Haskell und Debian

Joachim Breitner *
Karlsruher Institut für Technology
breitner@kit.edu

Abstract

Da die Programmiersprache Haskell immer beliebter wird und in immer weiteren Kreisen eingesetzt wird, müssen sich nun immer mehr Programmierer auch um die Besonderheiten der Installation von Haskell-Compilern und -Bibliotheken kümmern. Dies zu vereinfachen und vereinheitlichen ist die originäre Aufgabe von Distributionen wie Debian. Dieser Vortrag gibt einen Einblick in die Funktionsweise von Debian, insbesondere in Bezug auf Haskell-Pakete, erklärt was man als Anwender von Debian erwarten kann und was sich Debian von den Haskell-Bibliotheken wünscht.

Categories and Subject Descriptors K.6.3 [*Management of Computing and Information Systems*]: Software Management—Software process

Keywords linux distribution, package management, Debian

1. Was ist Debian?

Der Titel des Vortrages ist „Haskell und Debian“. Was Haskell ist muss ich hier wohl nicht erklären; Debian[1] wird dagegen dem ein oder anderen nicht geläufig sein. Debian bezeichnet eigentlich zwei Sachen: Zum einen die Linux-Distribution Debian, zum anderen das Debian-Projekt, das diese erstellt.

1.1 Eine Linux-Distribution

Das Konzept einer Distribution sollte allen Linux-Anwendern bekannt sein, Anwendern anderer Systeme aber vielleicht nicht, daher will ich es kurz erklären:

Um seinen Rechner vernünftig benutzen zu können braucht man eine Vielzahl von Programmen: Linux, also den eigentlichen Betriebssystemkern, Systemprogramme, Programmierumgebungen, Anwendungen und, nicht zu vergessen, Spiele. Diese werden alle von verschiedenen Leuten und Gruppen hergestellt und meist nur als Quellcode-Pakete vertrieben. Würde man diese direkt verwenden wollen müsste man, um sein System in Betrieb zu nehmen, von Dutzenden verschiedenen Webseiten hunderte verschiedene Pakete herunterladen, die Anleitung lesen, die Abhängigkeiten zwischen den Paketen auflösen, den Code kompilieren und installieren. Selbiges erneut bei Aktualisierungen, und das Deinstallieren von Programmen wäre auch nicht möglich.

An dieser Stelle setzen Distribution wie Debian an: Die Entwickler der Distribution laden die Quellen bei den Autoren herunter, kompilieren sie und packen das Ergebnis in *Pakete* – das sind die Dateien mit Endung `.deb`. Diese werden zusammen in einem *Repository* angeboten und können alle auf die gleiche Weise direkt installiert werden: Ein einfaches `apt-get install ghc` genügt, und der Haskell-Compiler ist installiert. Aktualisierungen laufen problemlos mit `apt-get upgrade` und nicht benötigte Pakete können mit `apt-get remove` sauber und vollständig entfernt werden. Wenn ein zu installierendes Paket ein anderes benötigt, wird dieses automatisch mit installiert und auch im weiteren Betrieb werden diese Abhängigkeiten stets geprüft. Kryptographie schützt den Anwender dabei vor bösartig veränderten Paketen.

Darüber hinaus prüfen die Debian-Entwickler dass die Programme auch wirklich Freie Software sind. Man kann sich also darauf verlassen, dass man zu jedem Paket in Debian die Quellen bekommt, diese verändern darf und die Veränderungen auch weitergeben darf. Auch das geht mit einheitlichen Befehlen (`apt-get source` und `dpkg-buildpackage`).

Ich erinnere mich noch vage an die Zeit als ich unter Windows Software noch auf irgendwelchen Webseiten direkt heruntergeladen habe und mich durch immer verschiedene Installer geklickt hatte. Inzwischen erreicht diese zentrale Art der Softwareverteilung, wie es sie unter Linux schon 15 Jahre gibt, als „Appstore“ getauft auch Nicht-Linux-Anwender.

