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

– describe the goal of the thesis - creating a programming language that:

– elegantly solving heterogenuity issues

– remains simple & highly ortogonal [[1]]

– embodies Spartan Programming principles

1.1. Motywacja pracy

– name and describe challenges of distributed systems - `http://lycog.com/distributed-systems/challenges-distributed-systems/` [[2]] [[3]]

– Heterogeneity being incresingly important with rise of technologies such as IoT [[4]]

– add heterogeneity clarification diagram

– Platform Independence being insufficient and/or impossible (vast number of very different devices)

– Platform Awareness being the key (embracing the diversity)

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

1.2. Zawartość pracy

– list what is found where in the thesis

2. Język FOOF

- simplicity but not crudeness [[1]]
- pragmatism [[5]]
- platform awareness
- orthogonal features [[1]]
- contrast with Scheme/Lisp (and SML ?) [[6]]

2.1. Podstawowe typy danych

- describe lists - pairs of atoms|lists [[7]]
- describe numbers
- describe symbols
- describe strings
- describe vectors ?
- describe maps ?

2.2. Funkcje

- a note about lambda calculus [[8]] [[9]]
- add a code fragment implementing booleans in lambda calculus ?
- describe lambdas
- describe named lambdas aka defines

2.3. Kontynuacje

– describe the notion of a continuation [[10]]

– briefly describe CPS transformation and comment code equivalence [[11]]

– add a code example of the CPS transform

– hint at greater detail in a future section

– hint at delimited control [[12]]

– describe exceptions via continuations

2.4. Przetwarzanie współbieżne i rozproszone

– describe Actor Model [[13]] [[14]]

– describe processes via continuations

– describe actor model primitives [[13]]

– comment on adding distribution

2.5. Reprezentacja wiedzy w języku

– describe use cases in the language

– describe various ways of knowledge representation [[15]] [[16]] [[17]]

– describe primitive operations

– hint at using an RBS

2.6. Makra

– describe macros

– add some code examples of available macros

– hint at problems of hygiene & add code example [[18]] [[19]]

– hint at problems of macros & modules coexisting [[20]]

– contrast macros with other techniques (fexprs) [[21]]

2.7. System modułowy

- describe the need for a module system [[20]]
- describe structures - namespaces for definitions
- describe modules - parameterized structures [[22]]
- 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) [[23]]

– mention possible bootstrapping

– briefly touch on the architecture [[24]]

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

3.1. Architektura kompilatora

– compiler block diagram

– list compilation phases [[24]] [[23]] [[11]]

– 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 [[6]] [[25]]

– hint at a possibility of replacing this with a PEG-based packrat [[26]] [[27]]

– note about special quasiquote syntax [[28]]

3.3. Makro-ekspansja

– describe macroexpantion phase

– describe why macroexpansion is hardcoded [[20]]

– list available macros

– show some examples of macro-expanded code

3.4. Obsługa Systemu Modułowego

– describe how modules are handled right now [[20]] [[22]]

– 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 whit CPS is [[11]] [[29]]

– describe in detail how to transform simple stuff

– describe in detail how to transform functions (recursion problems)

– describe in detail how to handle exceptions

– describe in detail why this is useful (partial evaluation, constant folding etc) [[30]]

– 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 (contrast with Core Erlang) [[31]] [[32]]

– describe how Scheme is used for direct code execution

– hint at further development using LLVM [[**?**]]

– mention a requirement to perform closure conversion or lambda lifting [[33]]

– add a code example contrasting closure conversion and lambda lifting

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 [[6]]

– describe equivalence of various constructs such as lambdas

4.3. Implementacja kontynuacji

– describe how continuations are handled without getting into CFS (returning cont + hole, contrast to how G-machine/TIM reductions work) [[11]] [[33]]

– 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`)

– contrast continuations with corutines and yielding [[34]]

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

– contrast with erlang [[35]]

4.5. Harmonogramowanie procesów

– uProc context diagram - add priority & rtime

– describe the Completely Fair Scheduler [[36]]

– add pseudocode listing showing the algorithm

– describe uProc context switching

– contrast current impl with previous one (lack of wait list, heaps instead of RBT) [[37]]

– contrast with erlang [[35]]

4.6. Implementacja Modelu Aktorowego

– describe actor model briefly [[13]] [[14]]

– 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

– contrast with erlang [[35]]

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 [[17]] [[38]] [[15]] [[**?**]] [[**?**]]

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 [[39]]

– add a diagram showing network merging

– describe briefly its history [[40]]

– Rete vs naïve approach (vs CLIPS or similar ?)

