Sprachinteroperabilität

Heute hier:

- Fremde Sprachen aus Haskell aufrufen (call-out)
- Haskell aus fremden Sprachen ausrufen (call-in)
- Datenübertragung zwischen den Grenzen: marshalling
- Das Haskell Foreign Function Interface (FFI)

Grundlagen

- Manuel 'Chilli' Chakravarty (ed):
  The Haskell98 Foreign Function Interface 1.0
- GHC User's Guide, Kapitel 8.
- Simon Peyton-Jones: Tackling the awkward squad: monadic input/output, concurrency, exceptions, and foreign-language calls in Haskell.

Probleme bei der Sprachinteroperbilität

- Unterschiedliche Datentypen
- Unterschiedliche Auswertungsparadigmen
- Unterschiedliche Parameterübergabedisziplin
  - Call by reference, call by value, im Register, auf dem Stack, Reihenfolge. . .
- Meist sehr ad-hoc gelöst

Haskell und das Standard-FFI

- Spracherweiterung zur Deklaration von
  - importierten Fremdfunktionen
  - exportierten Haskellfunktionen
- Entwurfsphilosophie:
  - Sehr einfach, keine Betonung auf Komfortabilität;
  - Ziel: FFI als standardisiertes back-end für andere Werkzeuge
- Module zum einfachen Marshalling
  - Marshalling: Konversion/Aufbereitung der Daten zwischen den Sprachen
  - Betonung liegt auf Haskell-seitigem Marshalling
- Unterstützt C, C++, JVM, Win32, .NET

Deklaration importierter Fremdfunktionen

- Eine einfache C-Funktion mit

  `foreign import ccall toLower :: Char -> Char`

- C-Funktion mit Seiteneffekten und anderem Namen

  `foreign import "PutChar" putChar :: Char-> IO ()`

- Sicherheit: `safe` vs. `unsafe`
  - `safe` (Default): Funktion kann wieder Haskell aufrufen.
  - `unsafe`: Funktion ruft nicht wieder Haskell auf.
  - Beispiel:

    ```
    foreign import "PutChar" unsafe putChar :: Char-> IO ()
    ```

- Achtung: Der Typ wird nicht überprüft!

Deklaration exportierter Haskellfunktionen

- Exportierte Funktion:

```
foreign export ccall "Foo" foo :: Int-> Int
```

- Export von Instanzen:

```
double :: Num a=> a-> a
foreign export ccall "doubleI"
                     double :: Int-> Int
foreign export ccall "doubleF"
                     double :: Float-> Float
```

Fremdtypen (foreign types)

- Externe Entitäten auf Fremdtypen beschränkt:
- Basisfremdtypen sind:
  - `Char`, `Int`, `Double`, `Float`, `Bool`;
  - `Int8`, . . . , `Word64`, `Ptr a`, `StablePrt a`, aus Modul `Foreign`
- Fremdtypen sind:

$$a_1 \texttt{ -> } a_2 \texttt{ -> } \ldots \texttt{ -> } r$$

  - a_i sind Basisfremdtypen, oder Typsynonyme, oder `newtype`'s davon;
  - r dito, oder `()`, oder $IO\ a$ (mit a dito)

Marshalling

- `Int` und `Word`
- Module `Foreign` und `CForeign`
- Abbildung Typen (Haskell) $\longleftrightarrow$ Typen (C)
  - Siehe FFI-Spezifikation, S. 22 (Tabelle 2)

Speicherverwaltung I: `Ptr` und `Storable`

- Typ `Ptr a` abstrakte Adresse auf Objekt vom Typ a
  - Instanz von `Eq`, `Ord`, `Show`
- Ausgezeichneter Zeiger ins 'Nichts'

```
nullPtr :: Ptr a
```

- Pointerarithmetik, Alignment, Casting

```
plusPtr  :: Ptr a -> Int-> Ptr a
minusPtr :: Ptr a -> Int-> Ptr a
alignPtr :: Ptr a -> Int-> Ptr a
castPtr  :: Ptr a -> Ptr b
```

