David L. Hicks (Ed.)

LNCS 3002

Metainformatics

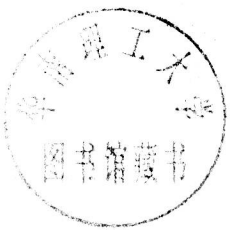
International Symposium, MIS 2003
Graz, Austria, September 2003
Revised Papers

Springer

David L. Hicks (Ed.)

Metainformatics

International Symposium, MIS 2003
Graz, Austria, September 17-20, 2003
Revised Papers



Springer

Volume Editor

David L. Hicks
Aalborg University Esbjerg
Department of Software and Media Technology
Niels Bohrs Vej 8, 6700 Esbjerg, Denmark
E-mail: hicks@cs.aue.auc.dk

Library of Congress Control Number: 2004104702

CR Subject Classification (1998): H.4, H.5.1, D.2, H.5.4, D.1, I.2, K.4, I.7

ISSN 0302-9743
ISBN 3-540-22010-0 Springer-Verlag Berlin Heidelberg New York

Springer-Verlag is a part of Springer Science+Business Media

springeronline.com

Printed in Germany

Typesetting: Camera-ready by author, data conversion by Olgun Computergrafik
Printed on acid-free paper SPIN: 11008378 06/3142 5 4 3 2 1 0

Lecture Notes in Computer Science 3002

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

Springer
Berlin
Heidelberg
New York
Hong Kong
London
Milan
Paris
Tokyo

Preface

This volume contains the final proceedings of the MetaInformatics Symposium 2003 (MIS 2003). The event was held September 17–20 on the campus of the Graz University of Technology in Graz, Austria.

As with previous events in the MIS series, MIS 2003 brought together researchers and practitioners from a wide variety of fields to discuss a broad range of topics and ideas related to the field of computer science. The contributions that were accepted to and presented at the symposium are of a wide variety. They range from theoretical considerations of important metainformatics-related questions and issues to practical descriptions of approaches and systems that offer assistance in their resolution. I hope you will find the papers contained in this volume as interesting as the other members of the program committee and I have.

These proceedings would not have been possible without the help and assistance of many people. In particular I would like to acknowledge the assistance of Springer-Verlag in Heidelberg, Germany, especially Anna Kramer, the computer science editor, and Alfred Hofmann, the executive editor for the LNCS series.

February 2004 David L. Hicks

Organization

Organizing Committee

Uffe K. Wiil (Aalborg University Esbjerg, Denmark)
David L. Hicks (Aalborg University Esbjerg, Denmark)
Peter J. Nürnberg (Technical University Graz, Austria)

Conference Secretary

Mathias Lux (Know-Center, Austria)

Program Committee

Chair	David L. Hicks (Aalborg University Esbjerg, Denmark)
Members	Darren Dalcher (Middlesex University, London, UK) David Millard (University of Southampton, UK) Peter J. Nürnberg (Technical University Graz, Austria) Siegfried Reich (Salzburg Research, Austria) Jessica Rubart (FhG-IPSI, Darmstadt, Germany) Klaus Tochtermann (Know-Center, Graz, Austria) Manolis Tzagarakis (University of Patras, Greece) Weigang Wang (FhG-IPSI, Darmstadt, Germany) Jim Whitehead (University of California, Santa Cruz, USA) Uffe K. Wiil (Aalborg University Esbjerg, Denmark)

Sponsoring Institutions

Technische Universität Graz, Austria
Styrian Competence Center for Knowledge Management (Know-Center), Austria
Aalborg University Esbjerg, Denmark

Lecture Notes in Computer Science

For information about Vols. 1–2910

please contact your bookseller or Springer-Verlag

Vol. 3042: N. Mitrou, K. Kontovasilis, G.N. Rouskas, I. Iliadis, L. Merakos (Eds.), NETWORKING 2004, Networking Technologies, Services, and Protocols; Performance of Computer and Communication Networks; Mobile and Wireless Communications. XXXIII, 1519 pages. 2004.

Vol. 3034: J. Favela, E. Menasalvas, E. Chávez (Eds.), Advances in Web Intelligence. XIII, 227 pages. 2004. (Subseries LNAI).

Vol. 3027: C. Cachin, J. Camenisch (Eds.), Advances in Cryptology - EUROCRYPT 2004. XI, 628 pages. 2004.

Vol. 3026: C. Ramamoorthy, R. Lee, K.W. Lee (Eds.), Software Engineering Research and Applications. XV, 377 pages. 2004.

Vol. 3025: G.A. Vouros, T. Panayiotopoulos (Eds.), Methods and Applications of Artificial Intelligence. XV, 546 pages. 2004. (Subseries LNAI).

Vol. 3019: R. Wyrzykowski, J. Dongarra, M. Paprzycki, J. Wasniewski (Eds.), Parallel Processing and Applied Mathematics. XIX, 1174 pages. 2004.

Vol. 3015: C. Barakat, I. Pratt (Eds.), Passive and Active Network Measurement. XI, 300 pages. 2004.