Debian bietet drei Versionen der Distribution an: *unstable*, *testing* und *stable*. Letzteres ist die zuletzt veröffentliche stabile Debian-Version, gerade Frisch Debian 6, genannt „Wheezy“. Diese ändert sich nach der Veröffentlichung bis auf Sicherheitsupdates nicht mehr und empfiehlt sich daher insbesondere für den Server-Betrieb oder auch für Arbeitsplätze, die mit minimalem Administrationsaufwand laufen sollen. In Debian testing ist das, was das nächste stable, Codename „Jessie“, werden soll. Nach Debian unstable, auch „sid“ genannt, werden laufend neue Paketversionen hochgeladen. Das ist also die richtige Version für technisch etwas versiertere Benutzer, die stets auf dem neusten Stand sein möchten. Es ist übrigens nicht so instabil wie der Name es glauben lässt: Ich benutze es seit über zehn Jahren ohne nennenswerte Probleme auf meinem Arbeitsrechner. Außer kurz vor einem Release wandern Pakete aus unstable nach 10 Tagen automatisch nach testing, falls kein schwerwiegender Fehler entdeckt wurde.

Die Menge an Software in Debian ist gewaltig: Debian Squeeze enthält etwa 30.000 Pakete und füllt 8 DVDs – zum Glück installiert man so ein System heutzutage per Internet.

1.2 Ein großes Projekt

Wie erwähnt ist Debian aber auch ein großes Projekt: Über 1000 freiwillige und ehrenamtliche Entwickler arbeiten daran, dass Debian (die Distribution) ein vielseitiges, umfangreiches und technisch hervorragendes System ist. Die meisten engagieren sich als Paket-Maintainer, also jene, die die Quellpakete der Softwareautoren nehmen, prüfen, testen, ggf. patchen, in Debian einpflegen und dann die Bugreports der User entgegen nehmen. Dabei herrscht weitgehend konstruktive Anarchie: Jeder Entwickler ist, im Rahmen der Policy, sehr frei in

* Unterstützt durch die Deutsche Telekom Stiftung

[1] http://www.debian.org

HaL8, 21. Juni 2013, Leipzig.

seiner Arbeit. So wird zum Beispiel nicht zentral gesteuert oder geprüft, welche Software überhaupt in Debian aufgenommen wird, sondern die Entwickler entscheiden das selbst nach eigenem Bedarf und Interesse.

Software, die benötigt wird, um andere Software zu bauen, muss dabei auf jeden Fall auch in Debian aufgenommen werden. Zum einen ist das wichtig, da es eines der Versprechen von Debian ist, dass der Anwender seinen Recht auf das Ändern der Software auch praktisch ausüben kann. Zum anderen ist Debian auf einem Dutzend Architekturen verfügbar. Damit das möglich ist müssen alle Debian-Pakete automatisch gebaut werden können.

Neben den Debian-Entwicklern gibt es auch Mitarbeiter, die sich um die Infrastruktur kümmern, die Übersetzungen und Dokumentationen erstellen, die Pressearbeit machen und vieles mehr.

Es gibt etliche Distributionen, die auf Debian aufbauen, allen voran Ubuntu, für die das im Folgenden gesagte ebenso gilt. Andere Distributionen wie Fedora, OpenSUSE, Arch oder NixOS handhaben Haskell gegebenenfalls etwas anders.

2. Haskell-Pakete: Cabal, Hackage und cabal-install

Auch ohne eine Linux-Distribution hat man es als Haskell-Programmierer schon ganz gut. So werden (fast) alle Haskell-Bibliotheken als *Cabal*-Paket vertrieben. Cabal ist sowohl ein Standard, wie die Quell-Pakete aufgebaut sein sollen, als auch eine Bibliothek, mit der solche Pakete sich kompilieren. Das wichtigste Element dabei ist die foo.cabal-Datei, die es im obersten Verzeichnis eines Cabal-Paketes geben muss. Ein Beispiel ist in Abbildung 1 zu sehen. Diese Datei beschreibt eine Bibliothek (`library`), es können aber auch Haskell-Programme so beschrieben werden. Darin wird unter anderem festgelegt, welche Module das Paket enthalten soll (`exposed-modules`) und welche anderen Pakete in welchen Versionen vorhanden sein müssen (`build-depends`).

In dem Quellpaket existiert auch eine `Setup.hs` oder `Setup.lhs`-Datei. Diese ist bei fast allen Haskell-Paketen identisch und verwendet die Bibliothek Cabal. Um das Paket zu installieren, kompiliert man nun das Setup-Programm und führt den Dreischritt `./Setup configure --user && ./Setup build && ./Setup install` aus. Dank der Option `--user` wird die kompilierte Bibliothek ins Benutzerverzeichnis installiert, statt systemweit. Die genannten Abhängigkeiten muss man dabei schon installiert haben, sonst scheitert der Konfigurationsschritt. Der Installationsschritt kopiert nicht nur die gebauten Dateien an den richtigen Ort, sondern registriert auch das Paket in der GHC-eigenen Paketdatenbank, damit der Compiler die Module dann auch findet. Mit dem zusätzlichen Schritt `./Setup haddock` kann man die Dokumentation der Bibliothek erstellen.

