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

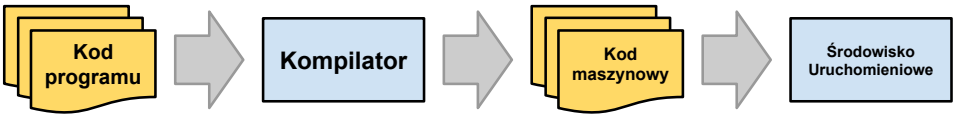
– describe the goal of the thesis - designing a programming language that:

  – elegantly solving heterogenuity issues

  – remains simple & highly ortogonal [[1]]

  – embodies Spartan Programming principles

– implementing its compiler



Rysunek 1.1: Schemat interakcji poszczególnych elementów języka.

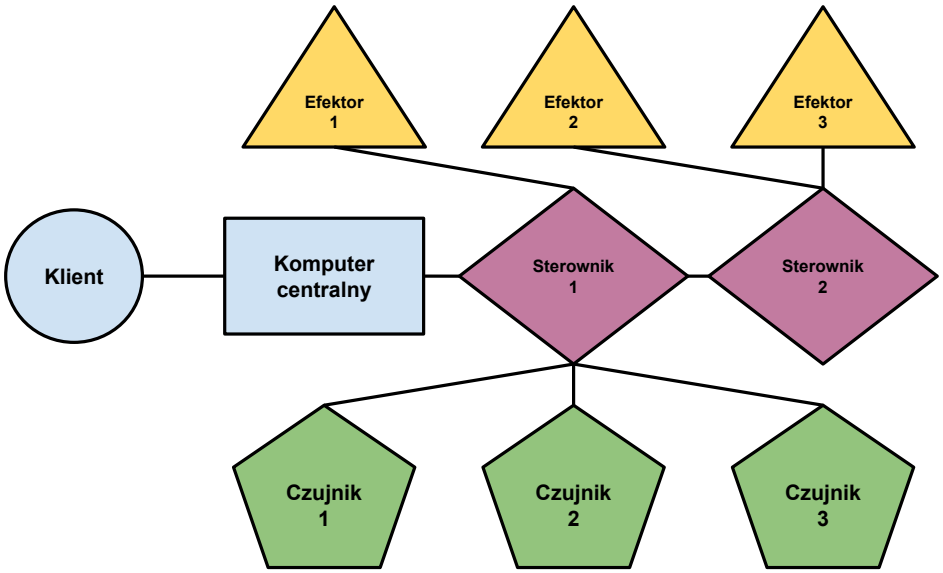
– implementing its runtime system

1.1. Motywacja pracy

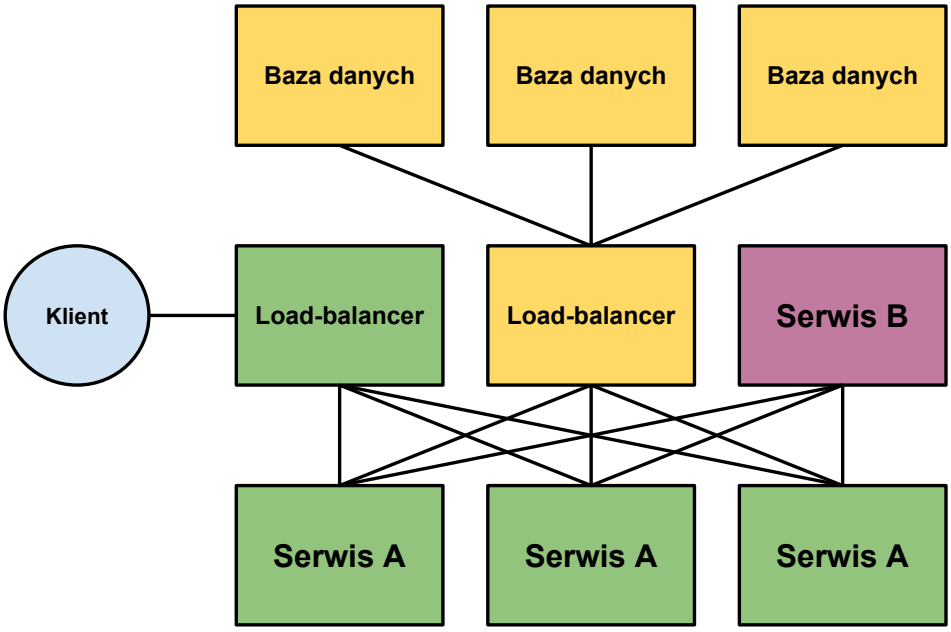
– name and describe challenges of distributed systems - `http://lycog.com/distributed-systems/challenges-distributed-systems/` [[2]] [[3]]

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

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



Rysunek 1.2: Przykład systemu opartego o heterogeniczną platformę sprzętową.



