Yi

Tero Jäntti
tekrjant@jyu.fi

22.4.2008

Tiivistelmä

Yi on Haskellilla toteutettu tekstieditori, joka on myös laajennettavissa sekä kustomoitavissa Haskellilla. Editori on hyvin modulaarinen; Esimerkiksi näppäinkomennot voi valita Emacsin ja Vi:n vastaavien välillä. Yin dynaamista linkitystä käyttävä rakenne sekä funktionaalinen toteutus mahdollistavat editorin laajentamisen sekä päivittämisen editoria sulkematta.

1 Johdanto

Yi on Haskell-kielellä toteutettu tekstieditori, joka on julkaistu avoimen lähdekoodin lisenssillä. Yi:n web-sivujen mukaan tavoitteena on luoda joustava, tehokas ja oikeellinen editorin ydin, joka on dynaamisesti muokattavissa Haskellilla [1]. Kirjoitushetkellä editori on melko keskeneräinen, mutta kehitys on aktiivista. Kirjoittaja on testannut Yi:n versiota `0.3`.

Yi muistuttaa varsin paljon Emacs-editoria mutta mukana on myös mm. vi-editorin näppäinkomennot. Emacsiin nähden Yi:hin suunnitellaan parempaa tukea parsereille ja sitä kautta parempia staattisia koodin analysointityökaluja. Tekijät myös uskovat Haskellin olevan parempi ohjelmointikieli tähän tarkoitukseen kuin Emacsissa käytetty Emacs Lisp. [1]

2 Ominaisuuksia

Yi:ssa on monia ominaisuuksia, joita hyvältä tekstieditorilta odottaisi, mm. syntaksin väritys, hakutoiminnot sekä hakemistojen käsittely. Tekstiä käsitellään puskureissa (buffer), joita voi olla auki useita kerrallaan. Käyttöliittymänä on komentopääte tai graafisen käyttöliittymän ikkuna, joka tosin ei nykyisellään tuo juuri mitään lisäarvoa ensiksimainittuun verrattuna. Käyttöliittymä on toteutettu erillisenä dynaamisesti ladattavana modulina.

Ohjelman konfigurointi tapahtuu Haskellia kirjoittamalla. Tämä on hyödyllistä, koska asetusten säilyttämiseen ei tarvita erillistä tiedostoformaattia ja koska Haskell on (ainakin minun mielestäni) varsin ilmaisuvoimainen kieli.

Käyttäjä voi muuttaa asetuksiaan tekemällä modulin `~/.yi/YiConfig.hs`, jossa konfiguraatio asetetaan yiMain-nimisen monadin kautta (monadi on tyyppiä `YiM ()` — katso luku 4). Yi:n mukana tulee useita esimerkkejä konfiguraatiotiedoston tekemiseen.

3 Näppäinkomentojen määrittely

Näppäinkomennot määritellään parserifunktioina, jotka lukevat käyttäjän syötettä ja sen perusteella palauttavat `Action`-muuttujia. `Action` on synonyymi tyypille `IO ()`. Näppäinmääritykset ovat siten tyyppiä:

```
keymap :: [Char] -> [Action]
```

Tälläinen funktio on käyttökelpoinen Haskellin laiskan laskennan ansiosta. Yi tarjoaa erilaisia operaatioita eri parsereiden yhdistämiseen, joiden ansioista parsereita voidaan määritellä eri tiedostoissa ja niiden yhdistäminen on selkeää ja vaivatonta. Steward ja Chakravarty toteavatkin, että em. operaatiot muodostavat oman kielensä näppäinkomentojen määrittämiseen (embedded domain specific language). [3]

Seuraava esimerkki Stewardin ja Chakravartyn paperista [3] määrittelee joitain ee-editorin näppäinkomentoja. Operaattori `>||<` yhdistää kaksi parseria, `insert` sekä `command`. `insert`:in määrityksessä `any` vastaa mitä tahansa merkkiä, jolle `action`-operaattorilla määritellään insert-toiminto. Vastaavasti `command` määritellään siten, että ohjausmerkkejä luettaessa (`cmd`) valitaan suoritettava toiminto ohjausmerkin perusteella.

```
keymap cs = fst3 $ execLexer lexer (cs, ())

lexer = insert >||< command

insert = any ‘action‘ \[c] -> Just (insertE c)

command = cmd ‘action‘ \[c] -> Just $ case c of
    ’\^L’ -> leftE
    ’\^R’ -> rightE
    ’\^U’ -> upE
    ’\^D’ -> downE
    ’\^B’ -> botE
    ’\^T’ -> topE
    ’\^K’ -> deleteE
    ’\^Y’ -> killE
    ’\^H’ -> deleteE >> leftE
    ’\^G’ -> solE
    ’\^O’ -> eolE
    ’\^X’ -> quitE
    _ -> undefined
```

