CronFP

Students

Sahithyen Kanaganayagam

Luca Moor

Introduction: Job scheduling is a critical component in modern computing systems, enabling automated execution of tasks at specified times or in response to specific events. Traditional job schedulers, such as cron, are widely used but often rely on imperative programming paradigms that can lead to complex and hard-to-maintain configurations. This project explores the application of purely functional, declarative, and denotational programming concepts to the domain of job scheduling in Haskell.

Approach: We adopted a research-oriented approach, beginning with an analysis of existing scheduling systems to identify their advantages and limitations. We then explored functional programming concepts, particularly Functional Reactive Programming (FRP), Algebra-Driven Design, and Embedded Domain-Specific Languages (EDSLs), to design a scheduling framework. After this research phase, we decided to implement a clone of the cron job scheduler for gaining practical insights about the application of FRP. Additionally, we designed and implemented an Intermediate Language (IL) as an EDSL to explore first ideas for a more general scheduling framework supporting both time-based and webhook-based scheduling.

Conclusion: During our work we demonstrated how FRP can be used to implement a scheduling solutions and how monads enable composable task definitions. The cron clone, implemented using the Rhine library, can parse a cron file and schedule commands accordingly using FRP streams. The IL allows users to define scheduled tasks in a functional manner, allowing for time-based schedules and webhook capabilities. Due to using functional programming concepts such as monads, the IL provides a flexible and expressive way to define scheduling logic.
In summary, this work provides an initial exploration of functional and denotational approaches to scheduling systems and highlights several open questions that can guide future investigation and development.

The logo of Haskell
Darrin A Thompson; Jeffrey Wheeler

Example of scheduling a WebHook
Own presentment

```haskell
main :: IO ()
main = runIL [4000] $ do
  atEachEventOccurence
    (webHook 4000 (methodGet, "/action"))
    $ \req -> do
      let host = show $ remoteHost req
      printAction $
        "triggered from host: " ++ host
      return $
        responseLBS
          status200
          [("Content-Type", "text/plain")]
          "hello, webhook!\n"
```

Example of scheduling an automated back up in the EDSL
Own presentment

```haskell
import IL

main :: IO ()
main = runIL [] $ do
  backupSundayMidnight
  backupAfternoon

backupSundayMidnight :: Action ()
backupSundayMidnight = atEachEventOccurence (recurrence onSundayMidnight) fullBackup

backupAfternoon :: Action ()
backupAfternoon = atEachEventOccurence (recurrence onAfternoon) backupHome

fullBackup :: () -> Action ()
fullBackup a = backupHome a >~> backupConf

onSundayMidnight :: Schedule
onSundayMidnight = TimeSchedule (Single 0, Single 0, Asterisk, Asterisk, Single 0)

onAfternoon :: Schedule
onAfternoon = TimeSchedule (Single 0, Single 16, Asterisk, Asterisk, Asterisk)

backupConf :: () -> Action ()
backupConf _ = commandAction "/usr/bin/backup-conf.sh"

backupHome :: () -> Action ()
backupHome _ = commandAction "/usr/bin/backup-home.sh"

-- in cron:: 0 0 * * 0 /usr/bin/backup-home.sh && /usr/bin/backup-conf.sh
--           0 16 * * * /usr/bin/backup-home.sh
```


Advisor
Prof. Dr. Farhad D. Mehta

Subject Area
Software Engineering, System Software