UNIVERSITATEA ,,BABEŞ-BOLYAI", CLUJ-NAPOCA
FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ

UNIVERSITATEA ,,EÖTVÖS LORÁNT", BUDAPEST
FACULTATEA DE INFORMATICĂ

Modelarea Funcţională a Sistemelor de Operare

Rezumatul tezei de doctorat

Doctorand
PÁLI Gábor János

Conducătorii ştiinţific
Horia F. POP (BBU)
KOZSIK Tamás (ELTE)

Cluj-Napoca, martie 2012

Cuprins

Cuprinsul tezei

Cuvinte cheie

Cuvinte cheie: funcţional de programare, sistemelor integrate, sisteme de operare, dataflow reţele, programarea, declarativ de programare, limba încorporarea, Haskell, specifice domeniului limba, sisteme de programare, reactivaă de programare executabile semantică, semantică pure, generare a codului, compilator, firul comun, muncî de paralelism, tacită de programare

Introducere

Calculatoare sunt parte a vieţii noastre actuale şi fiecare dintre ele se face din două componente principale: hardware şi software. Separarea dintre acestea componentelor este rezultatul unei evoluţii în cazul în care inginerii au determinat că este mult mai plină de satisfacţii pentru fabricarea de dispozitive hardware, cu un relativ scăzut nivel de interfaţă şi apoi a pus straturi suplimentare virtuale pe partea de sus a sa format prin programele derulate pe ea. Ca un rezultat al structura standardizată de calculatoare de astăzi, este greu de a găsi o maşină fără cel puţin un strat de software-ul asociat, de obicei menţionată ca fiind un sistem de operare. Suntem înconjuraţi de o multitudine de dispozitive încorporate în formă de smartphone-uri, routere, Calculatoare portabile personale, seturi de multimedia, etc, care sunt încă rularea unor programe de sine.

In zilele noastre un obstacol serios în răspândirea unui sistem de operare este lipsa sprijinul pentru acesta. Fără a hardware-ului adecvat sprijini un sistem de operare este practic sortită eşecului. Cu toate acestea, există anumite segmente ale piaţă, în cazul în care companiile şi ingineri de software sunt hrănite cu C şi derivaţii săi, şi sunt în căutarea ceva care promite mai mult automatizare în dezvoltarea de sisteme low-level. Evoluţia rapidă a hardware-ul, de asemenea, forţează programatorii să se gândească la un nivel superior, în cazul în care compilatorul asociat devine partenerul lor. Un astfel de câmp tipic de de aplicare a acestei metodologii este legat de diferite domenii, în cazul în care mai mult informaţii semantice pot fi extrase din caietul de sarcini, Prin urmare, mai bine limba-ţintă programe pot fi generate cu mai puţin intervenţie umană.

Această teză are următoarele contribuţii principale ale noastre ca potenţiale răspunsuri la provocările cu experienţă în domeniul industrial de astăzi eforturile.

- `Flow`, un limbaj minimalist adeziv pentru programe care combină scrise în diferite limbaje de programare specifice domeniului.scopul a debitului este de a transforma într-o limbă Haskell caietul de sarcini care este compact, dar destul de expresive pentru a descrie aplicaţii complexe ca o compozitie de domeniu specific de limbă de programe.

- Un înalt nivel de calcul model bazat pe reţele dataflow că pune în aplicare un mod independent de platformă, de a reprezenta complexe aplicatii. Ca o consecinţă a modelului, este posibil să se precizeze un simplu semantica pure de punere în aplicare (dat fiind faptul că conţinea componente sunt la rândul lor sunt, de asemenea, pur), şi oferă un mod elegant pentru a controla executarea cererii la un nivel superior, mod declarativ.

- Proiectarea şi punerea în aplicare a unui minimalist run-time sistem de cererile de sprijin construite cu limbajul de `Flow`. Structura sistemului de run-time este strâns legată de Modelul de calcul definite mai sus, ca scopul acesteia este de a furniza acelaşi captărilor de pe fiecare platforma. Prin urmare, scopul nostru a fost să scadă abstracţiuni în modelul de calcul la un nivel minim. Ea are mai multe avantaje: este mai uşor să fie portat la platforme hardware diferite şi este, de asemenea, mai uşor de a da semantica formale pentru elementele run-time de sistem.