4 Tietorakenteet

Tekstieditori on ohjelma jonka tila muuttuu varsin tiuhaan, mm. näppäinpainallusten jälkeen. Koska Yi on ohjelmoitu puhtaasti funktionaalisella kielellä, ohjelman tilaa ei varsinaisesti muuteta, vaan ”välitetään” eteenpäin. Niinpä ohjelman tilatieto ei ole hajallaan eri moduleissa, vaan koottuna yhteen tietorakenteeseen, nimeltään `Yi`, joka karkeasti ottaen on seuraavanlainen (joitain olennaisia osia jätetty pois) [2]:

- `Yi`
  - `yiEditor :: IORef Editor`
    - `buffers :: M.Map BufferRef FBuffer`
    - `windows :: WindowSet Window`
    - `tabwidth :: !Int`
    - ...
  - `yiUi :: UI`
  - `threads :: IORef [ThreadId]`
  - `input :: Chan Event`
  - `output :: Chan Action`
  - ...

Tilan muutokset on luonnollisesti toteutettu monadeilla. Kun haluamme muuttaa `Yi`:tä, `Editor`:ia tai `Bufferia`, muutoksia vastaavat monadit ovat:

- `YiM ()`
- `EditorM ()`
- `BufferM ()`

5 Funktioiden määrittely

Käyttäjä voi konfiguraatiotiedostoissaan määritellä funktioita, joilla editorin toimintaa voi muokata mieleisekseen. Tarkastellaan esimerkkinä funktiota, joka asettaa tekstipuskuriin Latex-lohkon:

```
insertLatexBlock :: String -> BufferM ()
insertLatexBlock b = do
  insertN ("\\begin{" ++ b ++ "}\n")
  savingPointB $ insertN ("\n\\end{" ++ b ++ "}")
```

Funktio on paluuarvoltaan `BufferM ()`, joten se on tekstipuskuria käsittelevä funktio. Parametina on lohkon nimi (esim. `verbatim` tai `enumerate`). `insertN` asettaa tekstiä puskuriin. `savingPointB` on funktionaali, joka säilyttää kursorin paikan. Toisin sanoen se palauttaa kursorin siihen kohtaan puskuria, missä se ennen parametrina annetun funktion suoritusta oli. Näin kursori jää sinne minne pitääkin, eli `begin`- ja `end`-komentojen väliin.

Tämä funktio on jo sellaisenaan käytettävissä editorissa, kunhan se on jossakin konfiguraatiomodulissa ja lueteltu modulin rajapinnassa. Funktiota voi käyttää valitsemalla komentokehote (painamalla alt-x) ja kirjoittamalla `insertLatexBlock`. Tällöin Yi kysyy merkkijonovakiota, jonka syöttämällä haluttu lohko asetetaan puskuriin. Käytettävyyttä voitaisiin vielä parantaa seuraavasti:

```
latexBlock :: YiM ()
latexBlock = withMinibuffer "Lohko:" return
               (withBuffer . insertLatexBlock)
```

Tämä funktio kysyy käyttäjältä lohkon nimeä ja käyttää edellistä funktiota lohkon asettamiseen. `withMinibuffer` ottaa parametreinaan käyttäjälle näytettävän kehotteen, täydennysfunktion sekä funktion, joka ottaa parametrinaan käyttäjän syötteen. Koska `withMinibuffer` on tyyppiä `YiM ()`, asetetaan myös tämän funktion tyypiksi `YiM ()`. Koska `withMinibuffer`-funktion viimeinen parametri on tyyppiä `YiM ()`, sovitetaan `insertLatexBlock` sen tyyppiseksi `withBuffer`-funktionaalilla.

Täydennysfunktio, jonka tyyppi on `String -> YiM String`, täydentää käyttäjän syötteen tabulaattoria painettaessa. Tässä täydennysfunktiona on `return`, joten täydennystä ei tapahdu. Parempi tietysti olisi tarjota funktio, joka täydentää Latex-lohkojen nimiä.

