FFI

Antti Marttila
antmatma@jyu.fi

8.4.2008

Tiivistelmä

Foreign Function Interface (FFI) on Haskellin lisäominaisuus, jonka avulla voidaan käyttää muilla kielillä toteutettuja rutiineja ja aliohjelmakutsuja. Käytettäessä pelkästään funkiokutsuja ja triviaaleja tietorakenteita, päästään kielirajojen ylityksestä helpolla.

1 Johdanto

Tässä raportissa esitellään yleisesti mitä Foreign Function Interface (FFI) on ja mitä ongelmia sen toteutuksessa on. Näytetään myös yksinkertaisimmat tavat toteuttaa FFI Haskellista käsin C-kieleen. Sekä esitellään pikaisesti välineitä FFI:n helpompaan käsittelyyn. Lopussa myös pohditaan miksi FFI on tarpeen ohjelmoinnissa.

2 Foreign Funktion Interface

Foreign Funktion Interface on nimensä mukaan rajapinta, jolla voidaan halutusta kielestä (esim. Haskell) kutsua toisella kielellä (esim. C) toteutettu rutiinia tai palvelua, eli aliohjelmaa. Huomion arvoista on, ettei FFI rajoitu vain funktiokutsuihin, vaan se voi myös tehdä metodikutsuja olioille ja toteuttaa ei-triviaalien tietotyyppien ja olioiden siirtoa kielestä toiseen yli kieli rajojen. Epätriviaalit toteutukset ovat kuitemkin suhteettoman hankalia toteuttaa.

FFI on Haskellin termi ja Common lispin vastaava on CFFI. Muut kielet kutsuvat tätä ominaisuutta omilla tavoillaan, esimerkiksi Javalla se on Java Native Interface.

FFI Haskellin tapauksessa tarkoittaa sitä, että kirjaston ja kääntäjän avulla on luotu ominaisuuksia joiden avulla Haskell-koodista voidaan kutsua mitä tahansa C-kirjastoa, funktiota tai tietorakennetta. Ei siis tarkoita sitä, että voidaan kirjoittaa sekä Haskellia että C:tä keskenään samaan tiedostoon ja että siitä saataisiin toimiva ohjelma.

Useissa tapauksissa FFI on määritelty korkeamman tason kielellä, joten se voi pyytää palveluja matalamman tason kieliltä kuten C:ltä ja C++:lta. Tämä tehdää usein siksi, että päästäisiin käyttöjärjestelmän API:n käsiksi tai ihan vain tehokkuuden parantamisen vuoksi.

Useat FFI:t sallivat myös isäntäkielen palvelujen kutsumisen palvelun tarjonneen kielen puolelta. Esimerkiksi C voi kutsua jotain Haskellin toteutusta. FFI ei siis ole vain yksisuuntaista. Tällöin voidaan esimerkiksi tehdä joku kaunis toteutus Haskellilla, jonka toteutus C:llä veisi tapeettomasti ohjelmointiaikaa ja olisi samalla tarpeettoman virhealtista. Voimme sitten kutsua tätä Haskellin toteutusta C-koodista. Tämä kuullostaa tarpeelliselta, mutta käytännössä sille ei olle tarvetta [3, luku 1.1].

Haskellin ja C:n tapauksessa funktiokutsuista tekee mielenkiintoisen se, että Haskell käsittelee periaatteessa yksiparametrisia funktioita jonossa ja C taasen iskee kaikki argumentit yhdellä kertaa kutsuttavalle aliohjelmalle [2, luku 6.1].

Eri tietotyyppien käsittelyssä voi olla eri kielillä eroja, jolloin FFI:n täytyy huolehtia siitä, että molemmat puhuvat samasta asiasta, kun käsitellään eri tietotyyppejä. Esimerkiksi Int C:ssä ja Haskellissa ei välttämättä ole sama asia, koska C:n int voi olla 16, 32 tai 64 bittinen esitys kun taas Haskellin int on lukualue $[-2^{29}, 2^{29} - 1]$ [1, *luku* 1.2].

Haskellin FFI:n käytössä tulee ottaa huomioon C:tä käytettäessä se, ettei toisessa ole ollenkaan roskienkeruuta ja vaikeiden tietorakenteiden siirto kielirajan yli ei ole helppoa.

3 Yksinkertaisia toteutuksia FFI:llä

Tässä kappaleessa esitellään yksinkertaiset toteutukset siitä, kuinka Haskellista päästään C:n valmiisiin kirjastoihin käsiksi lähes yhtä helposti kuin itse C:stä ja kuinka Haskellista päästään käsiksi itse tehtyyn C-ohjelmaan. Tässä kappaleessa soisi esiteltävän vielä osoittimien sekä ei triviaalien tietorakenteiden käyttöä, mutta niiden esimerkkikoodit jäivät toteuttamatta niiden vaativuuden vuoksi.