În Capitolul 1, vom oferi o privire de ansamblu asupra problemelor fundamentale în acest domeniu de cercetare, o descriere scurtă a probleam, împreună cu o Analiza a de muncă. Soluţia noastră propusă este apoi treptat revelat prin capitolele care urmează.

În Capitolul 2, vom introduce tehnicile utilizate, care foarte specifice la domeniu şi de abordare a muncii noastre. Există Haskell ca un instrument este în curs de folosit pentru a defini un alt limbaj de programare prin metoda numita *Încorporarea limba*. Tipul de sistem sofisticat şi flexibilitatea oferită de programare cu funcţii de Haskell face destul de potrivit pentru lucrul cu prototipuri de limbă. Ca un lucru de care, *Feldspar*, un limbaj specifice domeniului pentru semnal digital prelucrarea şi paralelismul este discutat pe scurt pentru a ilustra conceptele de scufundare.

În Capitolul 3, un limbaj de adeziv pentru limbile mici numit `Flow` este descris ca fiind contribuţia noastră.Scopul este de a menţine acelaşi gradul de composability este furnizat de către *Feldspar* descrise în capitolul precedent, dar la un nivel mai ridicat în cazul în care programele sunt reprezentate în diverse domenii specifice de limbi.Scopul `Flow` este, de asemenea, pentru a arăta modul în care componentele tipice într-un astfel de sistem pot fi capturate într-un mod abstract. În afară de faptul că, de semantica o masina de abstract este dat pentru a explica modul în care pentru a rula rezultat programe.

În Capitolul 4, vom continua cu analiza a programelor în raport cu rulează eficient aplicaţii scrise în limba `Flow`. Problemele discutate se numără modul în care componentele sunt programat pentru executarea prin distribuirea resurselor disponibile între le. În timpul anchetei, o generaţie strategie de cod, împreună cu un mic run-time sistem este elaborat, care să contribuie la minimizarea costurilor de complexitate care rezultă din utilizarea de o funcţională limbajul de programare. În plus faţă de aceasta, un mod de utilizare, controlabil de programarea este definit, care ajută la parallelize sus abstract maşină.

În Capitolul 5, discuţia continuă pe `Flow` de a lua o Cererea de exemplu avansat şi să demonstreze de design. Cu toate acestea, ne intenţia de mai departe cu aceasta aplicatie este de a caracteriza în mod explicit modela şi, astfel, o anumită clasă de sisteme de operare. Astfel de sisteme sunt frecvent utilizate pe hardware-ul încorporat, în cazul în care scopul de a sistemul de operare este de a opera o placa hardware dedicat pentru anumite sarcini - şi se limitează la un domeniu de aplicaţii specifice, de exemplu, digitală de procesare a semnalului. Prin urmare, luăm *Feldspar* de la capitolul 2 şi de a crea o prelungire pentru ca acesta să continue cu astfel de programare hardware-ul la un nivel mai ridicat.

În Capitolul 6, vom compara rezultatele noastre cu realizările actuale ale domeniu, ceea ce discuta principalele diferenţe sunt în abordarea noastră: ceea ce avantaje şi dezavantaje are în reflecţie de rezultatele altora şi răspunsuri la întrebările legate.

În Capitolul 7, concluziile noastre cu privire la acest subiect sunt rezumate în încheiere.

Concluzii

Această teză rezumă încercarea noastră de a folosi o funcţională contemporan limbă, Haskell, ca un limbaj de specificare şi de a dezvolta un compilator pentru ea. Am folosit tehnica de limbă embeddeding pentru a crea un lipici limbă, numit `Flow` să pună în aplicare o metodă pentru a compune programe scrise în arbitrare integrate specifice domeniului de limbi. În timpul discuţiei, am definit anumite convenţii şi condiţii care ajuta la construirea de aplicaţii pentru a fi rulat direct, ca parte a sistemelor integrate. Am folosit *Feldspar*, un domeniu specific încorporat limba ca un exemplu pentru a demonstra cum de a construi o cerere în domeniul de prelucrare a semnalului digital. De procesare a semnalului digital este un domeniu important în industrie şi cu apariţia de multi-şi pro-

cesoare multi-core, în mod constant caută noi tehnologii pentru dezvoltarea sistemelor într-un mod rapid, dar şi de încredere suspendable.