Es ist erfreulich dass alle Haskell-Pakete sich so installieren lassen. Aber es ist immer noch recht mühsam, und ein komplexeres Paket wie der Haskell-Webserver *yesod* hat über 50 Abhängigkeiten, die man alle finden, herunterladen und in der richtigen Reihenfolge installieren muss.

Das Finden ist dabei noch das Einfachste: Dafür gibt es Hackage[2], ein zentrales Sammelsurium (fast) aller Haskell-Pakete. Es ist vergleichbar mit CPAN für Perl und CTAN für TeX und enhält zur Zeit 5093 Pakete in insgesamt 30040 verschiedenen Versionen von 1161 Entwicklern[3]. Jeder kann,

```
name:          void
category:      Data Structures
version:       0.5.12
license:       BSD3
cabal-version: >= 1.6
license-file:  LICENSE
author:        Edward A. Kmett
maintainer:    Edward A. Kmett <ekmett@gmail.com>
stability:     portable
homepage:      http://github.com/ekmett/void
bug-reports:   http://github.com/ekmett/void/issues
copyright:     Copyright (C) 2008-2012 Edward A. Kmett
synopsis:      A Haskell 98 logically uninhabited data ↵
     type
description:   A Haskell 98 logically uninhabited data ↵
     type, used to indicate that a given term should not ↵
     exist.
build-type:    Simple

extra-source-files: .travis.yml CHANGELOG.markdown ↵
     README.markdown

source-repository head
  type: git
  location: git://github.com/ekmett/void.git

flag safe
  manual: True
  default: False

library
  exposed-modules:
    Data.Void
    Data.Void.Unsafe

  build-depends:
    base       >= 3 && < 10,
    semigroups >= 0.8.2

  ghc-options: -Wall

  if flag(safe)
    cpp-options: -DSAFE

  if impl(ghc)
    extensions: DeriveDataTypeable
    cpp-options: -DLANGUAGE_DeriveDataTypeable
```

Abbildung 1. Die Dabei `void.cabal`

nach einer kurzen Registrierung per Mail, seine Entwicklungen dort hochladen, eine Auswahl oder Qualitätskontrolle findet nicht statt.

Aber auch die anderen Schritte lassen sich automatisieren: Mit *cabal-install*, welches einen Befehl namens `cabal` bereitstellt, kann man ein Paket wie *yesod* samt allen Haskell-Abhängigkeiten mittels `cabal install yesod` installieren. Dabei werden, sofern möglich, bereits installierte Pakete weiter verwendet, aber falls nötig auch aktualisiert. Es ist dabei problemlos möglich, mehrere verschiedene Version des selben Pakets installiert zu haben.

3. Haskell als Debian-Pakete

Damit allerdings Debian-Anwender das nicht zu wissen brauchen werden auch Haskell-Bibliotheken – wie andere Software auch – zu Debian-Paketen verschnürt. Debian enthält die etwa 600 wichtigsten Haskell-Pakete, die jeweils auf drei Debian-Pakete aufgeteilt sind: Die Pakete für die Haskell-

[2] http://hackage.haskell.org

[3] http://hackage.haskell.org/cgi-bin/hackage-scripts/stats, 26.4.2013

Bibliothek *void* heißen `libghc-void-dev` für die kompilierte Bibliothek, `libghc-void-prof` für die für das Profiling nötigen zusätzlichen Daten und `libghc-void-doc` für die Dokumentation. Hier sind, wie bei allen Debian-Paketen, die Abhängigkeiten explizit angegeben, so dass ein `apt-get install yesod` automatisch 159 Pakete installiert, inklusive `GHC` und `gcc`, und dafür 600 MB braucht.

Die Auswahl der zur Zeit 600 Pakete folgt, wie auch sonst in Debian, keinen strengem Muster. Wir paketieren alles von dem wir glauben, dass es viele Anwender haben wollen: Alles was in der Haskell Platform[4] enthalten ist, Bibliotheken, die wir selbst mal gebraucht haben, aktuelle Trends wie die Webframeworks *happstack*, *yesod* und *snap*. Außerdem kümmern wir uns um eine Reihe von Anwendungen, die in Haskell implementiert sind: Das Versionskontrollsystem *Darcs*, den Haskell-Stil-Prüfer *HLint*, den Fenstermanager *xmonad*, das Wiki *gitit*, die Dateiverwaltung *git-annex*, die Zeiterfassung *arbtt*, die Buchhaltungssoftware *hledger* und andere. Auch hier gilt: Alle Abhängigkeiten dieser Pakete müssen auch in Debian enthalten sein.