Rysunek 1.3: Przykład systemu heterogenicznego niezależnie od platformy sprzętowej.

– 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
- mention funarg problem [[10]]
- mention recursion problem [[11]] [[12]]
- describe named lambdas aka defines

2.3. Kontynuacje

– describe the notion of a continuation [[13]]

– briefly describe CPS transformation and comment on code equivalence [[14]]

– add a code example of the CPS transform

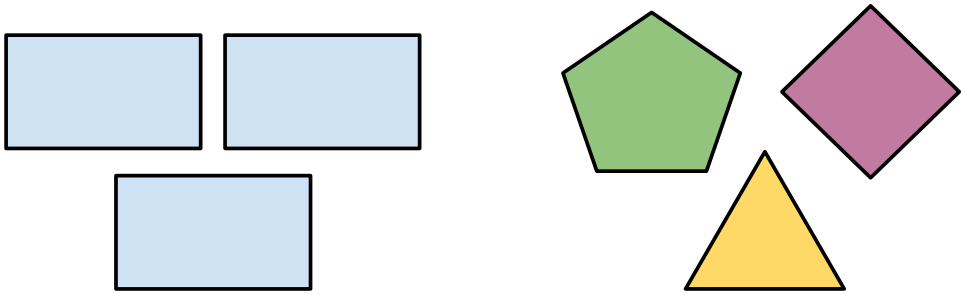
– hint at greater detail in a future section

– hint at delimited control [[15]]

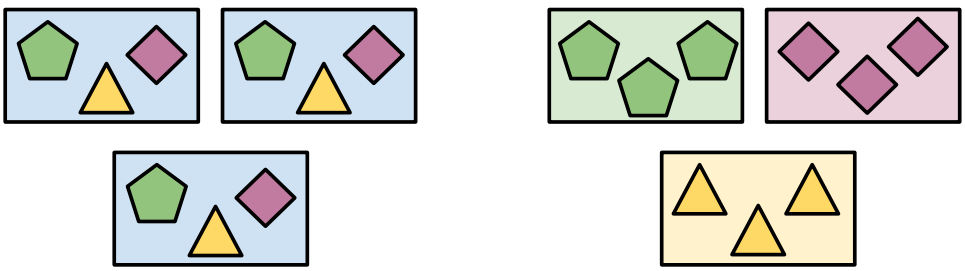
– describe exceptions via continuations

2.4. Przetwarzanie współbieżne i rozproszone

– briefly describe AMP vs SMP and contrast it with platform heterogeneity



Rysunek 2.1: Podstawowe różnice pomiędzy platformami homogenicznymi oraz heterogenicznymi.



Rysunek 2.2: Podstawowe różnice pomiędzy systemami asymetrycznymi i symetrycznymi.

– note that system doesn't need to run on a heterogenous platform to be heterogenous itself

– describe Actor Model [[16]] [[17]]

– describe processes via continuations (trampolines)

– describe actor model primitives [[16]]

– comment on adding distribution

2.5. Reprezentacja wiedzy w języku

– describe use cases in the language

– describe various ways of knowledge representation [[18]] [[19]] [[20]]

– 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 [[21]] [[22]]

– hint at problems of macros & modules coexisting [[23]]

– contrast macros with other techniques (fexprs) [[24]]

2.7. System modułowy

– describe the need for a module system [[23]]

– describe structures - namespaces for definitions

– describe modules - parameterized structures [[25]]

– describe units - runnable modules

– describe protocols - a set of capabilities of a module

– hint at protocols & SOA connection ?