Limba `Flow` a fost conceput cu cerinţele de astfel de sisteme în minte. Desigur, noi credem că limba `Flow` nu poate fi folosit doar pentru a construi sisteme de procesare digitală a semnalului, dar oferă suficientă flexibilitate pentru a fi utilizate în alte domenii, sau chiar cu algoritmi de mai multe domenii în acelaşi timp. În ceea ce priveşte acest lucru, `Flow` oferă libertatea de a autorilor acestor limbi mici ca ea are doar cerinţele minime cu privire la punerea în aplicare, şi-l face posibil, pentru a se extinde în continuare în interiorul anumită limbă pentru a ajunge la noi goluri.

Pe baza experienţelor în timpul nostru de cercetare in zona, putem trage următoarele concluzii.

- *Specifice domeniului de natura descrierea prevede o mai bună vizibilitate pentru a compilatorului, care pot fi exploatate în timpul optimizării şi generare a codului.* În cazul în care construcţiile pentru un program sunt prea scăzut nivel, atunci compilatorul trebuie să fie foarte inteligent pentru a identifica în abstracţii sursa programului. Un alt pericol de a fi de nivel scăzut este faptul că este uşor pentru a scrie programe care sunt greu de motive cu privire la, greu de a stabili indiferent dacă acestea sunt conforme cu caietul de sarcini. Specifice domeniului de limbi, pe de altă parte, oferă, de obicei, mai multe construcţii de nivel înalt, care ajuta la îndruma gândurile programator în jurul valorii de un model abstract, dat fiind că surprinde principalelor concepte ale domeniului. Prin utilizarea domeniului specific constructe, programator este capabil de a dezvălui mai mult de intenţiile sale, şi astfel compilator este capabil de a afla mai multe despre problema să fie rezolvată. Într-un caz fericit, definiţia de limba în sine coincide cu o specificaţie formală, şi anume că programele deveni ele însele caietul de sarcini. În caz de feldspat, ei sunt formule matematice că programele corespunzătoare pun în aplicare susţinută de [8, 9].

- *Caracterul declarativ al descrierii ajută programator la evita munca de erori şi de creativitate scrierea de cod boilerplate.* Restructurarea cererii necesită mai puţin efort. Din punctul de vedere elaborarea, abordarea declarativ punct de vedere tehnic, contribuie la caietul de sarcini fiind privit ca intrare pentru o bază de cunoştinţe, în care compilator are o oarecare libertate în traducere, astfel cum se ştie mai multe despre sistem, şi se ştie acest lucru la un nivel mai ridicat.Vedere declarativ, de asemenea, motivează programator să nu fie prea slab în detalii ale soluţie. O astfel de mod de a extrage esenţialul face rezultat programe de compacte - similare cu expresia reprezentând `audioproc` cerere -, şi, prin urmare, uşor de înţeles şi de schimba. Care este descrisă în [54].

- *Utilizarea compoziţia este benefic în dezvoltarea de sisteme de operare ca bine.* Alegerea de a exprima un sistem de operare, ca program, ca compoziţia de mai multe straturi implică o inginerie software puternic metodologie. Amestecat cu abordarea declarativ, compoziţia - sau fuzionarea - a straturilor pot profita de eliminarea de intermediar structuri de date, şi că modul în care ele pot fi topit într-un program specializat în timp ce acesta era scris ca o compoziţie de componente generice. noi cred că limba `Flow` prezentat în [53] este un bun exemplu de cum să se proiecteze şi să implementeze un vehicul abstract care poate fi folosit ca un strat suplimentar de a re-utiliza componente existente într-un setare diferită.