Generell wollen wir die Software, Bibliotheken wie Anwendungen, dem Anwender unverändert zur Verfügung stellen. Sollte es aber im Interesse unserer Anwender sein, scheuen wir uns auch nicht, den Code zu patchen. Häufig müssen wir etwa die in den Cabal-Dateien spezifizierten Abhängigkeiten ändern, damit alle in Debian enthaltenen Versionen zueinander kompatibel sind. Manchmal ändern wir auch den Code, um ihn etwa mit der neusten Compilerversion bauen zu können. Und insbesondere in Debian stable, wo keine Versionssprünge mehr erlaubt sind, versuchen wir sicherheitskritische Fehler auch in den alten Versionen zu beheben, so zuletzt bei *tls-extra*[5].

Auch versuchen wir, die Haskell Platform exakt nachzubauen, also von den darin enthaltenen Bibliotheken und Tools auch genau die in der Platform angegebenen Version in Debian zu haben. Nicht immer gelingt uns das, etwa wenn ein anderes Paket, das wir haben wollen, eine neuere Version eines Pakets aus der Platform benötigt – dann weichen wir auch mal leicht davon ab. Das Debian-Paket mit dem Namen `haskell-platform` installiert also immer die Pakete aus der Haskell Platform, aber jeweils in der Version, die Debian ausgewählt hat. Sobald *Stackage*[6], eine Art stabile und getestete Auslese von Paketversionen von Hackage, einigermaßen in Gang gekommen ist werden wir uns wohl auch stark daran orientieren, was uns vor allem Arbeit ersparen wird.

Die Debian-Haskell-Group[7] kann immer Unterstützung bei ihrer Arbeit brauchen. Neben dem Paketieren, Bauen und Hochladen von neuen Paketen geht es dabei auch darum, die Fehlerberichte zu bearbeiten und gegebenenfalls Bugs zu fixen. Außerdem können unsere eigenen Tools, etwa das, was erkennt, welche Pakete neu gebaut werden müssen, intensivere Pflege gebrauchen. Interesant ist auch die systemweise Integration von Tools wie *hoogle*: Damit kann man auch ohne Internetverbindung in allen installierten Paketen nach Funktionsnamen und -signaturen suchen. Wer Lust hat zu helfen melde sich bitte auf unserer Mailingliste[8].

[4] http://www.haskell.org/platform/

[5] http://bugs.debian.org/698545

[6] https://github.com/fpco/stackage

[7] http://wiki.debian.org/Haskell

[8] http://lists.debian.org/debian-haskell/

3.1 Vor- und Nachteile

Auf den ersten Blick bietet Haskell in Debian auch nicht mehr als die Installation mit *cabal-install*, warum sollte man also nun die Debian-Pakete verwenden? Es gibt eine Reihe von Vorteilen:

- Die Pakete sind schon compiliert. *Yesod* per *cabal-install* zu installieren braucht, auf meinem Laptop, 9,5 Minuten. Die Installation via `apt-get` ist in unter einer Minute abgeschlossen. Außerdem kann der Paketbau mit *cabal-install* auch fehlschlagen, etwa weil eine neue Version einer verwendeten Bibliothek die API geändert hat. Bei Debian wird so etwas bereits von den Debian-Entwicklern festgestellt und behoben.
- Abhängigkeiten jenseits der Haskell-Welt werden in Debian-Paketen abgebildet und automatisch mitinstalliert. Würde man auf einer Linux-Minimalinstallation dagegen versuchen, *yesod* mit *cabal-install* zu installieren, würde das scheitern: Es fehlen die Entwicklerdateien zur C-Bibliothek *zlib*.
- Die Pakete lassen sich wieder deinstallieren. Das kann *cabal-install* noch nicht – man kann lediglich *alle* installierten Pakete mittels `rm -rf ~/.cabal ~/.ghc` deinstallieren.
- Mit *cabal-install* kann es passieren, dass zur Installation eines Paketes ein bereits installiertes neu kompiliert werden muss. Hängt ein weiteres bereits installiertes Paket davon ab, wird jenes danach unter Umständen nicht mehr funktionieren und muss erneut installiert werden. Dieses Problem ist auch als *Cabal Dependency Hell* bekannt. Da Debian-Pakete vorkompiliert sind kann das nicht passieren.
- Manchmal findet *cabal-install* bei der Installation eines neuen Pakets keine Paketauswahl, in der alle Abhängigkeiten erfüllt sind (auch eine Variante der Cabal Dependency Hell). Auch solche Probleme werden in Debian schon von den Entwicklern gelöst.
- Man kann auch noch nachträglich die Profiling-Daten oder die Dokumentation installieren, während man mit *cabal-install* schon bei der Installation die entsprechenden Flags an `cabal install` übergeben muss.
- Jedes Paket ist in Debian in der Regel in genau einer Version vorhanden, mit der alle anderen Pakete können.
- Eine Man-in-the-Middle-Attacke könnte problemlos die Pakete, wie sie *cabal-install* von Hackage herunterlädt, durch eigene ersetzen. Mit Debian geht das dank signierter Paketlisten nicht.
- Debian trifft eine Vorauswahl an Paketen; bei Paketen in Debian ist die Chance größer dass es ein benutzbares und einigermaßen verbreitetes Paket ist.