– add a diagram showing how it is better

– contrast it with other algorithms [[41]]

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 [[39]]

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 (a lot)

– describe possible integration with the module system (fact inference)

– describe possible representation of rules by autonomus processes [[42]]

– add a diagram of concurrent rules

– hint at movig the implementation to the stdlib

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.

[2] P. E. McKenney, "Is parallel programming hard, and, if so, what can you do about it?." Free online version.

[3] A. S. Tanenbaum and M. v. Steen, *Distributed Systems: Principles and Paradigms (2Nd Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2006.

[4] J. Höller, V. Tsiatsis, C. Mulligan, S. Karnouskos, S. Avesand, and D. Boyle, *From Machine-to-Machine to the Internet of Things: Introduction to a New Age of Intelligence*. Elsevier, Apr. 2014.

[5] C. A. R. Hoare, "Hints on programming language design.," tech. rep., Stanford, CA, USA, 1973.

[6] M. Sperber, R. K. Dybvig, M. Flatt, A. van Straaten, R. Findler, and J. Matthews, *Revised [6] Report on the Algorithmic Language Scheme*. New York, NY, USA: Cambridge University Press, 1st ed., 2010.

[7] J. McCarthy, "Recursive functions of symbolic expressions and their computation by machine, part i," *Commun. ACM*, vol. 3, pp. 184–195, Apr. 1960.

[8] A. Church, "A set of postulates for the foundation of logic part i," *Annals of Mathematics*, vol. 33, no. 2, pp. 346–366, 1932. http://www.jstor.org/stable/1968702Electronic Edition.

[9] A. Church, "A set of postulates for the foundation of logic part ii," *Annals of Mathematics*, vol. 34, no. 2, pp. 839–864, 1933.

[10] J. C. Reynolds, "The discoveries of continuations," *Lisp Symb. Comput.*, vol. 6, pp. 233–248, Nov. 1993.

[11] A. W. Appel, *Compiling with Continuations*. Cambridge University Press, 1992.

[12] R. K. Dybvig, S. P. Jones, and A. Sabry, "A monadic framework for delimited continuations," tech. rep., IN PROC, 2005.

[13] C. Hewitt, P. Bishop, and R. Steiger, "A universal modular actor formalism for artificial intelligence," in *Proceedings of the 3rd International Joint Conference on Artificial Intelligence*, IJCAI'73, (San Francisco, CA, USA), pp. 235–245, Morgan Kaufmann Publishers Inc., 1973.

[14] W. D. Clinger, "Foundations of actor semantics," tech. rep., Cambridge, MA, USA, 1981.

[15] S. Hachem, T. Teixeira, and V. Issarny, "Ontologies for the internet of things," in *Proceedings of the 8th Middleware Doctoral Symposium*, MDS '11, (New York, NY, USA), pp. 3:1–3:6, ACM, 2011.

[16] H. Samimi, C. Deaton, Y. Ohshima, A. Warth, and T. Millstein, "Call by meaning," in *Proceedings of the 2014 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software*, Onward! 2014, (New York, NY, USA), pp. 11–28, ACM, 2014.

[17] D. S. C. G. Wang, W and K. Moessner, "Knowledge representation in the internet of things: Semantic modelling and its application," *Wang, W, De, S, Cassar, G and Moessner, K*, vol. 54, pp. 388 – 400.

