Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie

WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI, INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ

KATEDRA INFORMATYKI STOSOWANEJ

Praca dyplomowa magisterska

Projekt języka programowania wspierającego przetwarzanie rozproszone na platformach heterogenicznych.

Design of a programming language with support for distributed computing on heterogenous platforms.

Autor: *Kajetan Rzepecki*
Kierunek studiów: *Informatyka*
Opiekun pracy: *dr inż. Piotr Matyasik*

Kraków, 2015

Oświadczam, świadomy odpowiedzialności karnej za poświadczenie nieprawdy, że niniejszą pracę dyplomową wykonałem osobiście i samodzielnie, i nie korzystałem ze źródeł innych niż wymienione w pracy.

Serdecznie dziękuję opiekunowi pracy za wsparcie merytoryczne oraz dobre rady edytorskie pomocne w tworzeniu pracy.

Spis treści

1. Wstęp

Celem pracy jest zdążenie na czas [1].

1.1. Motywacja pracy

– Heterogeneity being incresingly important

– add heterogeneity clarification diagram

– Platform Independence being insufficient

– Platform Awareness being the key

– This language is supposed to solve heterogeneity using Platform Awareness.

– hint at previous work in this field (mah beng)

1.2. Zawartość pracy

– list what is found where in the thesis

2. Język FOOF

– elegantly solving heterogenuity issues

– remain simple & highly ortogonal

– embody Spartan Programming principles

2.1. Ideologia języka

– simplicity but not crudeness

– pragmatism

– platform awareness

– orthogonal features

2.2. Proste typy danych

– describe numbers

– describe symbols

– describe strings

2.3. Złożone typy danych

– describe lists

– describe vectors ?

– describe maps ?

2.4. Funkcje

– a note about lambda calculus

– add a code fragment implementing booleans in lambda calculus ?

– describe lambdas

– describe named lambdas aka defines

2.5. Kontynuacje

– describe the notion of a continuation

– describe CPS transformation and comment code equivalence

– add a code example of the CPS transform

– hint at greater detail in a future section

– hint at delimited control

– describe exceptions via continuations

2.6. Przetwarzanie współbieżne i rozproszone

– describe processes via continuations

– describe actor model primitives

– comment on adding distribution

2.7. Reprezentacja wiedzy w języku

– describe various ways of knowledge representation

– describe use cases in the language

– hint at using an RBS

2.8. Makra

– describe macros

– add some code examples of let -> lambda etc

– hint at problems of hygiene & add code example ?

– hint at problems of macros & modules coexisting

2.9. System modułowy

– describe structures - namespaces for definitions

– describe modules - parameterized structures

– describe units - runnable modules

– describe protocols - a set of capabilities of a module

– hint at protocols & SOA connection

3. Kompilator języka FOOF

– mention technology selection & limitations (large project, little time)

– mention possible bootstrapping

– briefly touch on the architecture

– hint at using Scheme for the boring details (datatypes etc)

3.1. Architektura kompilatora

– compiler block diagram

– list compilation phases

– list which phases have been actually implemented

– list which phases have been skipped and say why (optimization, code-gen, parsin)

3.2. Parser

– briefly describe how Scheme praser works and what it produces

– hint at a possibility of replacing this with a PEG-based packrat

3.3. Makro-ekspansja

– describe macroexpantion phase

– describe why macroexpansion is hardcoded

– list available macros

– show some examples of macro-expanded code

3.4. Obsługa Systemu Modułowego

– describe how modules are handled right now

– show some examples of macro-expanded structures & modules

– maby combine this with the previous section ?