Allerdings hat auch *cabal-install* seine Vorteile:

- Es kann alle knapp 5000 Pakete auf Hackage installieren, während in Debian nur die wichtigsten 600 Pakete enthalten sind.
- Man braucht keine Root-Rechte, um damit Pakete zu installieren.
- Man kann die Compile-Flags setzten, mit denen man manche Pakete genauer konfigurieren kann.
- *Cabal-install* kann mehrere Versionen desselben Paket installieren. Das ist gut, weil vielleicht ein Paket eine eine

ältere und ein anderes eine neuere Version einer Bibliothek benötigt.

- Es gibt Erweiterungen zu *cabal-install* fürs Sandboxing, also um Pakete getrennt voneinander zu installieren und beispielsweise mehrere GHC-Versionen gleichzeitig testen zu können.

3.2 Das Problem mit den ABIs

Generell sind Haskell-Pakete dankbare Ziele zum Paketieren, da sie sehr homogen sind. Das ermöglichst es dem Debian Haskell Team, mit vergleichsweise wenig Manpower viele Haskell-Pakete zu betreuen.[9] Es gibt aber ein paar Eigenheiten von Haskell und dem Haskell-Compiler GHC, die das Paketieren von Haskell für Debian knifflig oder aufwendig machen.

Das größte Problem ist, dass kompilierter Haskell-Code kein stabiles ABI hat. Das *Application Binary Interface* einer Bibliothek beschreibt, wie das Kompilat zu verwenden ist: Wie die Funktionen heißen und wie sie aufgerufen werden. Bei C-Bibliotheken ist die ABI sehr stabil: Ich kann eine neue Version einer Bibliothek bauen und installieren, und solange sich die vorhandenen Funktionssignaturen und Datenstrukturen nicht geändert haben, bleibt das ABI gleich und Programme, die gegen meine Bibliothek linken, müssen nicht neu gebaut werden. Bei Haskell kann es passieren, dass schon kleinste Änderungen an der Implementierung einer Funktion oder andere Compiler-Flags dafür sorgen, dass sich das ABI ändert und aller installierter Code, der meine Bibliothek verwendet, neu kompiliert werden muss. Ein Hauptgrund dafür sind Cross-Module-Optimierungen wie Inlining: Um eine Funktionsdefinition inlinen zu können muss der zugehörige Code Teil des ABI sein.

Wer alles per *cabal-install* selbst kompiliert kann dann ja einfach alles, was kaputt gegangen ist, neu kompilieren. Wer dagegen Haskell-Bibliotheken über Debian installiert, kann ja schlecht neue Debian-Pakete bauen. Also müssen die Debian-Entwickler das machen. Außerdem muss sichergestellt werden, dass die Anwender unter keinen Umständen kaputte Pakete installieren können – lieber soll ihnen eine Installation oder ein Update verwehrt werden, bevor inkompatible Pakete auf dem Rechner landen. Dies wird über ein System von virtuellen Paketen ermöglicht:

GHC berechnet für jedes Paket einen Prüfsumme, die das ABI beschreibt. Ändert sich das ABI, ändert sich auch die Prüfsumme. Zusammen mit dem Paketnamen und der Versionsnummer ergibt sich so die Package-ID, die z.B. `void-0.5.11-713ea60ae362e86803206553ab70774c` lauten kann. In der GHC-Paketdatenbank ist dazu vermerkt, gegen welches ABI von semigroups das Paket gebaut wurde, etwa `semigroups-0.9-2947823d7c0a6064bf9c08818332909a`. So weiß der Compiler, dass void nicht mehr benutzbar ist, wenn semigroups nicht mehr mit exakt diesem ABI vorhanden ist.

