu desenvolvimento foi maioritariamente feito quando Mark Jones começou a trabalhar na Universidade de Nottingham em Outubro de 1994.

No dia de São Valentim de 1995, *Hugs* foi lançado, suportando quase todas as características de *Haskell 1.2*, excluindo, entre outros, o sistema de módulos de *Haskell*. Por esta altura, na Universidade de Yale, Alastair Reid começou a fazer modificações ao *Hugs*, para suportar o sistema de módulos de *Haskell* que foi implementado no lançamento de *Hugs0* em Junho de 1996. Mark Jones continuou a desenvolver *Hugs* para as novas características de Haskell 1.3 como o I/O monádico.

Após estes lançamentos, Jones e Reid começaram a trabalhar em conjunto sobre a possibilidade de juntar as suas versões. O primeiro lançamento deste trabalho em conjunto foi com o *Hugs 1.4* [15], completado em Janeiro de 1998. No entanto, Mark Jones continuou a desenvolver paralelamente, e criou o *Hugs 1.3c*.

A necessidade de existir uma uniformidade sobre todas as versões diferentes existentes, tanto as de Jones, de Reid e de ambos em conjunto, fez com que fosse criado *Hugs 98* em março de 1999. O nome deriva de *Hugs* suportar *Haskell 98*, que marcou o fim da sua colaboração em conjunto pois Mark Jones deixou Nottingham e foi para OGI enquanto Alastair Reid foi para Utah. Desde então, não só Jones e Reid como também outros contribuidores ajudaram a melhorar o sistema e correção de erros. Isto decorreu até Maio de 2006, na altura em que *Hugs 98* teve a sua última versão e em Setembro de 2006 sua última atualização, encontrando-se no momento da escrita desta dissertação, sem mais desenvolvimentos.

2.2.2 Decisões de Concepção

No contexto do que estamos a desenvolver, é importante estudar a base sobre a qual a nossa aplicação irá ser construída, isto é, o sistema *Hugs*. Mais propriamente, será importante obter uma visão mais concreta sobre o seu funcionamento, requisitos e comportamentos. Para tal, focamos-nos em detalhes de construção e nos objectivos do sistema *Hugs*, utilizando para tal, tanto informação sobre o seu predecessor (*Gofer*) como também detalhes de novas versões (*Hugs 98*). Nesta secção abordamos as decisões do autor nos seguintes aspecto:.

Linguagem de Implementação: Primeiro temos de mencionar a decisão da escolha da linguagem a utilizar para o efeito de implementação. A decisão do autor recaiu como já foi dito na linguagem C, pela razão de funcionar em PCs de capacidade limitada (em especial de memória) e para permitir que o *Gofer* fosse portável e dessa forma, corresse em vários ambientes. No entanto, a utilização de uma linguagem funcional para implementar alguns componentes do *Gofer* teria sido melhor pois simplificaria a sua elaboração.

Gestão de Armazenamento: Foram implementados vários mecanismos específicos para a arquitectura do *Hugs* que iremos ilustrar.

Heap A *Heap* é utilizada por vários componentes do compilador e durante a execução do interpretador, mais concretamente na construção de árvores na análise sintática, para representar tipos que são usados na análise de tipos ou para implementar a tradução de funções para supercombinadores. Para representar dados na *Heap* é utilizada, principalmente, a estrutura de dados *Cell* (Pares, árvores, listas ou inteiros entre outros) e Texto (*Strings* ou nomes de identificadores). É possível o utilizador aumentar o tamanho da *Heap*, sendo o indicado para casos de máquinas com grande capacidade de memória, pois isso permitiria evitar problemas com a memória cheia no caso do coletor de lixo não conseguir obter “ganhos” de espaço. Mas a decisão de Mark Jones foi colocar a *heap* com tamanho fixo e desde aí não se alterou. A razão reside na capacidade de memória muito pequena das máquinas existentes na altura do desenvolvimento do sistema *Hugs*, o que obrigava que fosse definido o tamanho do espaço (pequeno mas suficiente para o máximo de valores possíveis) para a *Heap* e que se mantivesse com tal tamanho (pela simples razão de provavelmente não existir mais espaço na memória).

Tipos de Dados Cell O sistema *Hugs* tem como vantagens na gestão de armazenamento permitir a gestão automática do armazenamento e de ter implementado um coletor de lixo. O armazenamento utiliza vários tipo de dados, maioritariamente o tipo de dados *Cell*, não só para representar inteiros, como também apontadores para outros objetos contidos na *Heap*. São utilizados pelo *Hugs* em todos os componentes do seu sistema, sendo que cada um desses utiliza e gere a mesma área de memória consoante as suas necessidades. Isto torna o tipo de dados *Cell* importante, pois serve para representar vários tipos de valores tais como constantes primitivas (*Integers*, *Floating Points*, *Characters* e *String Literal*) ou pares (para estruturas como pares, árvores e listas). Nesta implementação, qualquer valor de *Cell* é codificada para um *integer* de tamanho fixo em "bits", sendo que consoante o intervalo de *integers* pode-se representar valores diferentes.

A forma como é feito este processo basea-se na utilização de blocos de inteiros de 16 bits, codificados de modo que alguns bits representam o tipo de valor e os restantes bits o valor ou apontador para a *Heap*. Exemplos disso são os valores inteiros negativos, que representam exclusivamente pares; de facto os pares são utilizados para a construção de listas (e árvores). Desta forma é evitada a representação de apontadores que usa 32 bits para representar um inteiro e um apontador para o próximo bloco da lista. Este tipo de representação é mais apropriado para computadores com pouca memória e apresenta-se como uma vantagem.

Como desvantagens, a descodificação dos tipo de dados *Cell* torna-se mais dispendiosa, pois requer operações aritméticas para realizar a descodificação dos valores inteiros utilizados como tipo de dados *Cell*. Torna-se também mais difícil redefinir a representação de valores *Cell* em tempo de execução pois é necessário o mapeamento do intervalo do inteiro.

Tabela de Strings Na gestão de armazenamento existe uma tabela de *strings*. Em tempo de compilação esta tabela é criada com espaço fixo, sendo esta decisão idealizada para o funcionamento em computadores de pequena capacidade de memória. No entanto, em computadores de maior capacidade de memória, uma tabela dinâmica que aumente consoante a quantidade de *strings* seria sempre a melhor opção de forma a evitar recompilações. De salientar também a utilização de *hash tables* para melhorar o desempenho na procura de strings na tabela, existindo mesmo várias camadas de *hash tables* o que melhora bastante a procura de *strings*.

Tipos de dados texto A tabela de *strings* contém uma *string* e um *offset* associado. Este *offset* é guardado num tipo de dado texto, e desta forma, uma *string* só é colocada na tabela uma vez.

As principais vantagens são: evitarem-se problemas como cópias da mesma *string*, utilização excessiva da *heap* e facilidade em comparações entre *strings*.

Garbage Collector O sistema *Hugs* utiliza um *mark-scan garbage collector* de forma a libertar memória na *Heap* quando necessário. Isto é feito guardando uma coleção de *roots*, ou seja, apontadores para estruturas de dados na *Heap*. O coletor de lixo é "conservador", isto é, pode marcar como estando em uso estruturas que já não são necessárias, derivado a que alguns valores na *Stack* do C poderão ser mal interpretados como sendo apontadores para a *heap*, e sendo assim, o *garbage collector* mantém os mesmo, dado que não tem forma de os confirmar. Alguns valores também estão no `yacc stack`, utilizado na análise sintática, que não é acessível pelo *garbage colector* do *Hugs*. A solução encontrada foi usar a *Stack* do *Hugs* de forma a "simular" o parser do *yacc*, ao que chamaram *Shadow Stack*. Como é óbvio, é uma duplicação de trabalho manter um segundo *stack* mas foi uma decisão de arquitectura específica para o problema em causa.

Compilação para super-combinadores: O *Hugs* utiliza uma técnica de implementação designada por *lambda-lifting*, de forma a eliminar definições de funções locais, resultando num programa que contém somente funções globais, denominadas de super-combinadores [16]. Foi implementado para transformar programas em *Haskell* para uma linguagem que possa ser compilada e executada na *G-machine*. Concretamente, o *Gofer* utiliza uma implementação do algoritmo proposto por Johnsson [17]. Este processo traduz para uma linguagem mais simples (isto é, remove listas em compreensão e equações) e mais adequada à implementação usando redução de grafos.

2.2.3 Componentes

Nesta secção, vamos descrever todos os módulos do sistema *Hugs*, fazendo uma breve descrição de cada um e discutindo as suas opções.

Componentes do Sistema Gofer: O sistema *Gofer* está dividido em componentes, cada um interligado com os restantes e todos utilizam a *storage*. De seguida daremos uma breve ilustração dos principais componentes e das suas ligações, abordando também a descrição do que mais importante existe em cada um desses componentes (ver Figura 2.1).

storage.c Onde é definida a gestão de armazenamento, ao qual todos os restantes componentes têm ligação. Nele encontramos definidas as estruturas e os mecanismos usados pelo