Die Klasse `Storable`

- Klasse `Storable` erlaubt Zugriff auf Inhalt der `Ptr`
  - Instanzen: Basisdatentypen, `Int` und `Word`, Zeiger
- Größe und Alignment:

```
sizeOf    :: Storable a=> a -> Int
alignment :: Storable a=> a-> Int
```

- Peek und Poke:

```
peek :: Storable a=> Ptr a-> IO a
poke :: Storable a=> Ptr a-> a-> IO ()
```

  - Alignment beachten!
  - Weitere Hilfsfunktionen für indizierte Zugriffe in Felder

```
peekElemOff :: Storable a=> Ptr a-> Int-> IO a
pokeElemOff :: Storable a=> Ptr a-> Int-> a-> IO ()
```

Memory Management I: `ForeignPtr`

- Das Problem:
  - In imperative Sprache explizite Finalisierung (Resourcefreigabe)
  - Aber in Haskell: implizite *garbage collection*

- Lösung: `ForeignPtr a` mit Finalizer
  - Wird aufgerufen, bevor Objekt abgeräumt wird.

```
data ForeignPtr a
newForeignPtr :: Ptr a-> IO ()-> IO (ForeignPtr a)
```

Memory Management II: `StablePtr`

- Noch einProblem:
  - Imperative Sprachen erwarten unveränderliche Referenzen
  - in Haskell sind Adressen flüchtig (*garbage collection*)

- Die Lösung: `StablePtr a`
  - Stabile Adresse garantiert.

```
data StablePtr a
newStablePtr :: a-> IO (StablePtr a)
deRefStablePtr :: StablePtr a-> IO a
freeStablePtr :: StablePtr a-> IO ()
```

From Heaven to Hell

Haskell importiert C:

```
module Main where

foreign import ccall "hello.h hello" helloWorld :: IO ()

main =
     do putStrLn "Here you go: "
        helloWorld
```

- Name der C-Quelldatei unerheblich
- Benötigte Deklarationen in `hello.h`
- Übersetzen:

```
ghc -c -ffi Heaven2Hell.hs
gcc -c hello.c
ghc -o hello Heaven2Hell.o hello.o
```

From Hell to Heaven

Haskell-Modul mit exportierter Funktion:

```
module Hell2Heaven where

foreign export ccall "fac" fac :: Int-> Int

fac :: Int-> Int
fac n | n <= 0    = 1
      | otherwise = n* fac (n-1)
```

- Compilation mit `ghc` erzeugt:

```
Hell2Heaven_stub.c -> Hell2Heaven_stub.o
Hell2Heaven_stub.h
```

- Einbinden von `fac` in `callHs.c`:
  - `Hell2Heaven_stub.h` enthält Signatur (Prototypen)
  - Haskell-Laufzeitsystem initialisieren/beenden
  - Alle Haskell-Funktionen in einem Modul zusammengefassen

- Kompilieren:

```
ghc -c -ffi Hell2Heaven.lhs
gcc -I /usr/local/lib/ghc-5.04.3/include \
    -c callHs.c
ghc -no-hs-main -o callHs callHs.o \
       Hell2Heaven.o Hell2Heaven_stub.o
```

  - Pfad auf `ghc`-includes systemspezifisch
  - Linken mit `ghc`, aber `main` aus `callHs.o`

Generell zu beachten

- Typen werden nicht überprüft
- Beliebige Typkonversionen möglich
- Fremdfunktionen können Haskell-Heap korrumpieren

Auf dem FFI aufsetzende Tools

- `GreenCard`
  - Generiert Marshalling aus Direktiven
  - Kein Export
- `C->Haskell`
  - Generiert Marshalling aus Haskell mit Direktiven und C Headerdateien
- `H/Direct`
  - Generiert Marshalling aus IDL
  - Import und Export; unterstützt für Java, C, COM+
  - Groß und etwas unhandlich

Zusammenfassung

- Das Foreign Function Interface (FFI)
  - Standardisiert
  - Einfach gehalten
  - marshalling in Haskell
- Externe Funktionen aus Haskell aufrufen.
- Funktionen aus Haskell heraus exportieren.
- Andere Werkzeuge für das FFI.
- Nächste Woche: Speed!