Diese Informationen werden beim Bau des Debian-Paketen extrahiert und in dessen Metadaten übernommen: Das Debian-Paket `libghc-semigroups-dev` hat einen Eintrag `Provides: libghc-semigroups-dev-0.9-29478`, der die ersten paar Ziffern der Prüfsumme enthält. Damit existiert nun ein virtueller Paketname für genau dieses ABI, und das Paket `libghc-void-dev` kann von diesem Paket abhängen. Die Paketverwaltungssoftware `dpkg` und `apt-get` stellt damit sicher, dass man `libghc-void-dev` nur installieren kann, wenn ein passendes `libghc-semigroups-dev` Paket installiert ist. Würde nun eine neue Version von `libghc-semigroups-dev` nach Debian hochgeladen, würde ein Update dieses trotzdem nicht installieren, bis `libghc-void-dev` neu gebaut ist. Jemand, der keines der Pakete installiert hat, kann die neue Version von *semigroups* installieren, nicht aber *void*.

Anhand dieser Metadaten können die Entwickler ablesen, welche Pakete noch neu gebaut werden müssen, damit wieder alles installierbar ist. Das Neu-Kompilieren selbst geht dabei automatisch über das Debian-eigene *buildd*-Netzwerk.

Wer diesen Ansatz nochmal in Formeln erklärt sehen will: Die Debian-OCaml-Maintainer, die ihn auch gewählt haben, haben darüber ein wissenschaftliches Paper[10] geschrieben.

4. Wunschliste an Haskell-Entwickler

Da im Publikum sicherlich einige Haskell-Entwickler sitzen, und viele die es werden wollen, ist dies eine gute Gelegenheit ein paar Wünsche loszuwerden, die es uns einfacher machen, ihre Pakete in Debian aufzunehmen, und auch sonst gute Praxis sind:

- Verwenden Sie Cabal und veröffentlichen Sie ihre Pakete auf Hackage. Je homogener die Pakete sind, desto einfacher ist es für uns, eine große Zahl von ihnen zu betreuuen.
- Geben Sie ihren Paketen treffende Beschreibungen. Cabal bietet dafür das Feld `synopsis` für eine einzeilige Beschreibung und `description` für einen mehrzeiligen Text. Formulieren Sie die Beschreibung so, dass sie jeder Haskell-Programmierer versteht und einordnen kann, auch jene ohne Informatikstudium oder fachspezifisches Wissen.
- Kümmern Sie sich um die LICENSE-Datei. Debian legt großen Wert darauf, die Copyright-Situation der Software zu prüfen und korrekt wiederzugeben. Das heißt, wenn Sie Code aus anderen Paketen übernehmen, beachten sie dessen Lizenz und geben Sie alle Autoren mit Copyright an.
- Halten Sie sich an die Package Versioning Policy[11]. Dadurch erkennen wir frühzeitig, mit welchen Versionen anderer Pakete Ihr Code zusammenarbeitet und sparen uns unnötige Bauversuche.
- Schreiben Sie Tests und geben Sie diese also solche in der Cabal-Datei an. Dann können wir diese bei jedem Paketbau durchlaufen lassen und z.B. auch Fehler finden, die nur auf seltsamen Architekturen auftreten (s390 hat 31 Bits…).

5. Fazit

Haskell ist erfolgreich genug, dass eine Distribution wie Debian es aufnehmen muss, und reif genug, dass das auch gut geht. Für Endwandender heißt das dass es keinen Unterschied macht, ob die Software, die er installieren will, in Haskell geschrieben ist: Das Programm kann für Debian paketiert werden, und wenn es das ist, ist es gginstallierbar wie jedes andere Programm auch.

[9] http://lists.debian.org/1361126251.4322.18.camel@kirk

[10] Mehdi Dogguy, Stephane Glondu, Sylvain Le Gall, Stefano Zacchiroli: Enforcing Type-Safe Linking using Inter-Package Relationships, JFLA10, pp. 29-54, 2010

[11] http://www.haskell.org/haskellwiki/Package_versioning_policy