3. Kompilator języka FOOF

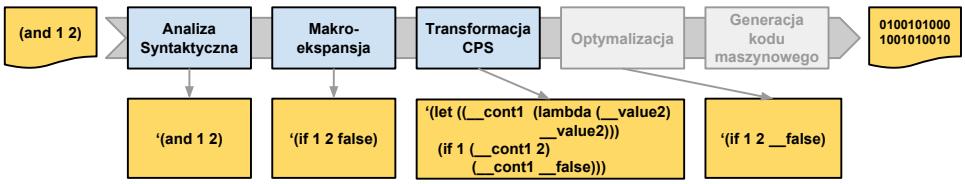
– mention technology selection & limitations (large project, little time) [[26]]

– mention possible bootstrapping

– briefly touch on the architecture [[27]]

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

3.1. Architektura kompilatora



Rysunek 3.1: Schemat poszczególnych faz kompilacji i przykładowych danych będących wynikiem ich działania.

– list compilation phases [[27]] [[26]] [[14]]

– list which phases have been actually implemented

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

3.2. Parser

– briefly describe how Scheme praser works and what it produces [[6]] [[10]]

– hint at a possibility of replacing this with a PEG-based packrat [[28]] [[29]]

– note about special quasiquote syntax [[30]]

3.3. Makro-ekspansja

– describe macroexpantion phase

– describe why macroexpansion is hardcoded [[23]]

– list available macros

– show some examples of macro-expanded code

3.4. Obsługa Systemu Modułowego

– describe how modules are handled right now [[23]] [[25]]

– 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 what CPS is [[14]] [[31]]

– describe in detail how to transform simple stuff

– describe in detail how to transform functions (recursion problems & crude solution via mutation [[32]], [[33]], [[11]])

– describe in detail how to handle exceptions

– describe in detail why this is useful (partial evaluation, constant folding etc) [[34]]

– 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) [[35]] [[36]]

– 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 [[37]]

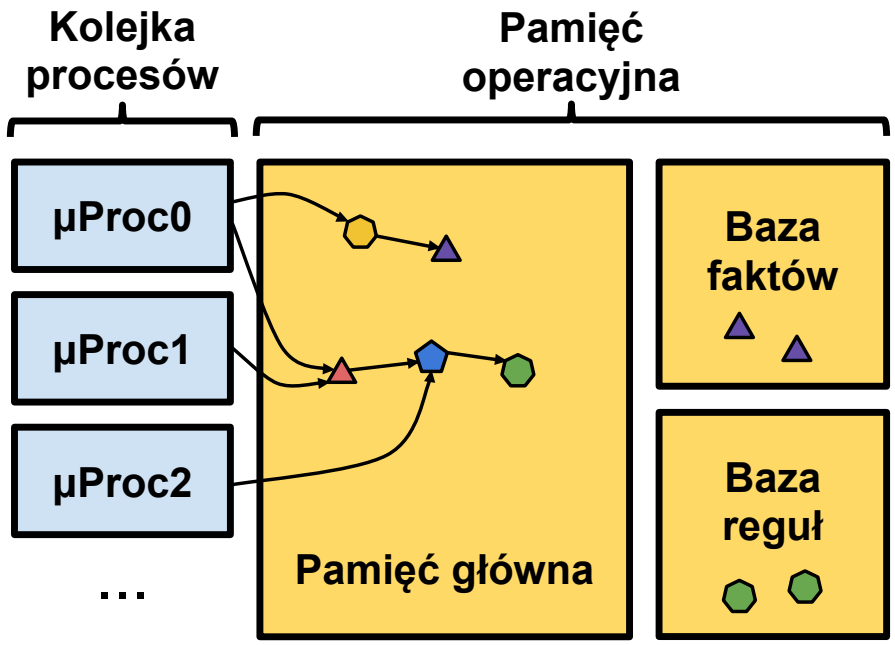
– add a code example contrasting closure conversion and lambda lifting

4. Środowisko uruchomieniowe języka

– briefly touch on the architecture

– mention Scheme bootstrap

4.1. Architektura środowiska uruchomieniowego



Rysunek 4.1: Schemat architektury środowiska uruchomieniowego języka FOOF.