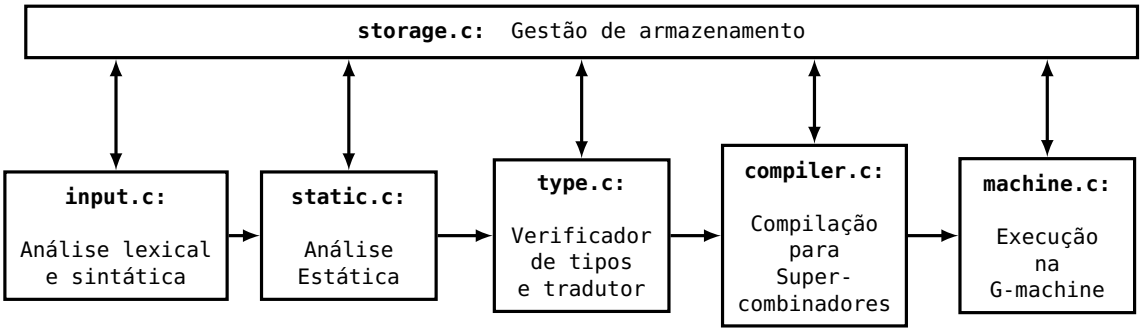


Figura 2.1: Componentes do sistema do *Gofer*

Hugs durante a compilação e execução de programas (como é o caso do Tipo de dados *Cell* e texto entre outros); o *Gofer* Stack, que é utilizado como armazenamento para várias partes do *Gofer*; um *storage allocation* automático e o coletor do lixo, que faz recuperação de memória na *heap* verificando se os apontadores para objectos nele contidos estão vivos (algoritmo *mark-scan*);

input.c Produz uma análise léxica e uma análise sintática do programa fonte, tanto a *scripts* carregados como a comandos e a expressões. Para o efeito, é utilizada uma combinação entre analisador léxical desenvolvido por Mark Jones e o gerador de analisador sintático *yacc*. O propósito da análise léxica e sintática é verificar se programas ou expressões estão escritos de uma forma correta, traduzindo-o durante este processo para uma linguagem abstrata intermédia, utilizando a gramática da linguagem. Sendo assim, o que podemos encontrar após o programa passar por ambos os componentes, é uma estrutura de dados composta por uma lista de equações com declarações de tipos;

static.c Este componente faz a análise e a verificação estática do código resultante das análises anteriores (léxical e sintática). Esta verificação reside em detectar se existem variáveis que já foram anteriormente definidas, verificar se contém repetição de variáveis, averiguar se as expressões de tipos são bem formadas, isto é, se os tipos existentes no lado direito existem no lado esquerdo da expressão e analisar os construtores de tipos, apurando se estão definidos no programa e se têm o número correto de argumentos. Como o *Gofer* aceita definições em qualquer ordem, o analisador estático só pode ser inicializado quando todo o programa passar pela análise sintática (yacc). Por último no analisador sintático é produzida análise de dependências, particularmente a inferência de kind e tipos, guardado essa informação para ser utilizada nas análises posteriores;

type.c Este componente tem de assegurar que todas as expressões e definições contêm um tipo. Caso contrário, deve tentar inferir um tipo, podendo mesmo emitir uma mensagem de erro apropriada em casos de não ter sucesso. Outra das suas funções é adicionar um parâmetro extra em funções com tipos *overloading* de forma a passar valores do dicionário na implementação de classes de tipos;

compiler.c Nesta fase, o código *Gofer* é traduzido em supercombinadores. Para tal, usa um esquema de tradução que tem os seguintes passos: primeiro o programa é passado para uma linguagem sem elementos mais complexos como listas em compreensão ou expressões de ligação (*binding*), e onde são simplificados os padrões de funções. De seguida é utilizado o *lambda-lifting* para eliminar definições locais em funções. O resultado é um programa somente com funções globais e fechadas chamadas de *supercombinadores*. Por último é processado o código resultante do *lambda-lifting* para produzir instruções para uma máquina abstrata, a *G-machine* [16];

machine.c Este componente implementa a execução de código máquina resultante da compilação dos *supercombinadores*. O avaliador implementa redução de grafos representados na *heap*, realizando assim a *lazy evaluation* do programa original.

2.2.4 Compilador do *Gofer*

O *Gofer* inclui um compilador que gera código C apropriado para as instruções da máquina abstrata originalmente desenvolvida modificando o *backend* do interpretador. Combinado um sistema run-time, uma versão simplificada da gestão de armazenamento e a implementação de funções primitivas permite geração de pequenos programas “standalone”.

O compilador de *Gofer* usa o pré-processador do C, em especial para testar a disponibilidade de blocos na *stack* e na *heap* de forma a evitar *overflow*. Testes sugerem que o compilador consegue ser, no máximo, duas vezes mais rápido que o interpretador em termos de run-time. Comparativamente com o compilador como o GHC fica bastante abaixo pois tais compiladores implementam melhores técnicas de análise e optimizações.

Outra das suas características, é o facto de que não suporta compilação separada, o que o torna insatisfatório para projectos de grandes dimensões. Em contrapartida, o compilador do *Gofer* produz ficheiros binários executáveis bastante mais pequenos que os restantes sistemas.

Outro dos aspetos positivos reside no facto de não necessitar de um *garbage collector* que seja

conservador na sua execução, mais propriamente, não necessita de analisar a *stack* do C para procurar os apontadores da *heap*, evitando muitos problemas de restrições. Ao invés, utiliza dois *garbage collector*s, um simples *mark-scan collector* e um *copy garbage collector*. Este último (*copy garbage collector*) permite a alocação de blocos por tamanho da variável na *heap* e implementa arrays como Haskell com $\mathcal{O}(1)$ de tempo de acesso. Em contraste, o mark-scan *garbage collector*, tanto do compilador como no interpretador, é menos eficiente na sua representação de arrays, pois fá-lo com estrutura de listas ligadas de *pair cells*.

O compilador não foi usado neste trabalho de dissertação.

2.2.5 Servidor do Hugs

O *Hugs* é normalmente utilizado como um programa interactivo. No entanto existem cenários nos quais podemos precisar de utilizá-lo de forma não interactiva, mais concretamente para escrever *shell* ou *cgi-scripts* em *Haskell* ou escrever *plugins* de outros programas [18].

Para este propósito, os autores de *Gofer* criaram um API para o *Hugs*, que permite aceder a algumas funcionalidades do *Gofer* de forma programática, como carregar e compilar *scripts*, compilar expressões, construir e avaliar expressões. Esta API permite-nos também a obtenção de erros e a criação de módulos dinâmicos. Foi sobre esta API que construímos a ligação do *Hugs* para a nossa aplicação em Java.

2.2.6 "Port" Hugs

Sendo o *Hugs* escrito em C portável, existe já um "Port" para *Android*, que iremos utilizar na nossa aplicação, fazendo as devidas alterações. Este "Port" está disponível livremente no github (`https://github.com/conscell/hugs-android`), na qual se encontra não só o programa completo como as instruções de instalação. Foi criado para ser integrado com a aplicação *Terminal Emulator for Android* (`https://play.google.com/store/apps/details?id=jackpal.androidterm`).

No entanto, este "Port" é limitado e não está adaptado para ser utilizado em *Smartphones*. Para o seu funcionamento, é preciso compila-lo com NDK; as bibliotecas do *Hugs* teriam de ser colocadas manualmente no SDCard; necessita do emulador do terminal, que requer as teclas utilizadas em PC para o seu funcionamento, utilizando para tal combinações de teclas no telemóvel para simular as teclas do PC, não contendo seleção por toque ou *scroll*. Todo este processo não é o mais indicado

a nível de usabilidade e requer um utilizador com alguns conhecimentos de programação.

2.3 Android

Nesta secção, iremos descrever sucitamente o sistema operativo Android, sobre o qual foi desenvolvida a nossa aplicação. Desta forma, vamos focar-nos nas suas principais características, em particular, as que usamos na implementação da nossa aplicação com o sistema Hugs.

O sistema operativo *Android* foi adquirido pela Google em 2005 de forma a que a empresa participasse no mercado de *smartphones*. O seu objetivo inicial era que funcionasse como sistema operativo para câmeras digitais, mas a empresa depressa percebeu que esse mercado não era suficientemente grande e mudou o seu rumo para um sistema operativo de *smartphones*. Nesta altura,o *Android* era um sistema fechado e secreto, só sendo revelado pela empresa que estava a desenvolver um Sistema Operativo para *smartphones* [19].

Outra das motivações que a Google teve ao adquirir o sistema *Android* enquadrava-se na necessidade de existir um sistema operativo móvel que pudesse ser flexível e atualizável por parte da Google. Mais tarde, a “Open Handset Alliance” [20], um grupo de empresas de tecnologias na qual a Google se incluía, anunciou que tinha o objectivo de construir um sistema operativo aberto para dispositivos móveis. Desde então foram comercializados *smartphones* com o sistema *Android*, obtendo várias atualizações com novas funcionalidades, correção de erros e problemas de versões anteriores.

Neste último ponto, vamos mencionar o poder computacional dos *smartphones*. Verifica-se um cada vez maior desenvolvimento a nível de *hardware*, que permite programas mais complexos, com novas funcionalidade e melhores gráficos. Desta forma, e com cada vez mais programadores a desenvolver este tipo de aplicações e mais pessoas a possuir um aparelho com este nível de poder computacional, tornou-se possível uma maior variedade de programas [21].

Hoje em dia no mercado de *tablets* e *smartphones* está predominante o sistema operativo *Android*. Continuamos a ver um crescimento deste ambiente como principal sistema operativo móvel desde 2012 (tabela 2.2) [22]. Atualmente uma grande variedade de produtos informáticos, desde voz sobre IP (VoIP), browsers, serviços de correio eletrónico, jogos, e outros tipos de aplicações, são desenvolvidos em *Android*.

2.3.1 Arquitectura Geral

Period	Android	iOS	Windows Phone	BlackBerry OS	Others
2012Q2	69.3%	16.6%	3.1%	4.9%	6.1%
2013Q2	79.8%	12.9%	3.4%	2.8%	1.2%
2014Q2	84.8%	11.6%	2.5%	0.3%	0.4%
2015Q2	82.8%	13.9%	2.6%	0.5%	0.7%

Tabela 2.2: Tabela de distribuição do OS Market Share por IDC (Agosto, 2015) [22]

Core: O Sistema *Android* é composto por várias camadas (Figura 2.2). Uma das noções mais popular, mas errada, em relação ao sistema *Android* é de que é uma versão Linux. De facto, no seu núcleo encontra-se o Kernel do Linux, mas no entanto, fica somente por este ponto a sua associação [23, 24].

Bibliotecas Android: Esta categoria ilustra bibliotecas baseadas em Java que são específicas para o desenvolvimento do *Android* [25]. Dentro destas podemos enumerar algumas das mais importantes, no caso:

Surface Manager Gera o acesso ao *display* do subsistema e compõe *layers* de gráficos 2D e 3D para várias aplicações.

Media Libraries Esta biblioteca suporta *playback* e gravação dos mais populares formatos áudio e vídeo, e também ficheiros de imagens.

SQLite Usado para aceder a dados publicados por fornecedores de conteúdos e inclui gestão de base de dados SQLite.

WebKit Um conjunto de classes com o objetivo de permitir acesso à web via *browser* para ser aplicável a programas.

OpenGL Uma interface de Java para a OpenGL ES gráficos 3D *rendering* API.

Android Runtime: Consiste no “Dalvik Virtual Machine” (DVM) e bibliotecas de Java Core. O DVM é utilizado pelas máquinas de *Android* para correr aplicações e é otimizada para ambientes com baixo poder de processamento e baixa capacidade de memória. Também permite múltiplas máquinas virtuais que fornece segurança, isolamento, e suporte na gestão de memória e “threading”. Em versões do Android mais recentes foi introduzido o Android Runtime (ART), sendo que na versão Lollipop, o DVM é completamente substituída pelo ART. As suas principais vantagens são a compilação *Ahead of Time* (AOT) [26] e melhoramentos no *garbage collector*.