3.1 Haskellista C:n kirjastoon

Seuraavassa esimerkkikoodissa päästään käsiksi C:n kirjastoon math, johon päästään tavallisessa C-koodissa käsiksi komennolla

```
#include <math.h>
```

joka ei hirveästi eroa seuraavasta esimerkistä.

```
0 module Main where
1
2 import Foreign.C.Types(CDouble)
3
4 foreign import ccall "math.h sin" sin_double :: CDouble -> CDouble
5
6 main = do
7   putStrLn $ unlines [ show (sin x, sin_double x) | x <- [0.0, 0.1 .. 1.0] ]
```

Rivillä 2 oleva Haskellin kirjastokutsu ottaa käyttöön C-yhteensopivan double-tietotyypin. Yleensä atomiset tietotyypit, joita on esimerkiksi int, double, boolean, ovat helppoja toteuttaa FFI:n avulla kielirajojen yli.

Rivillä 4 kutsutaan varsinaista C-kirjastoa [2, luku 6] [4, luku 2]. Foreign kertoo Haskellille, että kyseessä on FFI:n komento.

Import kertoo että ollaan hakemassa jostani kielestä jotain toteutusta. Importin vaihtoehtona on export, joka kertoo että olemme tarjoamassa vieraalle kielelle jotain aliohjelmaa.

Ccall kertoo, että olemme toteuttamassa standardin C-kääntäjän kanssa yhteistyötä. Ccallin vaihtoehtona voi olla cplusplus kommunikointiin C++-kielen kanssa, jvm Javan virtuaali koneen kanssa ja stdcall Win32 API:n kanssa. Tämä kutsumis konventio kertoo siis minkä kielen kanssa ollaan tekemisissä.

Sitten annetaan, kun kutsumis konventiona on ccall, stringi jonka alussa on kirjasto eli C:n otsikkotiedosto jota halutaan käyttää ja lopussa on funktion nimi, jota kirjastosta halutaan käyttää.

Viimeisenä on C-kielen funktion synonyymi Haskelista käytettynä. Eli loppuosa on tavanomainen funktion esittely Haskellissa. Funktion esittelyn täytyy olla sellainen, että se sopii kutsuttavaan C-aliohjelmaan.

Rivillä 7 käytetään Rivin 4 määrittelemää funktiota, kuin se olisi Haskellin oma funktio.

Tämä on vasta puolet siitä että päästään varsinaisesti ajamaan kahden kielen yhteistyössä toteuttamaa ohjelmaa. Käännös vaiheessa täytyy kertoa

GHC-kääntäjälle että ollaan käyttämässä FFI:tä. Seuraavassa on käännösohje GHC-kääntäjälle. Oletetaan että edellisen koodiesimerkin koodi on tiedostossa Main.hs.

```
$ ghc -o foo -ffi Main.hs
$ ./foo
```

Käännöskäskyn lisäparametrina oleva -ffi, joka voi olla myös -fffi, kertoo kääntäjälle, että ollaan käyttämässä FFI:tä ohjelmakoodissa.

3.2 Haskellista itsetehtyyn C-ohjelmaan

Seuraavassa esimerkissä käyetään omaa C-ohjelmaa Haskellista käsin.

Otsaketiedosto C-ohjelmalle.

```
0 //foo.h
1 #ifndef FOO_H
2 #define FOO_H
3
4   int a = 10;
5   int foo(int i);
6 #endif /* FOO_H */
```

Varsinainen C-ohjelma.

```
0 //foo.c
1 #include "foo.h"
2
3 int foo(int i)
4 {
5   return i * a;
6 }
```

Varsinainen Haskell ohjelma.

```
0 --Foo.hs
1 module Main where
2
3 import Foreign.C.Types
4
5 foreign import ccall safe "foo.h foo" foo :: CInt -> IO CInt
6
7 main = do
8     r <- foo 2
9     putStrLn $ show r
```

Rivillä 3 on otettu käyttöön kaikki C-yhteensopivat tyypit vaikka ohjelmasssa käytetään vain CInt-tyyppiä.

Rivillä 5 kutsutaan itsetehtyä ohjelmaa foo.h tiedoston avulla, josta otetaan käyttöön aliohjelma foo. Käskyn neljäntenä sanana oleva safe kertoo käskyn suojauksen tason. Safe antaa kutsutun C-ohjelman kutsua Haskelia ja tekee kielien välisestä kommunikaatoista raskaampaa. Toinen vaihtoehti on unsafe, joka keventää kommunikaatiota, mutta ei mahdollista C:n takaisinkutsuja. Jos kumpaakaan näistä vaihtoehdoista ei aseta päälle, niin oletuksena on arvo safe [1, luku 1.2]. Aliohjelmakutsi palauttaa tyypin IO CInt jossa IO takoittaa, että aliohjelma voi aiheuttaa sivuvaikutuksia.

