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

– Knowledge for Service Oriented Architecture

– Platform Awareness

1.2. Zawartość pracy

2. Język FOOF

– language grammar

– language philosophy

2.1. Proste typy danych

– atoms

2.2. Złożone typy danych

– lists

– vectors

– maps

2.3. Funkcje i kontynuacje

– about lambdas

– about continuations

– exceptions via continuations

2.4. System modułowy

– protocols

– modules

– units

– SOA

2.5. Przetwarzanie współbieżne i rozproszone

– actor model

– distribution

2.6. Reprezentacja wiedzy w języku

– unconstrained knowledge about the system

– RBS vs Prolog vs ontologies

– appendix óntologies”

3. Kompilator języka FOOF

– compiler block diagram

– list compilation phases

– mention bootstrapping

3.1. Implementacja typów danych

– 2 bit tags for simple objects

– tag word for complex objects

3.2. Implementacja systemu modułowego

– simple transform

3.3. Analiza leksykalna i syntaktyczna

– PEG

– macroexpander

3.4. Analiza semantyczna

– CPS

– delimited continuations

– error handling

3.5. Dalsze transformacje i optymaliza kodu

– closure conversion

– lambda lifting

– partial evaluation?

– constant folding?

3.6. Generacja kodu wynikowego

– might need appendix ŁLVM”

– name mangling

– low-level optimizations

– source level translation

3.7. Implementacja systemu uruchomieniowego języka

– primops

– alloc

– gc

– scheduler

– network stack

4. System uruchomieniowy języka

– block diagram of the system

– might need appendix ”tracing vs ref count”

– might need to throw this away

4.1. Architektura systemu uruchomieniowego

– built-in units

– process tree

4.2. Ładowanie i uruchamianie programów

– loading modules

– starting units

– mention unit redundancy

– list primops

4.3. Alokator pamięci

– $\mathrm{gc}_{\mathrm{malloc}}$

– allocation strategy

– garbage collection

– garbage cycles handling

– etc

– list primops

4.4. Harmonogramowanie procesów

– cue next section

– list primops

4.5. Komunikacja między węzłami

– cue next section

– list primops

4.6. Baza wiedzy

– cue next section

– list primops

5. Przetwarzanie współbieżne i rozproszone

– difference between concurrency & distribution

5.1. Reprezentacja procesów

– process object

5.2. Harmonogramowanie procesów

– CFS algorithm

5.3. Implementacja symetrycznego multiprocessingu

– Completele Fair Scheduler

– cue CPS

5.4. Implementacja Modelu Aktorowego

– message passing

– message queue

– copying vs passing pointers

5.5. Protokół komunikacji międzywęzłowej

– node & process IDs

– packets

5.6. Implementacja protokołu komunikacji

– TCP vs UDP

5.7. Listy kontroli dostępu

– safety measures

– might need to be thrown away

6. Reprezentacja i przetwarzanie wiedzy

– might need appendix Żete usage examples”

6.1. Reprezentacja wiedzy w języku

– facts

– rules

6.2. Algorytm Rete

– about Rete

– Rete vs naïve approach

6.3. Implementacja Rete - forward chaining

– naïve Rete

– all the nodes!

6.4. Implementacja backward-chaining

– fact store - linear, in-memory database

6.5. Integracja przetwarzania wiedzy w języku

– integration with the module system

– rules represented by autonomus processes

6.6. Przekazywanie wiedzy pomiędzy węzłami

– network protocols

– ACLs

7. Podsumowanie

7.1. Kompilator języka FOOF

– needs better optimizations

– needs better error handling

7.2. System uruchomieniowy

– needs more stuff

– needs macroexpansion

7.3. Przyszłe kierunki rozwoju

– type system

– datatypes

– runtime

– data-level paralellism

– data encryption & ACLs

– DHT (might be a good example program?)

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. Spisy rysunków i fragmentów kodu

Spis rysunków

Spis listingów