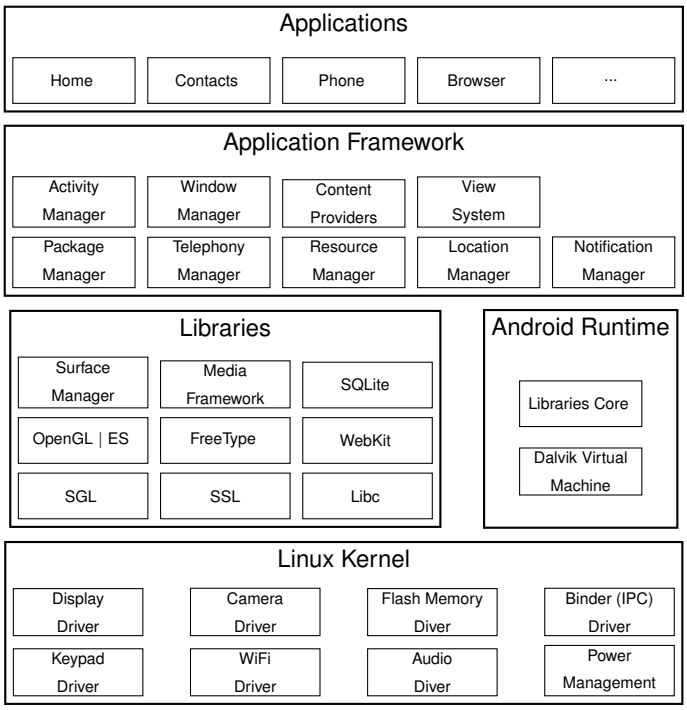


Figura 2.2: Arquitectura do Android

Framework para aplicações: São ferramentas básicas que são utilizadas pelas aplicações para utilizar funções do telemóvel. Os principais blocos da *Framework* para aplicações são:

Activity Manager Gera o ciclo de vida das aplicações e permite uma navegação comum em *backstack*.

Content Provider Permite que as aplicaçôes tenham acesso a dados de outras aplicações ou compartilhar os seus dados.

Notification Manager Permite que todas as aplicações apliquem alertas customizados na barra de estados.

Resource Manager Fornece acesso a recursos que não sejam código, como por exemplo localização de strings ou gráficos.

Aplicações: Nesta camada encontra-se não só as aplicações pré-instaladas pelo *Android* como *SMS client app* ou Web *browser* mas também aplicações desenvolvidas por terceiros.

2.3.2 Android SDK

Android SDK (Software Development Kit) permite que programadores possam criar aplicações para a plataforma *Android*. Consiste num conjunto de ferramentas de desenvolvimento que permitem projectos em Java, um emulador de dispositivos *Android*, entre outros. As aplicações correm no

DVM ou ART, enquanto o *layout* da aplicação é descrito na linguagem XML, mais concretamente declarando elementos UI ou elementos do ecrã, de forma análoga à que é criada uma página web, com elementos aninhados [27].

2.3.3 Android NDK

NDK (Native Development Kit) é uma ferramenta que permite programadores desenvolverem componentes em código nativo, em C ou C++, para dispositivos Android. Foi criado para ser integrado com o SDK (descrito como uma "companion tool" [28]), e é normalmente utilizado para desenvolver blocos de código cuja "performance" é critica ou para permitir re-utilizar código escrito em C/C++ (como o caso do sistema *Hugs*).

Capítulo 3

Desenvolvimento

O objetivo principal desta dissertação foi desenvolver uma “interface” para *Hugs* em *Android*, e para tal, foi utilizado um sistema *opensource* [29] chamado de *Hugs*, escrito em linguagem C.

Neste capítulo vamos descrever o desenvolvimento da aplicação, começando pelo desenho conceptual, no qual iremos rever as nossas ideias para a aplicação, a forma como interligamos os vários componentes, e a implementação, na qual descrevemos o caminho que nos levou à aplicação final.

Vamos ainda abordar as razões das nossas decisões que tomamos no decorrer do desenvolvimento da nossa aplicação.

3.1 Desenho Conceptual

Nesta secção iremos dar a conhecer a nossa aplicação, expondo os objetivos de funcionalidade, a interligação dos seus componentes e usabilidade. Iremos também definir os requerimentos, as ferramentas utilizadas e a arquitetura da aplicação.

3.1.1 Requisitos

A aplicação foi idealizada para que qualquer pessoa que detenha um *Smartphone* com *Android* possa utilizar a aplicação, independentemente da capacidade do smartphone. No entanto, foi decidido que a nossa aplicação iria correr a versão mais recente do *Android*, a versão 6.0 (versão do SDK 23) e como versão mínima o *Android 4.1* (SDK 16). Esta decisão foi tomada para evitar alguns problemas como é o caso do erro da função “rand” [30], que anteriormente à versão *Android 5.0*

(SDK 21) chamava-se *lrand48()*, que foi verificado durante o desenvolvimento da nossa aplicação no "port" da biblioteca *Hugs*.

Interação Utilizador/Aplicação (características de usabilidade): A aplicação terá as seguintes características:

- Uma caixa de diálogo que permita editar expressões e avaliá-las.
- Os resultados serão colocados numa caixa de texto (histórico), não editável, que permita repetir expressões anteriores.
- A limitação do resultado da avaliação, de forma a que seja possível definir o tamanho produzido pelo resultado proveniente do *Hugs*
- A limitação do tempo de avaliação, de forma a que ciclos demasiado grandes ou infinitos possam ser quebrados.
- Estas opções estão acessíveis através de uma *toolbar* [31].

3.1.2 Ferramentas

Nesta secção, iremos descrever as ferramentas utilizadas no desenvolvimento da nossa aplicação e também os passos realizados ao longo do trabalho.

Eclipse: Para a construção da aplicação utilizamos o Eclipse, um IDE de desenvolvimento Java que suporta várias linguagens via *plugins*. Esta ferramenta permite o desenvolvimento de aplicações para *Android* por intermédio do *plugin* chamado ADT (*Android Development Tools*) que possibilita acesso a vários comandos do SDK. Devido ao maior tempo de existência do *plugin* SDK para Eclipse, e desta forma, maior informação disponibilizada para resoluções de problemas, a nossa decisão recaiu sobre o Eclipse, isto apesar da Google o ter substituído pelo *Android Studio* em 2015 [32].

SDK: (*Software Development Kit*) foi utilizado no nosso projecto pelo ADT instalado no Eclipse. Este permite que seja criado um projeto *Android*, que contém não só o código que irá correr no *Android* (permitindo desenvolver código em Java e em XML), como também bibliotecas externas (possivelmente noutras linguagens).

O SDK inclui também emuladores, que simulam *smartphones* ou *tablets*, e que nos permitem testar modificações no código tanto a nível de *back-end* como de *layout*.

NDK: O NDK foi também importante no desenvolvimento, pois permite utilizar pedaços de código em linguagem nativa como C ou C++ na nossa aplicação. A base da nossa aplicação é o sistema *Hugs*, escrito na linguagem C, que foi incorporado no nosso projeto como biblioteca nativa.

Smartphones: De forma a obtermos os resultados provenientes do desenvolvimento da nossa aplicação em mais do que uma máquina, foram utilizados também *smartphones* de testes. Neste caso o Samsung Galaxy S6 (com sistema operativo Android 5.01 e 6.01 e o Vodafone Smart 4 (com sistema operativo Android 4.2.2) em complemento ao emulador do SDK.

3.1.3 Arquitetura

Nesta secção será explicada a arquitetura desenvolvida [Figura 3.1].Iremos rever a estrutura da aplicação, a descrição do trabalho realizado a nível aplicacional, em linguagem Java (motor da aplicação), as bibliotecas utilizadas e o trabalho realizado nas mesmas, no caso o sistema *Hugs*, a forma como interliga com a parte aplicacional desenvolvida em Java e a construção da interface, descrita em XML.

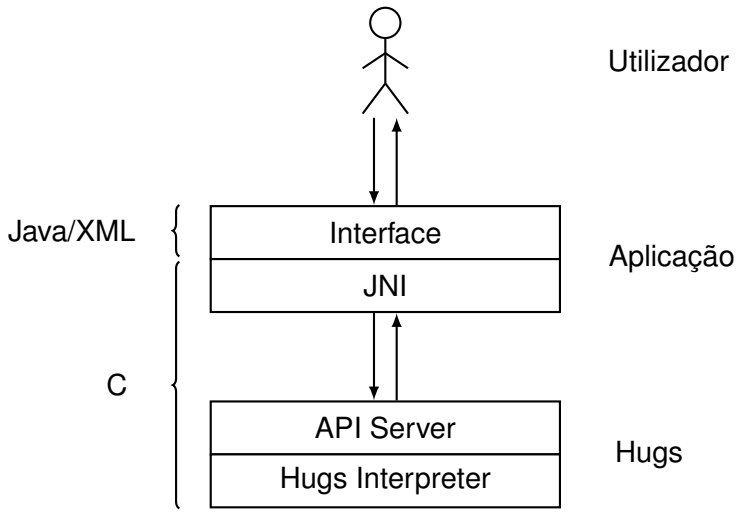


Figura 3.1: Arquitectura da aplicação

Backend: A nossa aplicação contém uma parte em Java que se interliga com o sistema *Hugs*. Esta ligação é feita chamando uma função do *API server* pelo lado do Java com argumentos: uma cadeia de texto, que é uma expressão em *Haskell* que será avaliada, o valor limite de *output* e o valor limite de avaliação.

Todo este processo é possível utilizando o JNI (Java Native Interface), do sistema *Android*, que permite à nossa aplicação fazer chamadas às bibliotecas externas produzidas em código nativo (i.e.

C/C++).

O sistema *Hugs* faz o restante trabalho, compilando a expressão em *Haskell*, e avaliando-a de seguida. Ao obter o resultado, traduz para *jstring*, enviado-a para a parte da aplicação em Java, devolvendo-a para o output (caixa de texto *textview*) após avaliação da expressão.

Funcionalidades:

- **Executar código *Haskell*** - A aplicação deve obter uma expressão proveniente do utilizador e fornecer o resultado apropriado.
- **Permitir reutilização de avaliações passadas** - Deve ser possível ao utilizador usar expressões que tenham sido anteriormente avaliadas, de forma a poderem ser editadas e reutilizadas.
- **Visualização de todo o histórico por *scroll*** - O histórico deve ter um método (*scroll*) para os casos em que o volume de pesquisas ultrapasse a dimensão do ecrã, impossibilitando, assim, a sua visualização.
- **Definir tamanho do *output*** - O tamanho do resultado da expressão inserida pelo utilizador, medido pelo número de caracteres da resposta, tem de ser ajustável.
- **Definir limite do cálculo** - Uma expressão tem de ser restrita em relação ao seu tempo de avaliação.
- **Apagar o histórico de *output*** - A aplicação dever poder limpar o histórico quando o utilizador desejar.
- **Permitir mensagens de erros** – Fornecer ao utilizador informações com detalhe sobre erros ou falhas do sistema
- **Execução do *Hugs* em *background*** – Permitir processar eventos de UI (*User Interface*).