– describe various parts

– mention that this is single threaded and requires forking for real concurrency

– 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

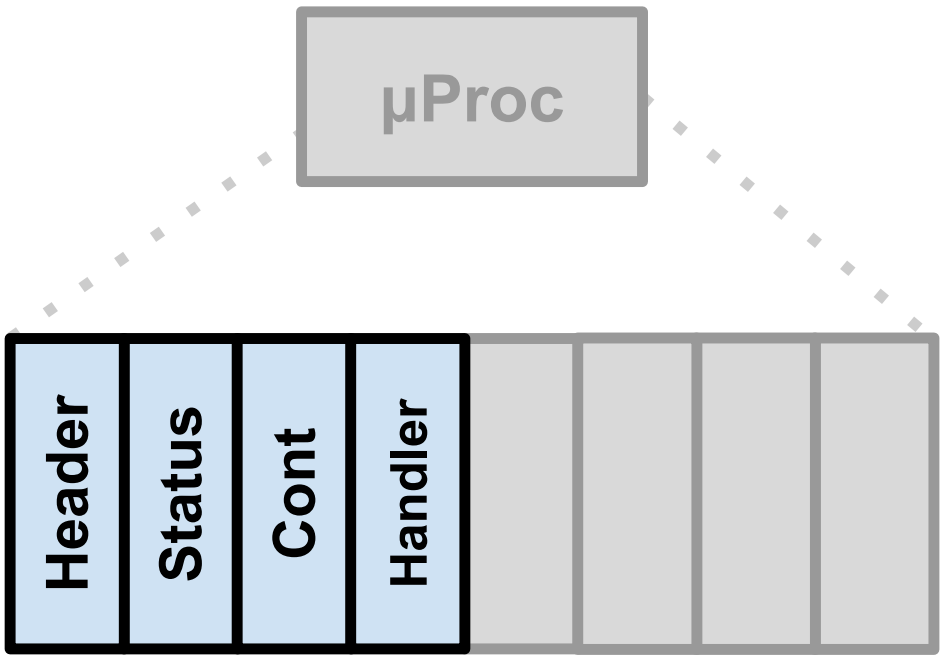
– describe how continuations are handled without getting into CFS (returning cont + hole aka trampoline, contrast to how G-machine/TIM reductions work) [[14]] [[37]]

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

– hint at debugging potential using step by step continuation execution with debug info inbetween

4.4. Implementacja procesów

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



Rysunek 4.2: Schemat kontekstu procesu obrazujący rejestry niezbędne do jego działania.

– describe uProc context registers

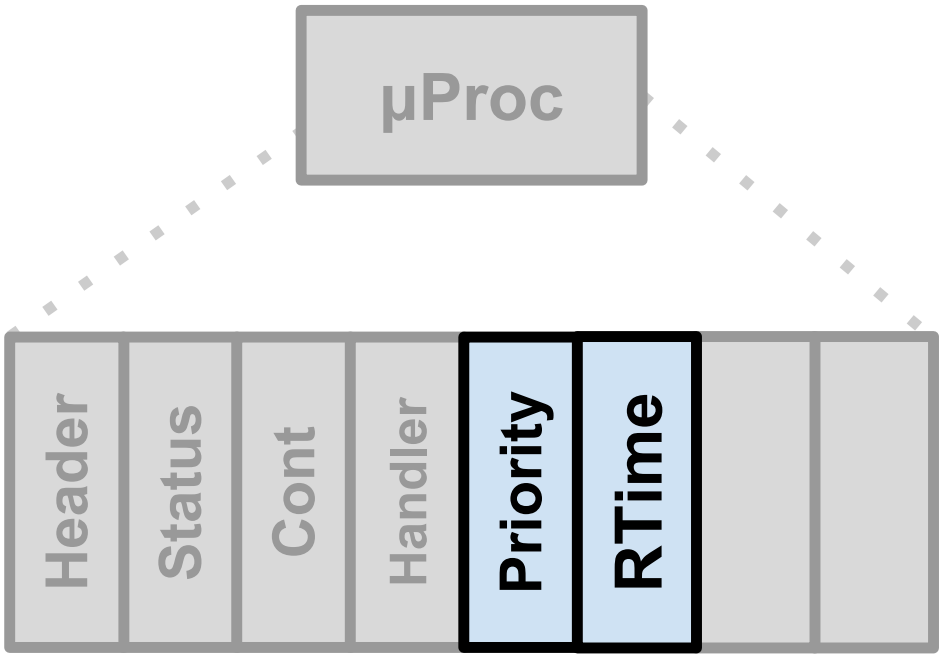
– describe how trampolines play into this scheme (recall `&yield-cont`)

– contrast trampolines with corutines (more suitable in CPS) and yielding (done implicitly) [[38]]

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

– contrast with erlang [[39]]

4.5. Harmonogramowanie procesów



Rysunek 4.3: Dodatkowe rejestry kontekstu mikroprocesu wymagade do implementacji algorytmu *Completely Fair Scheduler*.