– maby hint at special module access syntax (foo.bar.baz)

3.5. Transformacja *Continuation Passing Style*

– describe in detail why this is useful

– describe in detail how to transform simple stuff

– describe in detail how to transform functions

– describe in detail how to handle exceptions

– hint at emitting calls to primitive functions `&yield-cont`, `&uproc-error-handler` etc

3.6. Generacja kodu

– describe how a subset of both Scheme and FOOF is emitted

– describe how Scheme is used for direct code execution

– hint at further development using LLVM

– mention a requirement to perform closure conversion

– add a code example of closure conversion

4. System uruchomieniowy języka

– briefly touch on the architecture

– mention Scheme bootstrap

4.1. Architektura systemu uruchomieniowego

– block diagram of the system including the RBS

– describe various parts

– hint at in-depth description of RBS implementation in a future section

4.2. Implementacja podstawowych typów danych

– describe scheme bootstrap

– describe equivalence of various constructs such as lambdas

4.3. Implementacja kontynuacji

– describe how continuations are handled without getting into CFS (returning cont + hole)

– add a code example with step-by-step execution

4.4. Implementacja procesów

– add a diagram of the uProc context - only include status, cont & handler registers

– describe uProc context registers

– describe how continuations with returns play into this scheme (recall `&yield-cont`)

– describe how error handling is implemented (recall `&uproc-error-handler` etc)

4.5. Harmonogramowanie procesów

- uProc context diagram - add priority & rtime
- describe the Completely Fair Scheduler
- add pseudocode listing showing the algorithm
- describe uProc context switching
- mention previous implementation in beng
- contrast current impl with previous one (lack of wait list, heaps instead of RBT)

4.6. Implementacja Modelu Aktorowego

- describe actor model briefly
- uProc context diagram - add pid & msgqueue
- describe modifications to the runtime required by actor model (**current-uproc**, uproc list, context fields)
- describe implementation of various actor model primitives
- add some code examples and discussion of its effects and what happens

4.7. Dystrybucja obliczeń

- difference between concurrency & distribution
- describe modifications to the runtime in order to support distribution
- hint about using a simple protocol
- hint about moving this into stdlib

5. Reprezentacja i przetwarzanie wiedzy

– describe how this needs a separate section

– elaborate on different ways of knowledge representation

5.1. Reprezentacja wiedzy w języku

– describe facts - signalling, assertion & retraction

– describe rules briefly - adding & disabling, triggering

5.2. Algorytm Rete

– describe in detail the algorithm

– describe network merging

– add a diagram showing network merging

– describe briefly its history

– Rete vs naïve approach

– add a diagram showing how it is better

5.3. Implementacja Rete - wnioskowanie w przód

– describe what forward-chaining is

– describe naïve Rete - no network merging

– hint that this might be a good thing (future section)

– describe all the nodes

5.4. Implementacja wnioskowania wstecz

– describe what backward-chaining is

– describe fact store in detail - linear, in-memory database

– querying fact store = create a rule and apply all known facts to it

5.5. Integracja z Systemem Uruchomieniowym

– describe how it sucks right now

– describe possible integration with the module system

– describe possible representation of rules by autonomus processes

– add a diagram of concurrent rules

6. Podsumowanie

– reiterate the goal of the thesis

– state how well has it been achieved

6.1. Kompilator języka FOOF

– needs better optimizations

– needs better error handling

6.2. System uruchomieniowy

– needs more stuff

– needs macroexpansion

– needs to drop RBS and move it into stdlib

6.3. Przyszłe kierunki rozwoju

– more datatypes

– native compilation via LLVM

– bootstrapping compiler

– librarized RBS

– librarized distribution with data encryption & ACLs

– data-level paralellism

Bibliografia

[1] J. Backus, “Can programming be liberated from the von neumann style?: A functional style and its algebra of programs,” *Commun. ACM*, vol. 21, pp. 613–641, Aug. 1978.

A. Gramatyka języka FOOF

– concrete language grammar in PEG or BNF

B. Przykładowe programy

- some basic definitions & operations
- fibonacci
- parallell fibonacci
- module system - logger
- error handling - (raise (raise ”fight the powa”))
- RBS forward-chaining
- RBS backward-chaining

C. Spis wbudowanych funkcji języka FOOF

– list contents of bootstrap.scm

– describe what `&make-structure`, `&yield-cont` etc do

D. Spisy rysunków i fragmentów kodu

Spis rysunków

Spis listingów