3.2 Implementação

Após a descrição da aplicação, passamos agora para os passos realizados para a construção da nossa aplicação. De uma forma geral, iremos demonstrar e explicar o nosso procedimento, as problemáticas e as nossas decisões.

3.2.1 Interface Gráfica

O *layout* da aplicação é constituído por uma *toolbar*, duas caixas de texto (histórico e *input*) e um botão de avaliação (ver Figura 3.2)

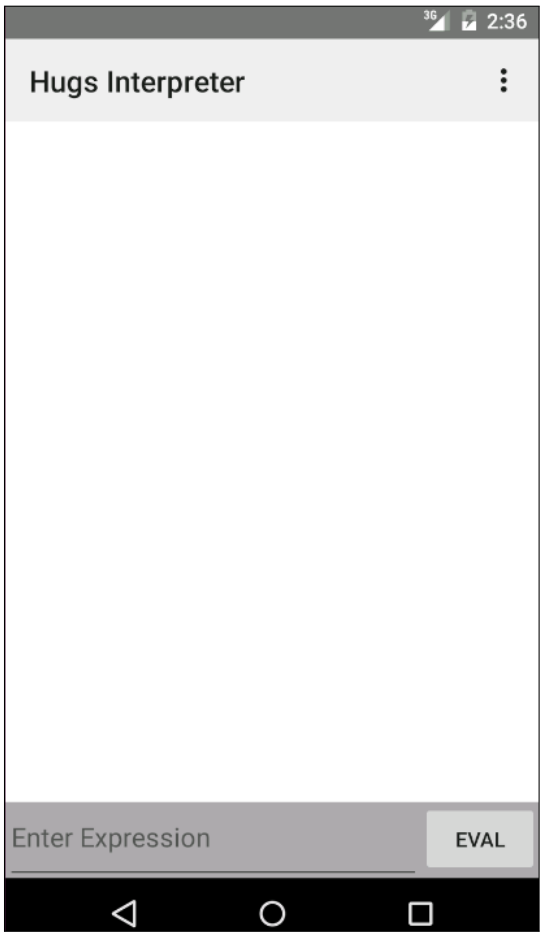


Figura 3.2: *layout* da aplicação no emulador de um Galaxy Nexus

Layout: Está dividido em duas partes: A orientação vertical da aplicação, composta pela *toolbar* e o histórico, imediatamente abaixo. Para que a caixa de texto e o botão coexistissem lado a lado, definimos a parte inferior da aplicação com orientação horizontal [33].

Como a Figura 3.2 mostra, a aplicação foi desenvolvida para ser utilizada com o *smartphone* em posição vertical (Código do *layout* disponível no anexo A).

Esta decisão foi tomada pois a utilização do *smartphone* em posição vertical facilita a utilização de toda a aplicação, em especial o histórico e a edição de avaliações. Quando em modo de escrita, o teclado surge e reduz o *layout* disponível, sendo retirada a *toolbar* das opções do ecrã. Este procedimento foi realizado da seguinte forma: define-se cada campo do *layout* como sendo um *LinearLayout* com medidas atribuídas, tanto em termos de largura, altura e peso e uma orientação própria (vertical no caso da *toolbar* e histórico e horizontal no caso do *input* e botão “eval”). Individualmente, cada caixa de texto e *toolbar* contém medidas próprias, o que faz com que, independentemente do tamanho do ecrã, a proporção da *toolbar*, histórico, *input* e botão “eval” se mantenham.

Toolbar: Colocada horizontalmente na parte superior da aplicação, a sua funcionalidade é possibilitar ao utilizador a obtenção de um menu de opções [34] da aplicação, no caso, as opções de limitar o *output*, limitar a compilação e limpeza do histórico (ver Figura 3.3).

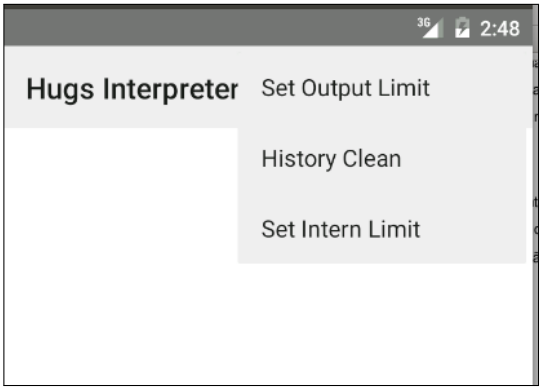


Figura 3.3: Menu das opções da aplicação

Histórico: Imediatamente a seguir à *toolbar* está o histórico da aplicação, contendo tanto o *input* como o *output* de avaliações anteriores. Trata-se de um *Textview* com a possibilidade de mostrar todas as expressões avaliadas, mesmo que este seja demasiado longo e ultrapasse as dimensões do ecrã. Para isto ser possível foi implementada no histórico uma funcionalidade de *scroll*. Também será possível reintroduzir as expressões anteriores, bastando para isso que sejam selecionada por toque, surgindo automaticamente na caixa de *input*. Os eventos de *scroll* e toque devem de ser iniciados separadamente a nível interno. A decisão que tomamos está descrita no Código 1.

```
1  public void main_function(View view) throws ExecutionException,
       TimeoutException, InterruptedException {
2  (...)
3     textview.setMovementMethod(new ScrollingMovementMethod());
4     (...)
5    }
6   private class Loading extends AsyncTask<String, Void, Bundle> {
7   (...)
8      protected void onPostExecute(Bundle w) {
9      (...)
10        textview.setOnTouchListener(new View.OnTouchListener() {
11           public boolean onTouch(View v, MotionEvent event) {
12                Layout layout = ((TextView) v).getLayout();
13                editText = (EditText) findViewById(R.id.edit_message);
```

```
14                String edittext = editText.getText().toString();
```

Código 1: Método scroll e função touch.

No entanto, a solução do Código 1 não foi perfeita, pois devido ao facto de ser possível *scroll* e de ser inserido texto de baixo para cima, não era possível perceber a que linha em específico correspondia o toque do utilizador. Para resolver esta questão, tivemos de fazer cálculos adicionais apresentados nos Códigos 2 e 3.

```
1   public void main_function(View view) throws ExecutionException,
        TimeoutException, InterruptedException {
2
3     TextView textview = (TextView) findViewById(R.id.text_id);
4     textview.setMovementMethod(new ScrollingMovementMethod());
5
6     Layout layout = textview.getLayout();
7     int height = textview.getHeight();
8     int scrollY = textview.getScrollY();
9     int firstVisibleLineNumber = layout.getLineForVertical(scrollY);
10    int lastVisibleLineNumber = layout.getLineForVertical(height+scrollY);
11    int size = lastVisibleLineNumber - firstVisibleLineNumber;
12    Log.v("Screensize", ""+size);
13
14    textview.setText(null);
15    textview.setScrollY(0);
16
17    for (int i = 0; i < size; i++){
18      textview.append("" + '\n');
19    }
```

Código 2: Inicializar linhas em branco no Textview.

Esta parte do código é utilizada para calibrar o *textview* no *layout* do telemóvel. O objectivo é que o *textview* ocupe o *layout*, e deste modo seja facilitada a identificação das coordenadas de toque.

```
1  public boolean onTouch(View v, MotionEvent event) {
2     (...)
3     int x = (int)event.getX();
4     int y = (int)event.getY();
5     if (layout != null) {
6          int line = layout.getLineForVertical(y);
```

```
7
8           int start = textview.getLayout().getLineStart(0);
9           int end = textview.getLayout().getLineEnd(textview.getLineCount() - 1);
10          String displayed = textview.getText().toString().substring(start, end);
11
12          String[] lines = displayed.split(System.getProperty("line.separator"));
13
14          int height = textview.getHeight();
15          int scrollY = textview.getScrollY();
16
17          int firstVisibleLineNumber = layout.getLineForVertical(scrollY);
18          int lastVisibleLineNumber = layout.getLineForVertical(scrollY+height);
19
20          editText.setText(lines[line+firstVisibleLineNumber].replace("> ", ""));
```

Código 3: Cálculos adicionais de deteção de toque.

No Código 2, nas linhas 3-4, obtemos os valores das coordenadas x e y do toque do utilizador, na linha 6 obtemos a linha na qual foi feito o toque, nas linhas 8-10 obtemos o texto completo que está no momento do toque no ecrã, na linha 12 obtemos o texto total e dividimos em linhas, linhas 14 e 15 obtemos o tamanho do ecrã e a quantidade de *scroll* realizado, linhas 17 e 18 obtém-se a primeira linha visível ao retirar do total de linhas a quantidade de *scroll* e a última linha somando-lhe a altura do ecrã. Por último, obtemos a linha escolhida do texto ao somarmos a primeira linha visível com a linha do toque (linha 20).

Outra das possibilidades testadas para resolver a questão do toque seria permitir ao utilizador selecionar livremente o tamanho do texto que pretendia e copiar para a caixa de *input* [35]. No entanto, preferimos o mecanismo de toque pois pareceu-nos mais intuitivo de usar (Código 4).

```
1  private class Loading extends AsyncTask<String, Void, Bundle> {
2  (...)
3     protected void onPostExecute(Bundle w) {
4     (...)
5
6        TextView tv = (TextView) findViewById(R.id.text_id);
7        tv.setTextIsSelectable(true);
8        String stringYouExtracted = tv.getText().toString();
9        int startIndex = tv.getSelectionStart();
10       int endIndex = tv.getSelectionEnd();
11       stringYouExtracted = stringYouExtracted.substring(startIndex, endIndex);
```

```
12      ClipboardManager clipboard = (ClipboardManager) getSystemService(
           CLIPBOARD_SERVICE);
13      clipboard.setText(stringYouExtracted);
```

Código 4: Método para utilização de seleção de texto.

Caixa de input: Será aqui que o utilizador terá a possibilidade de introduzir as expressões para que sejam avaliadas. Aqui aparecerá também, em caso de ter sido escolhido no histórico, o resultado de uma pesquisa anterior de forma a editá-lo ou executá-lo novamente [31].