– describe the Completely Fair Scheduler [[40]]

– add pseudocode listing showing the algorithm

– describe uProc context switching

– contrast current impl with previous one (lack of wait list - notifications, heaps instead of RBT, number of reductions instead of time) [[41]]

– contrast with erlang [[39]]

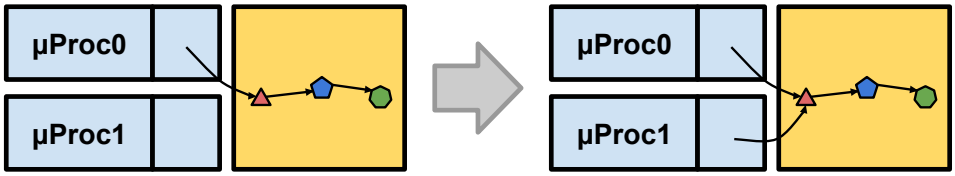
4.6. Implementacja Modelu Aktorowego

– describe actor model briefly [[16]] [[17]]

Rysunek 4.4: Dodatkowe rejestry kontekstu mikroprocesu wymagade do implementacji Modelu Aktorowego.

– describe modifications to the runtime required by actor model (**current-uproc**, uproc list, context fields)

– describe implementation of various actor model primitives



Rysunek 4.5: Diagram obrazujący efekty przekazywania wiadomości pomiędzy mikroprocesami.

– add some code examples and discussion of its effects and what happens

– contrast with erlang [[39]]

4.7. Dystrybucja obliczeń

– difference between concurrency & distribution

– describe modifications to the runtime in order to support distribution

– hint about using a simple protocol

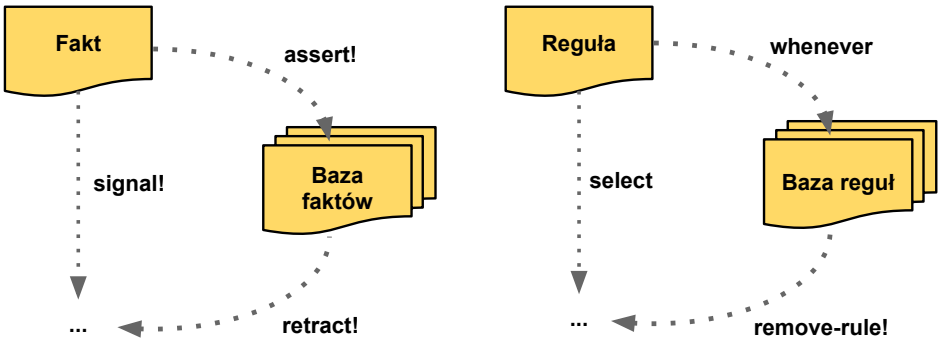
– hint about moving this into stdlib

5. Reprezentacja i przetwarzanie wiedzy

– describe how this needs a separate section

– elaborate on different ways of knowledge representation [[20]] [[42]] [[18]] [[?]] [[?]]

5.1. Reprezentacja wiedzy w języku



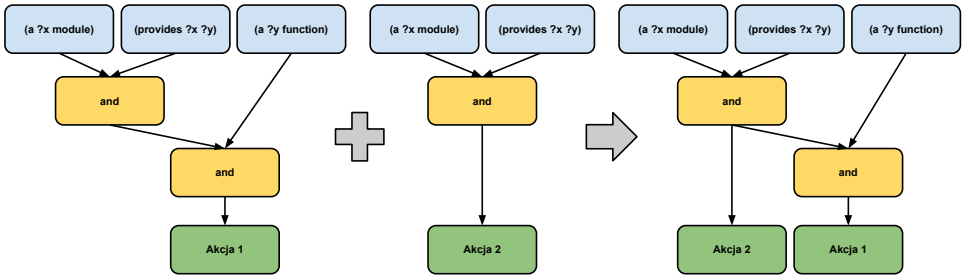
Rysunek 5.1: Schemat działania wbudowanych baz faktów i reguł.

– describe facts - signalling, assertion & retraction

– describe rules briefly - adding & disabling, triggering

5.2. Algorytm Rete

– describe in detail the algorithm [[43]]



Rysunek 5.2: Schemat łączenia podsieci w algorytmie *Rete*.