– *Semantica specifice domeniului cere doar un minim de specialitate run-time de mediu, care pot fi portate la mai multe arhitecturi, fără dificultăţi majore.* În cadrul tezei, vom prezenta semantica bazele de reprezentare şi de derularea programelor ar fi reţelele de dataflow care operează numai cu câteva abstracţiuni: câteva piscine de activităţi, sarcini, mesaje cozi, şi lucrătorilor. Ca parte a tezei, am dat un beton de exemplu pe API-ul trebuie să fie puse în aplicare pentru programare C limba pe partea de sus a unui sistem compatibil POSIX. Suntem încrezători că acesta poate fi redusă până la metal, deşi nu sunt încorporate sistem producătorii (de exemplu, Tilera), care oferă deja un POSIX-compatibil run-time mediu pe partea de sus a consiliilor lor. Rezultatele au fost publicate în [54].

În rezumat, observaţiile de mai sus ne permite să concluzionăm că potenţialul de limbi funcţionale în tehnologia compilatorului vin cu anumite avantaje în dezvoltarea de sisteme de operare, în care modelele pot fi luate în caietul de sarcini pentru generarea de cod de încredere.

Ne-ar dori să recunoască suportul financiar al Ericsson Software De cercetare, de afaceri Ericsson Unităţi de reţea şi SSF (Suedia), Maghiară Agenţia Naţională de Dezvoltare (KMOP-2008-1.1.2), Programul Operaţional Dezvoltarea Resurselor Umane sectionale 2007 - 2013 (POSDRU/6/1.5/S/3-2008, România). Am dori să mulţumesc personalului membri şi studenţi ai Grupului de feldspat, Departamentul de programare Limbi şi compilatorul de la ELTE, Catedra de Informatică şi Inginerie de la Universitatea Chalmers din Tehnologie, Departamentul de Informatică de la Universitatea Babeş-Bolyai, care a contribuit la dezvoltarea acestei teze şi muncă de cercetare legate de prin comentariile lor şi de lucru. Mai mult, am ar dori exprim recunoştinţa noastră pentru a fi activ implicat în Proiectul FreeBSD, în cazul în care suntem responsabili pentru menţinerea Glasgow Haskell Compiler şi asociate Haskell pachete de intrigi, la fel de oferindu-ne perspective în sistemele de operare şi contemporane de programare funcţional, în acelaşi timp. Acesta este la fel de important pentru ne că am avut oportunitatea de a preda Haskell pe ELTE la nivel începător şi avansat pentru mulţi semestre.

Bibliografie

[1] M. Accetta, R. Baron, W. Bolosky, D. Golub, R. Rashid, A. Tevanian, Y. Michael. *Mach: A New Kernel Foundation for UNIX Development*, Proc. of the USENIX 1986 Summer Conference, June 1986.

[2] E. Axelsson, G. Dévai, Z. Horváth et al. *Feldspar: A Domain-Specific Language for Digital Signal Processing Algorithms*, Proc. 8th ACM/IEEE International Conf. on Formal Methods and Models for Codesign, MemoCode, IEEE Computer Society, 2010.

[3] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, A. Warfield. *Xen and the Art of Virtualization*, Proc. of the 19th ACM Symposium on Operating Systems Principles (SOSP), New York, New York, 2003.

[4] M. Ben-Ari. *Principles of the Spin Model Checker* (1st Ed.), Springer, 2008.

[5] W.R. Bevier, L. M. Smith. *A Mathematical Model of the Mach Kernel*, Technical Report 102, Computational Logic, Inc., 1994

[6] M. Brorsson, K-F. Faxen, K. Popov. *ArchiDeS: A Programming Framework for Multicore Chips*, Swedish Workshop on Multicore Computing (MCC09), 2009.

[7] P.E. Dagand, A. Baumann, T. Roscoe. *Filet-o-Fish: Practical and Dependable Domain-Specific Language for OS Development*, ACM SIGOPS Operating Systems Review, 2010.

[8] G. Dévai, Z. Gera, Z. Horváth, G. Páli et al. *Feldspar – A Functional Embedded Language for DSP*, 8th International Conference on Applied Informatics (ICAI2010), 2010.