Botão Eval: O botão *Enter* do teclado virtual do *Android* fará com que a expressão colocada pelo utilizador na caixa de texto seja avaliada (chama a função principal da aplicação, na qual define os parâmetros do layout e faz a chamada ao *Hugs*). Esta funcionalidade foi obtida modificando a instrução do botão *Enter*, que por defeito executava um parágrafo na caixa de texto, passando desta forma a chamar a função de avaliação. No entanto, verificamos alguns problemas na redefinição do mesmo, em concreto no *Android* 4.2.2, no qual o botão *Enter* do teclado virtual tem um comportamento diferente após a redefinição programada por nós. Isto acontece porque o método que usamos tem um comportamento irregular e nem sempre é acionado [36]. Por estes motivos, criamos um botão “Eval”, que quando carregado, chama a função de avaliação, tal como a tecla *Enter*.

```
1  public void turn_on_enterKey(EditText editText) {
2     editText.setOnKeyListener(new OnKeyListener() {
3          public boolean onKey(View view, int keyCode, KeyEvent keyevent)
4             if ((keyevent.getAction() == KeyEvent.ACTION_DOWN) && (keyCode ==
                 KeyEvent.KEYCODE_ENTER)) {
5                try { //utilizado na thread de inicializacao
6                  main_function(view); //avaliacao de expressao
7                } catch (ExecutionException e) {
8                      e.printStackTrace();
9                } catch (TimeoutException e) {
10                     e.printStackTrace();
11               } catch (InterruptedException e) {
12                     e.printStackTrace();
13               }
14               return true;
15            }
16            return false;
17         }
18   });
```

```
19
20    }
```

Código 5: Redefinir enter key do teclado.

No Código 5 é representada a função de *OnKey*, que em caso de um evento, verifica se o botão é acionado e se é o botão *Enter* (linha 4). Caso positivo, é executada a função principal (linha 6) [36].

Botão Cancel: O seu propósito inicial era que o utilizador pudesse cancelar uma avaliação em casos em que algum problema surgisse, em especial no caso de bloqueio por uma computação que não termine. Este botão surgiria em tempo de avaliação da expressão, no lugar do botão "Eval" e funcionaria cancelando a *thread* na qual o *Hugs* seria lançado. Iremos tratar desta possibilidade com detalhe mais à frente.

Opção Clean: Foi implementada na *toolbar* a opção "History Clean" para o efeito de limpeza do histórico, ou seja, redefine o *textview* para não conter texto.

3.2.2 Ligação ao Hugs

A aplicação que desenvolvemos contém o programa principal em Java que comunica com a descrição do *layout* (XML) como também com o sistema *Hugs* para fazer as avaliações das expressões inseridas. De seguida, iremos descrever a ligação do *Hugs* com o programa em Java (composta pela chamada do *Hugs*), a *API Server* implementada para iniciar o *Hugs Server*, a avaliação das expressões, tratamento de erros e os limites.

Chamadas ao Hugs (rotinas): Após a introdução de código *Haskell* por parte do utilizador, essa expressão representada como uma String Java, é enviada para o *Hugs* ao fazer a chamada da função "evalHugsExpr" com o argumento dessa mesma String (ver Código 6). A função encontra-se numa *thread*, mais propriamente o *AsyncTask*, sendo colocada em *background* de forma a que a aplicação seja reativa a eventos enquanto a avaliação da expressão no *Hugs* é realizada [37].

```
1 protected String doInBackground(String... param) {
2     String result = evalHugsExpr(user_input, output_limit,reduction_limit);
3     return result;
4   }
```

Código 6: Chamada ao Hugs.

Como argumentos, usamos também o limite de *output* e limite de redução de grafo. No caso do limite de *output*, podemos decidir quantos caracteres ele terá (ver 3.2.2), sendo que essa opção consta na *toolbar* e é definida no código 7. O limite de execução (código 8) reduz o número de passos de redução de grafos que representa uma expressão e desta forma, impede uma computação que não termine (ver Sub-Secção 3.2.2) (código completo do *output* e redução de grafos está disponível no anexo B e anexo D)).

```
1         case R.id.menu_one:
2              if (item.isChecked()) item.setChecked(false);
3              else item.setChecked(true);
4              OUTPUT_LIMIT = 150;
5              return true;
6
7                    (...)
```

Código 7: Output limits.

O Código 7 contém o primeiro de quatro casos para definir o *output limit*, sendo que para cada caso, se ocorrer uma modificação por parte do utilizador (linhas 2-3), o *output limit* é alterado (linha 4). O Código completo encontra-se no anexo D

```
1         case R.id.menu_small:
2           if (item.isChecked()) item.setChecked(false);
3           else item.setChecked(true);
4           REDUCTION_LIMIT = 10000;
5           return true;
```

Código 8: Limite da redução de Grafos.

O Código 8 é semelhante ao *output limit*, com as diferenças a residirem no tipo de menu e nas variáveis (linha 1) e valores que foram atribuídos em caso de escolha (linha 4).

Hugs Server API: O desenvolvimento desta aplicação envolveu alterações ao sistema *Hugs*. Antes destas alterações, tivemos de criar uma interface de ligação entre o Java e o C, construída com JNI. Esta receberá os argumentos retornados do Java como jstring e fará a inicialização do *server do Hugs* [38].

```
1 jstring
2 Java_com_happ_MainActivity_evalHugsExpr(JNIEnv* env,
3                                         jobject thiz, jstring input, jint
                                             useroutlim, jint userlimred) {
```

```
4 (...)
5   char* hugs_argv[] = {"hugs"};
6   int hugs_argc = -1;
7   HugsServerAPI* hugs = initHugsServer(hugs_argc,hugs_argv);
```

Código 9: Jstring.

No Código 9, as linhas 1-2 indicam o início de uma função do tipo *jstring* na qual recebe os argumentos da função em Java do *evalHugsExpr* (Função da *Main Activity*), nas linhas 5-7 inicia-se o servidor do *Hugs*.

Compilação: A compilação de expressões introduzidas pelo utilizador realiza-se da seguinte forma:

Primeiro é construída uma *String*, que contém a função *Haskell* “show” [39], sendo que terá como argumento a expressão inserida no *input* do programa. A razão na qual estamos a usar a função “show” é devido ao facto da *Hugs API* fazer a avaliação de, por exemplo, *strings* e *integers* com funções diferentes, *EvalString e EvalInt*. Utilizando a função “show”, todos os resultados são convertidos em *Strings*.

Por fim a *String* resultante fica colocada no topo da pilha de execução e é retirada com a chamada da função “evalString( )” e retornada após a conversão de *char* para *jstring*.

```
1  #define ANSWER(a) ((*env)->NewStringUTF(env, a))
2  (...)
3    char string[STRING_LENGTH];
4    char* string_to_compile1 = strcpy(string,"show ( ");
5    char* string_to_compile2 = strncat(string_to_compile1,user_input,
        STRING_LENGTH - 20);
6    char* final_string = strncat(string_to_compile2," ) :: String");
7    hugs->pushHVal(hugs->compileExpr("Prelude",final_string));
8
9    char* result = hugs->evalString();
10   (...)
11   return ANSWER(result);
```

Código 10: Chamada ao compilador do Hugs (JNI).

No Código 10, a linha 1 converte a string C em jstring para passar para Java, nas linhas 3-6 é construída uma string que concatenará “show” mais a string proveniente do utilizador, (exemplo *show ("1+1") :: String)*), na linha 7 esta string é compilada com o módulo *Prelude* em âmbito, a linha 9 o resultado da avaliação que será colocado no topo da stack de execução, será devolvido para a variável *result* e na linha 11 é retornado o valor à parte do Java.

Tratamento dos erros: É necessário enviar mensagens de falhas ou erros para o utilizador. Estas mensagens podem ser geradas pelo próprio *Hugs*, ou são detetadas e geradas pela *interface no Hugs* por nós construída [40]:

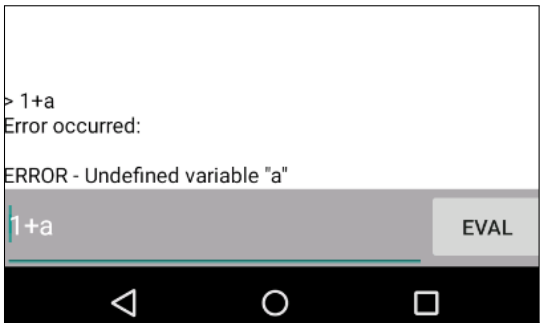


Figura 3.4: Erro e touch no histórico

```
1  HugsServerAPI* hugs =
       initHugsServer(hugs_argc,
       hugs_argv);
2  if (hugs == NULL) {
3    return ANSWER("Error
       initialization Hugs
       interpretar");
4  }
```

Código 11: Deteção de falha na iniciação do Hugs Server.

```
1  char *result = hugs->evalString();
2  err = hugs->clearError();
3  if (err) {
4    return ANSWER(err);
5  }
```

Código 12: Deteção de erro resultante da avaliação de uma expressão

No Código 11, o erro é produzido pela deteção de falha na iniciação do Hugs Server (linhas 2-3) e envia a mensagem de erro correspondente; no Código 12, o erro é interno (da análise ou avaliação) e é simplesmente devolvido para a parte de Java linhas (2-4).

Limite de output: É possível que a avaliação de uma expressão produza um *output* demasiado grande (exemplo, uma lista [1..10000] ou uma lista infinita); isto motivou um limite para o *output*. A forma como o fizemos foi definir um limite, truncando o *output* proveniente da avaliação da compilação. O corte é definido por um limite, que tem um valor de omissão, mas este pode ser alterado pelo utilizador (ver 3.2.1).

O código 13 recebe o resultado da avaliação da expressão do utilizador e faz o processamento necessário:

```
1 jstring
2 Java_com_happ_MainActivity_evalHugsExpr( JNIEnv* env,
```

```
3                                        jobject thiz, jstring input, jint
                                            useroutlim, jint userlimred)
4 {
5     (...)
6     char *result = hugs->evalString();
7     (...)
8     int output_limit = useroutlim;
9     int result_lenght = strlen(result);
10
11    char target[BIG_LENGTH];
12
13    if (result_lenght > output_limit) {
14
15            assert(BIG_LENGTH > output_limit);
16       memset(target, 0, sizeof(target));
17       strncpy(target, result, output_limit);
18       target[output_limit] = '\0';
19       char *target_char = target;
20       char *result_with_points = strcat(target_char,"...");
21       result = result_with_points;
22
23    }
```

Código 13: Limite de output.