6 Dynaaminen lataus

Eräs Yi:n mielenkiintoisista ominaisuuksista on mahdollisuus laajentaa tai päivittää editoria sulkematta sitä. Tämä on mahdollista, koska ohjelmalla on pieni (noin sata koodiriviä) staattisesti linkitetty ydin. Staattisen ytimen tehtävänä on moduleiden lataaminen. [3]

Kun ohjelma (staattinen ydin) käynnistetään, se lataa konfiguraatiomodulit sekä varsinaisen editorin koodin ja kutsuu tämän `main`-funktiota, antaen sille parametrina konfiguraation. Myös editorin laajennokset, kuten erilaiset käyttöliittymät tai näppäinasetukset, ladataan dynaamisesti.

Editorin uudelleenkonfigurointi tapahtuu pääpiirteissään seuraavasti [3]:

1. `reloadE`-funktiota kutsutaan.
2. `reloadE` kutsuu staattisen ytimen `reconf`-funktiota.
3. Staattinen ydin kääntää ne konfiguraatiomodulit, joissa on tapahtunut muutoksia.

4. Jos oli muutoksia, ladataan muuttuneet modulit ja palautetaan konfiguraatio `Just`-muodossa. Muuten palautetaan `Nothing`.

Lisäksi täytyy ottaa huomioon mm. tyyppien yhteensopivuus. Staattinen ydin nimittäin palauttaa konfiguraation polymorfisena arvona.

Kun halutaan ladata jokin laajennos tai peräti koko editori uudestaan, tilanne on hieman mutkikkaampi. Yi:n puhtaasti funktionaalinen toteutus on tässä oleellista. Staattisen ytimen `remain`-funktio suorittaa ohjelman uudelleen latauksen seuraavien vaiheiden kautta, jotka on esitetty Stewardin ja Chakravartyn paperissa [3]:

1. Funktiota kutsutaan dynaamisesta koodista parametrinaan ohjelman tilatieto (mukaanlukien luvussa 4 mainittu `Yi`-tyyppi).
2. Dynaaminen koodi poistetaan ohjelman instanssista.
3. Dynaaminen koodi (mahdollisesti uudempi versio) ladataan.
4. Staattinen ydin kutsuu tiettyä dynaamisen koodin funktiota, jolle annetaan tilatieto parametrina.

Staattinen ydin siis välittää tilatiedon uudelle dynaamisen ohjelmaosion instanssille. Näin käyttäjä voi jatkaa editointia siitä mihin hän jäi ennen ohjelman uudelleenlatausta. Aivan näin yksinkertainen uudelleenlatausprosessi ei tietenkään ole. Esimerkiksi, jos joihinkin koostetyyppeihin on uudessa ohjelmaversiossa tullut uusia kenttiä, ei vanhat arvot toimi sellaisenaan uudessa versiossa. Tämä on ratkaistu serialisoimalla tilatieto sopivaan muotoon, jonka lukuvaiheessa mahdolliset uudet kentät täytetään sopivalla oletusarvolla [3].

7 Yhteenveto

Yi on kehitysvaiheessa oleva editori, joka on laajennettavissa ja kustomoitavissa dynaamisesti, editoria sulkematta. Funktio-ohjelmoinnin rakenteet tekevät joidenkin asioiden määrittelystä suoraviivaista, kuten näppäinkomentojen määrittely syötteen parsereita yhdistelemällä. Funktionaalinen toteutus myös käsittelee ohjelman tilaa keskitetysti, mikä tekee editorin päivittämisen ajon aikana kohtuullisen helpoksi.

Nykyisellään Yi ei ole kovin helposti lähestyttävä. Aika näyttää, vetoaako Yi suurempaan yleisöön vai jääkö se haskellistien työkaluksi.

Viitteet

[1] Jean-Philippe Bernardy, Don Stewart ym., *Yi, kotisivu* URL: `http://www.haskell.org/haskellwiki/Yi`, viitattu 16.4.2008.

[2] Jean-Philippe Bernardy, Don Stewart ym., *Yi, lähdekoodi, versio 0.3* URL: `http://hackage.haskell.org/packages/archive/yi/0.3/yi-0.3.tar.gz`, viitattu 21.4.2008.

[3] Don Steward, Manuel M. T. Chakravarty, *Dynamic Applications From the Ground Up*, Programming Languages and Systems, School of Computer Science and Engineering, University of New South Wales, 2005.