[9] G. Dévai, M. Tejfel, Z. Gera, G. Páli et al. *Efficient Code Generation from the High-Level Domain-Specific Language Feldspar for DSPs*, Proc. ODES-8: 8th Workshop on Optimizations for DSP and Embedded Systems, assoc. with IEEE/ACM International Symposium on Code Generation on Optimization (CGO), 2010.

[10] I.S. Diatchki, M.P. Jones, R. Leslie. *High-Level Views on Low-Level Representations*, Proceedings of the 10th ACM SIGPLAN International Conference on Functional Programming, 2005

[11] I.S. Diatchki, M.P. Jones. *Strongly Typed Memory Areas Programming Systems-Level Data Structures in a Functional Language*, Proceedings of the 2006 ACM SIGPLAN Workshop on Haskell, 2006

[12] I.S. Diatchki. *High-Level Abstractions for Low-Level Programming*, PhD thesis, OGI School & Engineering, Oregon Health & Science University, 2007

[13] I.S. Diatchki, T. Hallgren, M.P. Jones, R. Leslie, A. Tolmach. *Writing Systems Software in a Functional Language: An Experience Report*, Proceedings of the 4th Workshop on Programming Languages and Operating Systems, 2007

[14] C. Elliott, S. Finne, O. de Moor. *Compiling Embedded Languages*, Proc. Semantics, Applications and Implementation of Program Generation (SAIG 2000), 2000.

[15] Enea AB. *Enea OSE: Multicore Real-Time Operating System*, `http://www.enea.com/`, 2011.

[16] Feldspar Group. *Feldspar Home Page*, `http://feldspar.inf.elte.hu/` Date: 05-26-2011.

[17] The `feldspar-language` package. `http://hackage.haskell.org/package/feldspar-language-0.4.0.2`, Date: 04-27-2011.

[18] The `feldspar-compiler` package. `http://hackage.haskell.org/package/feldspar-compiler-0.4.0.2`, Date: 04-27-2010.

[19] M. Felleisen, B. Findler, M. Flatt et al. *Building Little Languages With Macros*, Dr. Dobb's Journal, April 2004.

[20] G. Fu. *Design and Implementation of an Operating System in Standard ML*, Master's thesis, University of Hawaii at Marona, 1999

[21] W. Fu, C. Hauser. *A Real-Time Garbage Collection Framework for Embedded Systems*, ACM SCOPES, 2005

[22] Galois, Inc. *HaLVM Home Page*, `http://www.halvm.org/`, Date: 01-15-2012.

[23] Z. Gilián. (supervisors: Z. Horváth, G. Páli) *Abstract Description of System-Level Layers in a Functional Language*, BSc. thesis, Eötvös Loránd University, Faculty of Informatics, 2010.

[24] A. Gill, J. Launchbury, S.L. Peyton Jones. *A Short-Cut to Deforestation*, Proc. Int. Conf. on Functional Programming Languages and Compiler Architecture (FPCA), 1993.

[25] M. Golm, M. Felser et al. *The JX Operating System*, Proceedings of the 2002 USENIX Annual Technical Conference, Monterey, CA., 2002

[26] K. Granuke. *Extensible Scheduling in a Haskell-Based Operating System*, Master's thesis, Portland State Universtity, 2010.

[27] T. Hallgren. *Fun with Functional Dependencies*, Proc. of the Joint CS/CE Winter Meeting, Varberg, Sweden, January 2001.

[28] T. Hallgren, M.P. Jones et al. *A Principled Approach to Operating System Construction in Haskell*, The 10th ACM SIGPLAN International Conference on Functional Programming, 2005

[29] T. Harris, S. Marlow, S. Peyton-Jones, M. Herlihy. *Composable Memory Transactions*, Proc. of the 10th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '05), Chicago, Illinois, June 15–17, 2005.

[30] The High-Assurance Systems Programming Project (Hasp). *The Habit Programming Language: The Revised Preliminary Report*, November 2010.

[31] The High-Assurance Systems Programming Project (Hasp). *Hasp Home Page*, `http://hasp.cs.pdx.edu/`, Date: 01-31-2012.