No Código 13, a linha 8 converte `jint` para `int`, a linha 9 calcula o comprimento do resultado da compilação da expressão, caso esse seja maior que o limite definido pelo utilizador (linha 13), assegura-se que o limite de output não ultrapasse o tamanho do vector "target" (linha 15), é retirado o resultado até ao limite estipulado (linhas 16-17), feito o processamento e colocadas reticências que sinalizam que o *output* foi truncado (linhas 18-20) e retornado a *String* resultado (linhas 21).

Limite de computação: Na execução de expressões de uma linguagem de programação como *Haskell* pode ser possível que a avaliação não termine ou demore demasiado tempo. Este facto foi resolvido ao limitar a execução de forma a evitar que a aplicação bloqueie. Mais especificamente, o *Hugs* avalia as expressões ao transformá-las numa estrutura de grafos. A nossa opção foi portanto limitar diretamente a redução de grafos no interpretador da máquina abstrata do *Hugs*. Por omissão, foi por nós escolhido um valor limite, mas demos ao utilizador a possibilidade de o alterar [11].

```
1   int REDUCTION_LIMIT = 100000; //exemplo de reducao
```

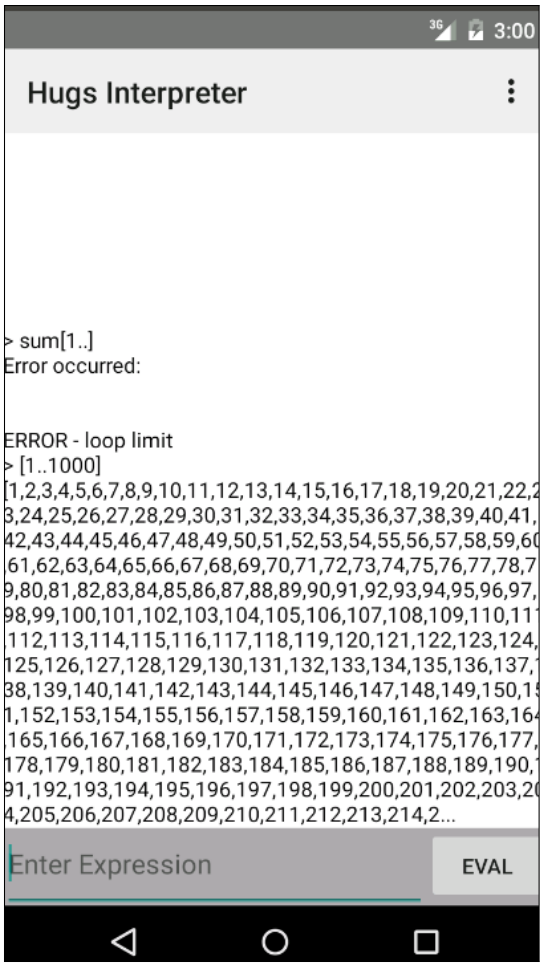


Figura 3.5: Limites

```
2
3   jstring
4   Java_com_happ_MainActivity_evalHugsExpr( JNIEnv* env,
5                                       jobject thiz, jstring input, jint
                                           useroutlim, jint userlimred) {
6
7   REDUCTION_LIMIT = userlimred;
```

Código 14: Limite de Redução na API Server.

Para fazer a ligação entre ficheiros, construímos um *header file* da nossa API e incluímos a variável *reduction_limit* como externa para desta forma, poder ser usada no ficheiro do interpretador da G-machine do *Hugs*

```
1   Void eval(n) /* Graph reduction evaluator */
2 Cell n; {
3    StackPtr base = sp;
4    Int ar;
5
6    STACK_CHECK
7    if (++evalDepth == MAX_EVAL_DEPTH)
8   hugsStackOverflow();
9 #if GIMME_STACK_DUMPS
10   evalRoots[++rootsp] = n;  /* Save pointer to root expression */
11        /* should probably test that rootsp*/
12        /* is in interval 0..NUM_STACK-1 */
13 #endif
14
15 unw:switch (whatIs(n)) { /* unwind spine of application */
16
17  if (numReductions > REDUCTION_LIMIT){
18   ERRMSG(0) "loop limit"
19   EEND;
20  }
21
22  (...)
23
24  if (name(n).primDef) /* reduce */
25   (*name(n).primDef)(root);
26  else
```

```
27        run(name(n).code,root);
28        numReductions++;
29    if (numReductions > REDUCTION_LIMIT){
30      ERRMSG(0) "loop limit"
31      EEND;
32    }
```

Código 15: Limite de Redução na machine.c.

No Código 15, é colocada uma condição para o caso de o número de reduções atuais ser maior que o máximo definido; Caso ultrapasse, termina a redução de grafos e retorna uma mensagem de erro (linhas 17-20).

Esta não foi no entanto, a nossa única possibilidade para resolver o problema da computação não terminar. A nossa abordagem inicial recaiu para tentar cancelar a *thread* de avaliação. Para tal, desenvolvemos uma segunda *thread* que funcionaria em paralelo com a *thread* principal, e a eliminaria após um *timeout*. Consideramos que se a *thread* principal não obtivesse resposta após algum tempo de execução, então estaria a executar um ciclo infinito e deveria ser cancelada. Esta solução pareceu-nos a melhor, pois dado um determinado tempo, a avaliação iria parar, independentemente da capacidade de computação do *smartphone*, ao invés da forma que implementamos, na qual definimos um limite na redução de grafos, que fará variar o tempo de execução de *smartphones* para *smartphones*. No entanto, esta hipótese denotou alguns problemas; o principal problema constatado após experiências realizadas, foi que a destruição de uma *thread* de código nativo em *Android* era pouco fiável, pois não só teríamos de destruir a *thread* como também a chamada do *Hugs* [37]. Esta possibilidade nunca se tornou solução viável e foi abandonada.

Código completo da API está disponivel no anexo D.

3.2.3 Instalação de Bibliotecas

O *Hugs* inclui um conjunto de bibliotecas padrão, como o *Prelude.hs* [41]. Estas bibliotecas têm de ser colocadas em caminhos específicos de forma a que estejam presentes quando da execução de código *Haskell*.

As instruções para utilização do “port” do *Hugs*, um terminal do interpretador *Hugs* para *Android* [6], é de que estas bibliotecas devem ser colocadas no *SD Card* e, por comandos “bash”, copiá-las para a pasta interna da aplicação.

O sistema de ficheiros do *Hugs* necessita de estar colocado num local com permissões de leitura

para a aplicação, e desta forma decidimos ir por um caminho diferente, ao realizar esta operação automaticamente e para a pasta da aplicação: Primeiro modificámos a configuração do *Hugs* nos ficheiros config.h e options.h. para o caminho da nossa aplicação.

```
1 #define HUGSDIR "/data/data/com.haskellapp/hugs/lib/hugs"
```

Código 16: Caminho da biblioteca do Hugs em option.h.

Depois colocámos as bibliotecas de *Haskell* como *assets* [42] da aplicação e copiámos estes para a pasta da aplicação durante a inicialização (anexo E).

De forma a que a cópia só acontecesse uma vez, introduzimos uma *flag* no nosso código (Código 17) de sinalização, com o propósito de identificar a primeira vez que a biblioteca é copiada. Outra possibilidade seria testar, a existência da biblioteca do *Haskell* na pasta da aplicação, o que também seria aceitável. Na prática, utilizamos uma variável que fica guardada nas definições da aplicação, e desta forma apenas na primeira inicialização ela fará a cópia da biblioteca do *Hugs* [42].

```
1    SharedPreferences sharedPreferences = getSharedPreferences("HUGS", Context.
        MODE_PRIVATE);
2    Editor editor = sharedPreferences.edit();
3    SharedPreferences prefs = getSharedPreferences("HUGS", MODE_PRIVATE);
4    int idName = prefs.getInt("First_Time", 0);
5
6    if (idName == 0) {
7      copyFileOrDir("hugs");
8      Log.v("HaskellApp", "Copied hugs Folder");
9    }
10   (...)
11   editor.putInt("First_Time", 1);
12   editor.commit();
```

Código 17: Flag da cópia das Bibliotecas de Haskell.

Capítulo 4

Conclusão

Neste capítulo iremos fazer algumas considerações sobre o trabalho efetuado e as possíveis formas de melhorar a nossa aplicação.

4.1 Objectivos

Nesta dissertação de Mestrado tínhamos como objetivo o desenvolvimento de uma aplicação para o sistema *Android*. Esta aplicação usa o sistema *Hugs* para permitir avaliação interativa de código *Haskell*.

O objetivo principal foi cumprido e conseguimos um ambiente para programar *Haskell* no sistema operativo móvel mais usado atualmente (*Android*). Resultou também na construção desta aplicação o conhecimento do funcionamento do sistema *Hugs*, do sistema *Android* e desenvolvimento aplicacional, tanto a nível de desenvolvimento de uma interface como desenvolvimentos de software, em particular JVM e JNI.

A aplicação que desenvolvemos também tem o intuito de divulgar este trabalho a nível educacional. Desta forma, disponibilizá-la no *Google Play* (`https://play.google.com/store/apps/details?id=com.haskellapp`) e futuramente num sistema aberto como o *GitHub* de forma a que outros programadores possam melhorar a versão que desenvolvemos.

Podemos dizer que atingimos as nossas principais metas. A interface da aplicação afigura-se bastante intuitiva, baseada no modelo *read-eval-print-loop*. É possível efetuar cálculos simples ou construir funções e produzir o resultado correspondente. No entanto, os resultados não são perfeitos: optámos por utilizar a função “show” para mostrar resultados, isto porque a função “show” converte

qualquer valor apropriado (por exemplo, valores aritméticos, listas, pares, etc...) para o tipo *String*. Mas a utilização do "show" só nos permite fazer computação pura e "demonstrar" expressões que obtenham um resultado, e sendo assim não permite, por exemplo, executar ações I/O.

Foi necessário proceder a alterações à execução do *hugs*. Em especial, tivemos de evitar ciclos infinitos e *output* demasiado longo. Consideramos isto importante e que fosse definido pelo utilizador. Atingimos estes objetivos; no entanto, gostaríamos de ter colocado mais opções: o programa poder ser cancelado devido à ultrapassagem de um *timeout* em vez de definirmos um limite de reduções, iria permitir uma maior uniformidade em relação a interrupção da aplicação (em termos de capacidade dos *smarphones*); a existência de um mecanismo de cancelar a avaliação da expressão introduzida de forma a que o utilizador tivesse mais controlo sobre a aplicação. Isto não foi possível devido ao facto de que o sistema *Hugs* correr como código nativo e não é controlado pela máquina abstrata do sistema *Android*, tornando-se inviável o seu cancelamento; existe a possibilidade de ao invés de utilizar *thread* para fazer a chamada ao sistema *hugs* em *background*, ser criado um novo processo (mais apropriado para cancelar, pois poderia ser feito através de sinais mas mais complicado para troca de informações entre processos) ou criado uma *thread* do Java [43]. A nossa decisão recaiu no *AsyncTask* porque achamos ser a mais apropriada para os nossos objetivos [44]. A introdução de scrips na aplicação e a aplicação poder carregar e guardar scripts em servidores como cloud ou web, que não foi investigada e introduzida por limitação temporal desta dissertação.