– describe briefly its history [[44]]

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

– add a benchmark diagram showing how Rete is better

– contrast it with other algorithms [[45]]

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 [[43]]

5.4. Implementacja wnioskowania wstecz

– describe what backward-chaining is

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

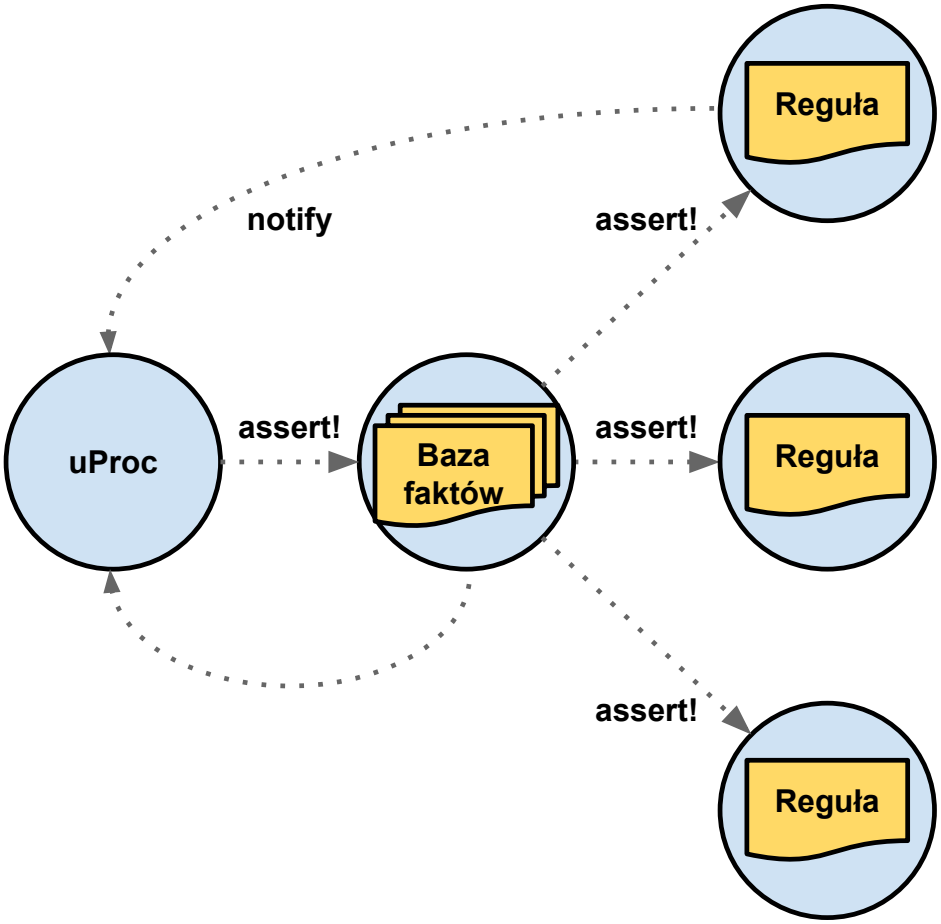
– querying fact store = create a rule and apply all known facts to it

5.5. Integracja z Systemem Uruchomieniowym

– describe how it sucks right now (notify-whenever instead of generic whenever, logic rule removal)

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

– describe possible representation of rules by autonomus processes [[46]]



Rysunek 5.3: Schemat działania rozproszonej wersji algorytmu *Rete*.

– 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. Środowisko uruchomieniowe

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

[11] M. Felleisen and D. P. Friedman, "(Y Y) works!," 1991.

[12] M. Goldberg, "On the recursive enumerability of fixed-point combinators," 2005.

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

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

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

[16] 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.

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

[18] 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.

[19] 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.

[20] 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.

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

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

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

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

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

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

[27] 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.

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

[29] 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.

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

[31] 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.

[32] D.-A. German, "Recursion without circularity or using let to define letrec," 1995.

[33] O. Kaser, C. R. Ramakrishnan, and S. Pawagi, "On the conversion of indirect to direct recursion," vol. 2, pp. 151–164, Mar. 1993.

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

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

[36] 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.

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

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

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

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

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

[42] 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.

[43] 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.

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

[45] 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.

[46] 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

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

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