[18] A. Bawden, "First-class macros have types," in *In 27th ACM Symposium on Principles of Programming Languages (POPL'00*, pp. 133–141, ACM, 2000.

[19] C. Queinnec, "Macroexpansion reflective tower," in *Proceedings of the Reflection'96 Conference*, pp. 93–104, 1996.

[20] J. M. Gasbichler, *Fully-parameterized, first-class modules with hygienic macros.* PhD thesis, Eberhard Karls University of Tübingen, 2006. http://d-nb.info/980855152.

[21] J. N. Shutt, *Fexprs as the basis of Lisp function application or $vau: the ultimate abstraction.* PhD thesis, Worcester Polytechnic Institute, August 2010.

[22] A. Rossberg, "1ML - core and modules united," 2015.

[23] A. Ghuloum, "An Incremental Approach to Compiler Construction," in *Scheme and Functional Programming 2006*.

[24] A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman, *Compilers: Principles, Techniques, and Tools (2Nd Edition)*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2006.

[25] H. Abelson and G. J. Sussman, *Structure and Interpretation of Computer Programs*. Cambridge, MA, USA: MIT Press, 2nd ed., 1996.

[26] G. Hutton and E. Meijer, "Monadic parser combinators," 1996.

[27] B. Ford, "Parsing expression grammars: A recognition-based syntactic foundation," in *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '04, (New York, NY, USA), pp. 111–122, ACM, 2004.

[28] A. Bawden, "Quasiquotation in lisp," tech. rep., University of Aarhus, 1999.

[29] A. Kennedy, "Compiling with continuations, continued," in *Proceedings of the 12th ACM SIGPLAN International Conference on Functional Programming*, ICFP '07, (New York, NY, USA), pp. 177–190, ACM, 2007.

[30] D. Bacon, "A Hacker's Introduction to Partial Evaluation," 2002.

[31] R. Carlsson, "An introduction to Core Erlang," in *In Proceedings of the PLI'01 Erlang Workshop*, 2001.

[32] R. Carlsson, B. Gustavsson, E. Johansson, T. Lindgren, S.-O. Nyström, M. Pettersson, and R. Virding, "Core Erlang 1.0.3 language specification," tech. rep., Department of Information Technology, Uppsala University, Nov. 2004.

[33] S. P. Jones and D. Lester, *Implementing functional languages: a tutorial*. Prentice Hall, 1992. Free online version.

[34] A. L. D. Moura and R. Ierusalimschy, "Revisiting coroutines," *ACM Trans. Program. Lang. Syst.*, vol. 31, pp. 6:1–6:31, Feb. 2009.

[35] J. Armstrong, R. Virding, C. Wikström, and M. Williams, *Concurrent Programming in ERLANG (2Nd Ed.)*. Hertfordshire, UK, UK: Prentice Hall International (UK) Ltd., 1996.

[36] C. S. Pabla, "Completely fair scheduler," *Linux J.*, vol. 2009, Aug. 2009.

[37] R. Sedgewick, "Left-leaning red-black trees," 2008.

[38] P. Barnaghi, W. Wang, C. Henson, and K. Taylor, "Semantics for the internet of things: Early progress and back to the future," *Int. J. Semant. Web Inf. Syst.*, vol. 8, pp. 1–21, Jan. 2012.

[39] C. L. Forgy, "Rete: A fast algorithm for the many pattern/many object pattern match problem," *Artificial Intelligence*, vol. 19, no. 1, pp. 17 – 37, 1982.

[40] C. L. Forgy, *On the Efficient Implementation of Production Systems.* PhD thesis, Pittsburgh, PA, USA, 1979. AAI7919143.

[41] D. P. Miranker, *TREAT: A New and Efficient Match Algorithm for AI Production Systems.* PhD thesis, New York, NY, USA, 1987. UMI Order No. GAX87-10209.

[42] A. Gupta, C. Forgy, A. Newell, and R. Wedig, "Parallel algorithms and architectures for rule-based systems," *SIGARCH Comput. Archit. News*, vol. 14, pp. 28–37, May 1986.

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