Quanto ao desempenho da aplicação, deparamo-nos com alguns problemas que tentámos resolver de forma eficiente, que foi o caso da instalação das bibliotecas do *Hugs*. Fizemos algumas alterações para que, ao iniciar, a aplicação obtenha as bibliotecas do *Hugs* provenientes do *apk* e as coloque na pasta da aplicação no *Android*. Esta operação demora 10-15 segundos em *Smartphones* com pouca capacidade, e isso fez com que tivéssemos de encontrar uma solução: copiar as bibliotecas uma só vez (a primeira vez). Isto permite que somente quando é iniciada pela primeira vez as bibliotecas sejam carregadas.

4.2 Trabalho futuro

Após a elaboração da aplicação, enumeramos algumas pontos adicionais que poderiam ser implementados para melhorar a nossa aplicação:

Scripts A possibilidade de editar e carregar *scripts* facilitaria a introdução de programas mais complexos. Esta funcionalidade é permitida pelo *Hugs* e seria um melhoramento bastante produtivo para a aplicação. Na prática, seria colocar código *Haskell* em ficheiros externos e importá-lo

para a aplicação de forma a que ele possa ser editado e compilado, apresentando o resultado no *layout* da aplicação.

Layout Como mencionado, a possibilidade de importar ficheiros em *Haskell* viabilizaria também a edição do ficheiro. Para tal, uma modificação do *layout* da aplicação faria todo o sentido, ao criar-se um mini-editor de texto para editar, executar, carregar ou guardar o programa. Possivelmente poderia conter exemplos, para melhorar a experiência de um novo utilizador na aplicação, de forma semelhante ao que o *Try Haskell* permite.

Ligação com outras aplicações A possibilidade da aplicação aceitar *scripts* permitiria a obtenção de ficheiros com programas mais complexos. Estes programas poderiam, por exemplo, ser provenientes da rede ou de uma página *Web*. Outra das possibilidades seria definir no próprio sistema *Android*, a nossa aplicação como omissão para abrir ficheiros *Haskell* (com extensão .hs). Esta possibilidade teria de ser aliada a definições de segurança, como a extensão *Safe Haskell* [45], que permite entre outras coisas, correr código não-fidedigno. Esta extensão esta implementada no GHC mas não no *Hugs* [46, 47], o que implica que o seu suporte pode não ser trivial.

Bibliografia

[1] Smartphone Demography, acedido em junho de 2016. URL http://www.pewinternet.org/2015/10/29/the-demographics-of-device-ownership/.

[2] Aplication Growth, acedido em junho de 2016. URL http://blog.appfigures.com/app-stores-growth-accelerates-in-2014/.

[3] SuperPC vs Smartphones, acedido em julho de 2016. URL http://www.phonearena.com/news/A-modern-smartphone-or-a-vintage-supercomputer-which{_}id57149.

[4] CppDroid, acedido em novembro de 2016. URL https://play.google.com/store/apps/details?id=name.antonsmirnov.android.cppdroid{&}hl=pt{_}PT.

[5] AIDE, acedido em novembro de 2016. URL https://play.google.com/store/apps/details?id=com.aide.ui.

[6] Hugs Github, acedido em junho de 2016, . URL https://github.com/conscell/hugs-android.

[7] Paul Hudak, John Hughes, Simon Peyton Jones, and Philip Wadler. A history of Haskell: being lazy with class. In *Proceedings of the third ACM SIGPLAN conference on History of programming languages - HOPL III*, volume 12, pages 12–1–12–55, New York, New York, USA, 2007. ACM Press. ISBN 978159593766X. doi: $10.1145/1238844.1238856$. URL http://dl.acm.org/citation.cfm?id=1238856http://portal.acm.org/citation.cfm?doid=1238844.1238856.

[8] Paul Hudak. Conception, evolution, and application of functional programming languages. *ACM Computing Surveys*, 21(3):359–411, sep 1989. ISSN 03600300. doi: $10.1145/72551.72554$. URL http://portal.acm.org/citation.cfm?doid=72551.72554.

[9] Amr Sabry. What is a purely functional language? *Journal of Functional Programming*, 8:1–22, 1998.

[10] Haskell Plataforms, acedido em junho de 2016. URL https://www.haskell.org/platform/.

[11] Mark P Jones. The implementation of the Gofer functional programming system. Technical report, Yale University, New Haven, 1994.

[12] Philip Wadler and Stephen Blott. How to make ad-hoc polymorphism less ad hoc. In *Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages - POPL '89*, New York, New York, USA, 1989. ACM Press. ISBN 0897912942. doi: 10.1145/75277.75283. URL http://portal.acm.org/citation.cfm?doid=75277.75283.

[13] Do-Notation, acedido em junho de 2016. URL https://en.wikibooks.org/wiki/Haskell/do{_}notation.

[14] Mark P. Jones. Dictionary-free overloading by partial evaluation. *Lisp and Symbolic Computation*, 8(3):229–248, sep 1995. ISSN 0892-4635. doi: 10.1007/BF01019005. URL http://link.springer.com/10.1007/BF01019005.

[15] Hugs 1.4 Manual, acedido em junho de 2016, . URL https://www.cs.auckland.ac.nz/references/haskell/hugs{_}manual/.

[16] Simon Peyton-Jones. The Implementation of Functional Programming Languages. Prentice Hall. ISBN 013453333X.

[17] Thomas Johnsson. *Lambda Lifting: Transforming Programs to Recursive Equations*. Springer-Verlag, 1985. ISBN 3-387-15975-4.

[18] Chris Angus. Trends in Functional Programming (volumes 1 2) by Greg Michaelson, Phil Trinder and Hans-Wolfgang Loidl (editors volume 1), and Stephen Gilmore (editor volume 2). Intellect Books, Bristol, 2001, 2002. *Journal of functional Programming*, 13(4):S0956796803214878, jul 2003. ISSN 09567968. doi: 10.1017/S0956796803214878. URL http://www.journals.cambridge.org/abstract{_}S0956796803214878.

[19] David R. Tobergte and Shirley Curtis. Android Market, acedido em junho de 2016, 2013. ISSN 1098-6596. URL http://www.nytimes.com/2015/05/28/technology/personaltech/a-murky-road-ahead-for-android-despite-market-dominance.html?{_}r=0.

[20] Open Handset Alliance, acedido em junho de 2016, 2007. URL http://www.openhandsetalliance.com/press{_}110507.html.

[21] Smartphone improvement, acedido em junho de 2016, 2015. URL http://www.androidauthority.com/smartphone-performance-improvements-{_}-626109/.

[22] Kantar Worldpanel. Smartphone Os Market Share. España, acedido em junho 2016, 2015. URL http://www.kantarworldpanel.com/global/smartphone-os-market-share/.

[23] Shiju P John. Android Architecture, acedido em junho de 2016, 2015. URL http://www.eazytutz.com/android/android-architecture/.

[24] J. Melorose, R. Perroy, and S. Careas. Linux vs Android, acedido em junho de 2016, 2015. ISSN 1098-6596. URL https://www.yac.mx/pt/mobile-security/android/android-is-based-on-linux-but-what-does-that-mean.html.

[25] What is Android? URL https://silextech.wordpress.com/tag/surface-manager/.

[26] Ahead of Time compilator, acedido em junho de 2016. URL https://blog.mozilla.org/luke/2014/01/14/asm-js-aot-compilation-and-startup-performance/.

[27] Android SDK, acedido em junho de 2016. URL http://www.webopedia.com/TERM/A/Android{_}SDK.html.

[28] Android NDK website, acedido em junho de 2016. URL http://www.android.pk/android-ndk.phphttps://developer.android.com/ndk/index.htmlhttp://www3.ntu.edu.sg/home/ehchua/programming/android/android{_}ndk.html.

[29] Hugs98 website, acedido em junho de 2016, . URL https://www.haskell.org/hugs/.

[30] Android Rand Error, acedido em junho de 2016. URL http://stackoverflow.com/questions/28010753/android-ndk-returns-an-error-undefined-reference-to-randhttp://stackoverflow.com/questions/31194260/android-ndk-and-newer-api-support.

[31] Android View, acedido em junho de 2016. URL https://developer.android.com/reference/android/view/View.html.

[32] Google acaba com ADT, acedido em junho de 2016. URL http://pplware.sapo.pt/smartphones-tablets/android/google-vai-acabar-com-o-desenvolvimento-android-no-eclipse/.

[33] Android. Android UI, acedido em junho de 2016. URL https://developer.android.com/guide/topics/ui/index.html?hl=enhttp://www.eazytutz.com/android/android-linearlayout/.

[34] Android Menu, acedido em junho de 2016. URL https://developer.android.com/guide/topics/resources/menu-resource.html?hl=zh-tw.

[35] Android Copy Paste, acedido em junho de 2016. URL https://developer.android.com/guide/topics/text/copy-paste.html.

[36] Android Key Problem, acedido em junho de 2016. URL https://developer.android.com/reference/android/view/KeyEvent.html.

[37] Asynctask, acedido em 2016. URL https://developer.android.com/reference/android/os/AsyncTask.html?hl=zh-tw.

[38] API Server, acedido em junho de 2016. URL http://edi.fmph.uniba.sk/{~}winczer/fp/hugs1.4/server.ps.

[39] Function Show, acedido em junho de 2016. URL http://zvon.org/other/haskell/Outputprelude/show{_}f.html.

[40] Hugs errors, acedido em junho de 2016, . URL https://mail.haskell.org/pipermail/hugs-bugs/2008-October/001864.htmlhttps://www.cs.kent.ac.uk/people/staff/sjt/craft2e/errors/allErrors.htmlhttps://mail.haskell.org/pipermail/hugs-bugs/2007-May/001806.html.

[41] Prelude, acedido em junho de 2016. URL https://hackage.haskell.org/package/base-4.8.2.0/docs/Prelude.html.