[32] Z. Horváth, Z. Hernyák, V. Zsók. *Implementing Distributed Skeletons Using D-Clean and D-Box*, Proc. of the 17th International Workshop on Implementation and Application of Functional Languages (IFL), Dublin, Ireland, September 19–21, 2005, pp. 1–16.

[33] P. Hudak, J. Hughes, S.L. Peyton Jones, P. Wadler. *A History of Haskell: Being Lazy with Class*, Proc. of the 3rd ACM SIGPLAN Conf. on History of Programming Languages (HOPL III), 2007.

[34] J. Hughes. *Why Functional Programming Matters*, The Computer Journal – Special Issue on Lazy Functional Programming, Vol. 32, Issue 2, Oxford University Press, April 1989.

[35] G.C. Hunt, J.R. Larus. *Singularity: Rethinking the Software Stack*, ACM SIGOPS Operating Systems Review, Vol. 42., No. 2., pp. 37–49., 2007

[36] D.J. King, J. Launchbury. *Lazy Depth-First Search and Linear Graph Algorithms in Haskell*, Glasgow Workshop on Functional Programming, 1994.

[37] G. Klein, R. Kolanski. *Formalising the L4 Microkernel API*, Proceedings of the 12th Computing: The Australasian Theory Symposium, 2006

[38] G. Klein, P. Derrin, K. Elphinstone. *Experience Report: seL4 – Formally Verifying a High-Performance Microkernel*, Proceedings of the 14th International Conference on Functional Programming, 2009

[39] G. Klein, K. Elphinstone et al. *seL4: Formal Verification of an OS Kernel*, Proceedings of the 22nd ACM Symposium on Operating Systems Principles, 2009

[40] C. Lattner. *LLVM: An Infrastructure for Multi-Stage Optimization*, MSc. thesis, Computer Science Dept., University of Illinois at Urbana-Champaign, 2002.

[41] E. Lee, C. Hylands, J. Janneck et al. *Overview of the Ptolemy Project* Technical Report UCB/ERL M01/11, EECS Department, University of California, Berkeley, 2001.

[42] R. Leslie. *A Functional Approach to Memory-Safe Operating Systems*, PhD thesis, Portland State University, 2011.

[43] P. Li, S. Marlow, S.P. Jones, A. Tolmach. *Lightweight Concurrency Primitives for GHC*, Haskell Workshop 2007

[44] D. Licata, S. Peyton-Jones. *View Patterns in GHC*, AngloHaskell 2007.

[45] J. Liedtke. *Improving IPC by Kernel Design.* Proc. of the 14th ACM Symposium on Operating System Principles (SOSP), pp. 175–188, 1993.

[46] J. Liedtke. *On Micro-Kernel Construction.* Proc of the 15th ACM Symposium on Operating System Principles (SOSP), pp. 237–250, 1995.

[47] A. Madhavapeddy, R. Mortier, R. Sohan, T. Gazagnaire, S. Hand, T. Deegan, D. McAuley, J. Crowcroft. *Turning Down the LAMP: Software Specialization for the Cloud*, Proc. of the 2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud), Boston, Massachusetts, 2010.

[48] S. Marlow. *Parallel and Concurrent Programming in Haskell*, Central European Functional Programming Summer School (CEFP), Eötvös Loránd University, Budapest, June 14–24, 2011.

[49] S. Marlow, R. Newton, S.L. Peyton Jones. *A Monad for Deterministic Parallelism*, Haskell '11: Proc. of the 4th ACM SIGPLAN Symposium on Haskell, Tokyo, Japan, ACM, 2011.

[50] A. McCreight, T. Chevalier, A. Tolmach. *A Certified Framework for Compiling and Executing Garbage-Collected Languages*, Proc. of 2010 ACM International Conference on Functional Programming, Baltimore, September 2010.

[51] M.K. McKusick, G.V. Neville-Neil. *The Design and Implementation of the FreeBSD Operating System*, Boston, Mass.: Addison-Wesley, 2004

[52] R. Milner, M. Tofte, R. Harper, D. MacQueen. *The Definition of Standard ML (Revised)*, MIT Press, 1997.