Käännökset toteutetaan seuraavilla komennoilla.

```
$ gcc -c foo.c
$ ghc -c -fffi Foo.hs
$ ghc -o foo foo.o Foo.o
$ ./foo
```

Kahdessa ensimmäisessä käännöskomennossa ei linkitetä vielä mitään -c liitteen avulla. Vasta kolmannessa komennossa linkitetään ohjelman vaatimat osat toisiinsa.

4 Työkaluja FFI:n käyttöön

Green Card on Haskellin esikääntäjä. Se muuttaa kaikki % alkuiset rivit FFI:yhteensopiviksi [3]. Käytettäessä Green Card koodia Haskellissa, voidaan saada aikaiseksi seuraavanlaista koodia.

```
0 module M where
1
2 %fun sin :: Float -> Float
3
4 sin2 :: Float -> Float
5 sin2 x = sin (sin x)
```

Kaikki muu on standardia Haskellia, paitsi rivi 2, jonka se muuttaa seuraavanlaiseksi C-rajapintaa kutsuvaksi koodiksi.

```
0 module M where
1
2 sin :: Float -> Float
3 sin f = unsafePerformPrimIO (
```

```
4           case f of { F# f# ->
5           _casm_ "%r = sin(%0)" f#  `thenPrimIO` \ r# ->
6           returnPrimIO (F# r#)})
7
8 sin2 :: Float -> Float
9 sin2 x = sin (sin x)
```

Rivit 2-6 on Green Cardin generoimaa koodia.

Muita FFI:n käyttöä helpottavia valmiita toteutuksia ovat C2Haskell ja H/Direct joita ei tässä käydä sen tarkemmin.

5 Pohdintaa

C on kielenä hyvin lähellä assembleria, joka taas on lähes konekieltä. Tämä aiheuttaa sen, että C:llä kirjoitettu koodi on lähellä rautaa ja sen kirjastototeutukset ovat yleensä hyvin nopeita. Eivät varmaan koskaan hitaampia kuin muilla "korkeamman tason" kielillä toteutetut kirjastot. Toinen C:n vahvuus on se, että jos on ylipäätään tehty kirjasto, niin se on lähes poikkeuksetta toteutettu myös C:llä.

Edellisen perusteella voisi väittää, että kaikki pitäisi tehdä C:llä. Ei. C on taasen hyvin virhealtista ja hankalaa koodattavaa muihin "korkeamman tason" kieliin verrattuna. Täten on parempi toteuttaa ohjelmansa esimerkiksi Haskellilla, joka vähentää ohjelmoijan virheitä ja helpottaa ohjelman suunnittelua tekemisen aikana, sekä on huomattavasti lyhyempää koodia kirjoittaa kuin C.

Nyt edellisen perusteella voisi väittää, että kaikki pitäisi tehdä Haskellilla. Varovainen kyllä. Mutta parhaaseen lopputulokseen päästään silloin, kun kuitataan Haskellin hitaat pullonkaulat ohjelmasta C-kirjastoilla FFI:n avulla. Näin saadaan Haskellin näköistä koodia, joka ei haittane "epäpyhää" Haskellistia, mutta saadaan aikaiseksi ohjelma jota immperatiivisen kannattajat eivät pääse haukkumaan tarpeettoman hitaaksi. Ongelmaksi tulee tietysti se, ettei ohjelma ole toteutettu täysin funktionaalisesti. Toisaalta rikkooko jo pelkkä pinon käyttö Haskellissa sivuvaikuttomattomuuden.

6 Yhteenveto

Tästä raportista ei välttämättä tullut lukijalle sellaista kuvaa, että FFI olisi kovin vaikeaa tai ongelmallista, mutta sitä se on. Joka tapauksessa oli se vaikeaa tai ei, täytyy olla mahdollisuus käyttää muilla kielillä tehtyjä to-

teutuksia. Kunhan ei lähdetä leikkimäät epätriviaaleilla tietorakenteilla, ja pidättytään C:n aliohjelmien käytössä, päästään suhteellisen helpolla.

Lähteet

[1] Manuel Chakravarty [editor], University of New South Wales, *The Haskell 98 Foreign Function Interface 1.0 An Addendum to the Haskell 98 Report*, 2003.

[2] Simon PEYTON JONES, Microsoft Research, Cambridge, *Tackling the Awkward Squad: monadic input/output, concurrency, exceptions, and foreign-language calls in Haskell* , February 5, 2008.

[3] Thomas Nordin and Simon Peyton Jones, *Green card: a foreign-language interface for Haskell*, saatavilla WWW-muodossa <URL: `http://www.dcs.gla.ac.uk/fp/software/ghc/ghc-doc/green-card.html`>, 21.4.1997.

[4] Sigbjorn Finne, Sven Panne, Manuel Chakravarty, Malcolm Wallace, and The GHC Team, *A Haskell foreign function interface*, saatavilla WWW-muodossa <URL: `http://haskell.org/definition/ffi/ffi-article.html`>, 2000.