[42] Android Storage, acedido em junho de 2016. URL https://developer.android.com/guide/topics/data/data-storage.html?hl=zh-twhttps://developer.xamarin.com/guides/android/application{_}fundamentals/resources{_}in{_}android/part{_}6{_}-{_}using{_}android{_}assets/.

[43] Java threads vs. Android AsyncTask: Which to use?, acedido em Agosto de 2016. URL http://www.vogella.com/tutorials/AndroidBackgroundProcessing/article.html.

[44] Android background processing with Handlers, AsyncTask and Loaders - Tutorial, acedido em Agosto de 2016. URL http://www.vogella.com/tutorials/AndroidBackgroundProcessing/article.html.

[45] Safe Haskell, acedido em Agosto de 2016. URL https://downloads.haskell.org/{~}ghc/7.8.4/docs/html/users{_}guide/safe-haskell.html.

[46] The Hugs-GHC Extension Libraries, acedido em Agosto de 2016. URL https://downloads.haskell.org/{~}ghc/3.02/docs/users{_}guide/libs.html.

[47] Language extensions supported by Hugs and GHC, acedido em Agosto de 2016. URL https://www.haskell.org/hugs/pages/users{_}guide/hugs-ghc.html.

Apêndice A

Código “Layout”

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/action_settings"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <android.support.v7.widget.Toolbar
       xmlns:app="http://schemas.android.com/apk/res-auto"
      android:id="@+id/my_toolbar"
      android:layout_width="match_parent"
      android:layout_height="?attr/actionBarSize"
      android:background="?attr/colorPrimary"
      android:elevation="4dp"
      android:theme="@style/ThemeOverlay.AppCompat.ActionBar"
      app:popupTheme="@style/ThemeOverlay.AppCompat.Light"/>


    <ImageView
       android:contentDescription="@string/desc"
    android:id="@+id/initial_image1"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:layout_weight="1"
      android:visibility="gone"
```

```
27        android:background="@color/black"
28        android:adjustViewBounds="true"
29        android:src="@drawable/hugs" />
30
31
32
33
34
35      <LinearLayout
36        android:layout_width="match_parent"
37        android:layout_height="0.0dp"
38        android:layout_weight="0.9"
39        android:orientation="vertical">
40
41        <TextView
42          android:id="@+id/text_id"
43            android:layout_width="fill_parent"
44            android:layout_height= "match_parent"
45            android:gravity="bottom|left"
46            android:background="@color/black"
47            android:textColor="@android:color/holo_blue_dark"
48            android:textColorHighlight="@android:color/primary_text_dark"/>
49
50      </LinearLayout>
51
52        <!--
53
54          android:textSize="32sp"
55
56          android:textIsSelectable="true"
57
58            android:onClick="onClick"
59        android:clickable="true"
60        android:focusable="true"
61           android:focusableInTouchMode="true"
62           android:clickable="true"
63        -->
64
65    <LinearLayout
66       android:layout_width="match_parent"
```

```
67        android:layout_height="0.0dp"
68        android:layout_weight="0.1"
69        android:background="@android:color/darker_gray"
70        android:orientation="horizontal">
71
72
73
74         <EditText android:id="@+id/edit_message"
75           android:layout_width = "0.0dp"
76           android:layout_height= "50.0dp"
77           android:layout_weight="0.40"
78           android:gravity="start|center"
79           android:hint="@string/edit_message"
80           android:textColor="@android:color/white"
81           android:textColorHighlight="@android:color/primary_text_dark"/>
82
83       <ProgressBar
84           android:id="@+id/loadingPanel"
85           android:layout_weight="0.2"
86            android:layout_width="wrap_content"
87            android:layout_height="wrap_content"
88            android:indeterminate="true" />
89
90      <Button
91           android:id="@+id/button_send"
92            android:layout_width="10.0dp"
93            android:layout_height="50.0dp"
94        android:layout_weight="0.1"
95           android:text="@string/button_send"
96           android:onClick="main_function" />
97
98
99        <Button android:id="@+id/button_cancel"
100           android:gravity="center"
101           android:layout_weight="0.2"
102         android:layout_width="20.0dp"
103         android:layout_height="50.0dp"
104         android:text="@string/button_clear"
105         android:onClick="cancel_event"/>
106
```

```
107
108     </LinearLayout>
109
110
111   </LinearLayout>
```

Apêndice B

Código Opções Redução Grafos

```
1  public boolean onOptionsItemSelected(MenuItem item) {
2      switch (item.getItemId()) {
3
4          case R.id.menu_small:
5            if (item.isChecked()) item.setChecked(false);
6            else item.setChecked(true);
7            REDUCTION_LIMIT = 10000;
8              Log.v("HaskellApp",String.valueOf(REDUCTION_LIMIT));
9            return true;
10
11         case R.id.menu_normal:
12           if (item.isChecked()) item.setChecked(false);
13           else item.setChecked(true);
14           REDUCTION_LIMIT = 100000;
15             Log.v("HaskellApp",String.valueOf(REDUCTION_LIMIT));
16           return true;
17
18         case R.id.menu_big:
19           if (item.isChecked()) item.setChecked(false);
20           else item.setChecked(true);
21           REDUCTION_LIMIT = 1000000;
22             Log.v("HaskellApp",String.valueOf(REDUCTION_LIMIT));
23           return true;
24
25 }
```

Apêndice C

Código Opções Limites de “output”

```
case R.id.menu_one:
            if (item.isChecked()) item.setChecked(false);
            else item.setChecked(true);
            OUTPUT_LIMIT = 150;
              Log.v("HaskellApp",String.valueOf(OUTPUT_LIMIT));
            return true;

      case R.id.menu_two:
            if (item.isChecked()) item.setChecked(false);
            else item.setChecked(true);
            OUTPUT_LIMIT = 400;
              Log.v("HaskellApp",String.valueOf(OUTPUT_LIMIT));
            return true;

      case R.id.menu_three:
            if (item.isChecked()) item.setChecked(false);
            else item.setChecked(true);
            OUTPUT_LIMIT = 750;
              Log.v("HaskellApp",String.valueOf(OUTPUT_LIMIT));
            return true;

      case R.id.menu_four:
            if (item.isChecked()) item.setChecked(false);
            else item.setChecked(true);
            OUTPUT_LIMIT = 999;
              //1000 gives error?
```

```
              Log.v("HaskellApp",String.valueOf(OUTPUT_LIMIT));
            return true;
```

Apêndice D

Código API

```
#include "runhugsApp.h"
#include <stdio.h>
#include <jni.h>
#include <stdlib.h>
#define HUGS_SERVER 1
#include "server.h"
//#include "server2.h"
#include <string.h>


#define ANSWER(a) ((*env)->NewStringUTF(env, a))

//inicializados com valores por defeito
int REDUCTION_LIMIT = 100000;
int STRING_LENGTH = 100;
int BIG_LENGTH = 10000;


jstring
Java_com_happ_MainActivity_evalHugsExpr( JNIEnv* env,
                                        jobject thiz, jstring input, jint
                                           useroutlim, jint userlimred)
{

  REDUCTION_LIMIT = userlimred;
```

```
26  //int SIZE = 100000;
27  //return ANSWER(REDUCTION_LIMIT);
28  const char *user_input = (*env)->GetStringUTFChars(env, input, 0);
29
30  char* hugs_argv[] = {"hugs"};
31  int hugs_argc = -1;
32
33  //return ANSWER(initHugsServer (hugs_argc,hugs_argv));
34
35
36  HugsServerAPI* hugs = initHugsServer (hugs_argc,hugs_argv);
37
38  if (hugs == NULL){
39    return ANSWER("Error initialization Hugs interpretar");
40  }
41
42  hugs->setOutputEnable(0);
43
44
45  char* err = hugs -> clearError();
46  if (err) {
47    return ANSWER(err);
48    /*
49    char string[SIZE];
50    freopen("/dev/null", "a", stdout);
51    setbuf(stdout, string);
52    char *error3 = strcat(err,"-");
53    char *error2 = strcat(error3,string);
54    return ANSWER(error2);
55    */
56  }
57
58    char string[STRING_LENGTH];
59    char *string_to_compile1 = strcpy(string,"show ( ");
60    char *string_to_compile2 = strcat(string_to_compile1,user_input);
61    char *final_string = strcat(string_to_compile2," ) :: String");
62    hugs->pushHVal(hugs->compileExpr("Prelude",final_string));
63
64    char *result = hugs->evalString();
65
```

```
66   err = hugs -> clearError();
67
68   if (err) {
69     return ANSWER(err);
70   }
71
72
73     int output_limit = useroutlim;
74     int result_lenght = strlen(result);
75
76     if (result == ""){
77       result_lenght = 0;
78     }
79
80     char target[BIG_LENGTH];
81     char snum[10];
82
83   if (result_lenght > output_limit){
84
85     assert(BIG_LENGTH > output_limit);
86     memset(target, 0, sizeof(target));
87     strncpy(target, result, output_limit);
88     target[output_limit] = '\0';
89     char *target_char = target;
90     char *result_with_points = strcat(target_char,"...");
91     result = result_with_points;
92
93   }
94
95   //shutdownHugsServer(hugs);
96     return ANSWER(result);
97
98
99
100    exit(0);
101
102 }
```

Apêndice E

Código para instalação da biblioteca do Hugs

```
1  private void copyFileOrDir(String path) {
2     AssetManager assetManager = this.getAssets();
3     String assets[] = null;
4     try {
5        assets = assetManager.list(path);
6        if (assets.length == 0) {
7           copyFile(path);
8        } else {
9           String fullPath = "/data/data/" + this.getPackageName() + "/" + path;
10          File dir = new File(fullPath);
11          if (!dir.exists())
12             dir.mkdir();
13          for (int i = 0; i < assets.length; ++i) {
14             copyFileOrDir(path + "/" + assets[i]);
15          }
16       }
17    } catch (IOException ex) {
18       Log.e("tag", "I/O Exception", ex);
19    }
20 }
21
22 private void copyFile(String filename) {
23    AssetManager assetManager = this.getAssets();
```

```
24
25      InputStream in = null;
26      OutputStream out = null;
27      try {
28         in = assetManager.open(filename);
29         String newFileName = "/data/data/" + this.getPackageName() + "/" +
               filename;
30         out = new FileOutputStream(newFileName);
31
32         byte[] buffer = new byte[1024];
33         int read;
34         while ((read = in.read(buffer)) != -1) {
35            out.write(buffer, 0, read);
36         }
37         in.close();
38         in = null;
39         out.flush();
40         out.close();
41         out = null;
42      } catch (Exception e) {
43         Log.e("tag", e.getMessage());
44      }
45
46   }
```