[53] G. Páli. *Extending Little Languages into Big Systems.* To appear in Horváth Z., Zsók V. (Eds.): Central European Functional Programming School. Fourth Summer School, CEFP 2011. Revised Selected Lectures. Lecture Notes in Computer Science, (ISSN 0302-9743), Vol. 7241, pp. 295–312.

[54] G. Páli. *Declarative Scheduling of Dataflow Networks*, Annales Universitatis Scientarium Budapestinensis de Rolando Eötvös Nominatae, Sectio Computatorica, Vol. 36(2012), 2012 *(to appear).*

[55] G. Paller. *Rafael: An Intelligent, Multi-Target, Signal-Flow Compiler*, PhD thesis, Technical University of Budapest, Department of Electromagnetic Theory, June 1995.

[56] R. Paterson. *Arrows and Computation*, The Fun of Programming, pp. 201–222, Palgrave, 2003.

[57] S.L. Peyton Jones, J. Launchbury. *State in Haskell*, Lisp and Symbolic Computation 8(4), pp. 293–341, 1995.

[58] S.L. Peyton Jones, D. Vytiniotis, S. Weirich, G. Washburn. *Simple Unification-Based Type Inference for GADTs*, Proc. of the 11th ACM SIGPLAN International Conference on Functional Programming (ICFP), pp. 50–61, Portland, Oregon, 2006.

[59] R. Pike, D. Presotto, K. Thompson, H. Trickey. *Plan 9 from Bell Labs*, Proc. of the Summer 1990 UKUUG Conference, 1990.

[60] R. Pike. *Systems Software Research is Irrevelant*, 2000.

[61] PulseAudio Home Page. `http://www.pulseaudio.org/`, Date: 10-20-2011.

[62] D. Rémy, J. Vouillon. *Objective ML: A Simple Object-Oriented Extension of ML*, ACM Symposium on Principles of Programming Languages (PLOP), 1997.

[63] D.M. Ritchie, R. Pike et al. *The Inferno Operating System*, Bell Labs Technical Journal, Vol. 2., No. 1., Winter 1997, pp. 5–18., 1997

[64] L. Ryzhyk, P. Chubb, I. Kuz, E. Le Sueur, G. Heiser. *Automatic Device Driver Synthesis with Termite*, Proceedings of the 22nd ACM Symposium on Operating System Principles, 2009

[65] F. Schneider, G. Morrissett, R. Harper. *A Language-Based Approach to Security*, Informatics: 10 Years Back, 10 Years Ahead, 2000

[66] J. Shapiro, S. Weber. *Verifying Operating System Security.* Technical Report MS-CIS97–26, University of Pennsylvania, Philadelphia, PA, USA, 1997.

[67] A. Schüpbach, S. Peter, A. Baumann et al. *Embracing Diversity in the Barrelfish Manycore Operating System*, Proc. of the Workshop on Managed Many-Core Systems (MMCS08), June 2008.

[68] A.S. Tanenbaum, A.S. Woodhull. *Operating Systems: Design and Implementation, Third Edition*, Prentice Hall, 2006

[69] S. Thompson. *Type Theory and Functional Programming*, Addison-Wesley, 1991.

[70] L. Torvalds. *Linux: A Portable Operating System*, MSc. thesis, University of Helsinki, Department of Computer Science, 1997.

[71] B. Torma. (supervisor: G. Páli) *Development of a Framework for the Simulation of Real-Time Systems in a Functional Language* BSc. thesis, Eötvös Loránd University, Faculty of Informatics, 2011.

[72] A. Vajda. *Programming Many-Core Chips* (1st Ed.), Springer, 2011.

[73] A. Weelden, R. Plasmeijer. *Towards a Strongly Typed Functional Operating System*, Implementation of Functional Languages, 14th International Workshop, 2002

[74] D. Wentzlaff, A. Agarwal. *Factored Operating Systems (fos): The Case for a Scalable Operating System for Multicores*, ACM SIGOPS Operating System Review: Special Issue on the Interfaction among the OS, Compilers, and Multicore Processors, April 2009.