Vol. 3012: K. Kurumatani, S.-H. Chen, A. Ohuchi (Eds.), Multi-Agnets for Mass User Support. X, 217 pages. 2004. (Subseries LNAI).

Vol. 3011: J.-C. Régin, M. Rueher (Eds.), Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. XI, 415 pages. 2004.

Vol. 3010: K.R. Apt, F. Fages, F. Rossi, P. Szeredi, J. Váncza (Eds.), Recent Advances in Constraints. VIII, 285 pages. 2004. (Subseries LNAI).

Vol. 3009: F. Bomarius, H. Iida (Eds.), Product Focused Software Process Improvement. XIV, 584 pages. 2004.

Vol. 3007: J.X. Yu, X. Lin, H. Lu, Y. Zhang (Eds.), Advanced Web Technologies and Applications. XXII, 936 pages. 2004.

Vol. 3006: M. Matsui, R. Zuccherato (Eds.), Selected Areas in Cryptography. XI, 361 pages. 2004.

Vol. 3005: G.R. Raidl, S. Cagnoni, J. Branke, D.W. Corne, R. Drechsler, Y. Jin, C.G. Johnson, P. Machado, E. Marchiori, F. Rothlauf, G.D. Smith, G. Squillero (Eds.), Applications of Evolutionary Computing. XVII, 562 pages. 2004.

Vol. 3004: J. Gottlieb, G.R. Raidl (Eds.), Evolutionary Computation in Combinatorial Optimization. X, 241 pages. 2004.

Vol. 3003: M. Keijzer, U.-M. O'Reilly, S.M. Lucas, E. Costa, T. Soule (Eds.), Genetic Programming. XI, 410 pages. 2004.

Vol. 3002: D.L. Hicks (Ed.), Metainformatics. X, 213 pages. 2004.

Vol. 3001: A. Ferscha, F. Mattern (Eds.), Pervasive Computing. XVII, 358 pages. 2004.

Vol. 2999: E.A. Boiten, J. Derrick, G. Smith (Eds.), Integrated Formal Methods. XI, 541 pages. 2004.

Vol. 2998: Y. Kameyama, P.J. Stuckey (Eds.), Functional and Logic Programming. X, 307 pages. 2004.

Vol. 2997: S. McDonald, J. Tait (Eds.), Advances in Information Retrieval. XIII, 427 pages. 2004.

Vol. 2996: V. Diekert, M. Habib (Eds.), STACS 2004. XVI, 658 pages. 2004.

Vol. 2995: C. Jensen, S. Poslad, T. Dimitrakos (Eds.), Trust Management. XIII, 377 pages. 2004.

Vol. 2994: E. Rahm (Ed.), Data Integration in the Life Sciences. X, 221 pages. 2004. (Subseries LNBI).

Vol. 2993: R. Alur, G.J. Pappas (Eds.), Hybrid Systems: Computation and Control. XII, 674 pages. 2004.

Vol. 2992: E. Bertino, S. Christodoulakis, D. Plexousakis, V. Christophides, M. Koubarakis, K. Böhm, E. Ferrari (Eds.), Advances in Database Technology - EDBT 2004. XVIII, 877 pages. 2004.

Vol. 2991: R. Alt, A. Frommer, R.B. Kearfott, W. Luther (Eds.), Numerical Software with Result Verification. X, 315 pages. 2004.

Vol. 2989: S. Graf, L. Mounier (Eds.), Model Checking Software. X, 309 pages. 2004.

Vol. 2988: K. Jensen, A. Podelski (Eds.), Tools and Algorithms for the Construction and Analysis of Systems. XIV, 608 pages. 2004.

Vol. 2987: I. Walukiewicz (Ed.), Foundations of Software Science and Computation Structures. XIII, 529 pages. 2004.

Vol. 2986: D. Schmidt (Ed.), Programming Languages and Systems. XII, 417 pages. 2004.

Vol. 2985: E. Duesterwald (Ed.), Compiler Construction. X, 313 pages. 2004.

Vol. 2984: M. Wermelinger, T. Margaria-Steffen (Eds.), Fundamental Approaches to Software Engineering. XII, 389 pages. 2004.

Vol. 2983: S. Istrail, M.S. Waterman, A. Clark (Eds.), Computational Methods for SNPs and Haplotype Inference. IX, 153 pages. 2004. (Subseries LNBI).

Vol. 2982: N. Wakamiya, M. Solarski, J. Sterbenz (Eds.), Active Networks. XI, 308 pages. 2004.

Vol. 2981: C. Müller-Schloer, T. Ungerer, B. Bauer (Eds.), Organic and Pervasive Computing – ARCS 2004. XI, 339 pages. 2004.

Vol. 2980: A. Blackwell, K. Marriott, A. Shimojima (Eds.), Diagrammatic Representation and Inference. XV, 448 pages. 2004. (Subseries LNAI).

Vol. 2979: I. Stoica, Stateless Core: A Scalable Approach for Quality of Service in the Internet. XVI, 219 pages. 2004.

Vol. 2978: R. Groz, R.M. Hierons (Eds.), Testing of Communicating Systems. XII, 225 pages. 2004.

Vol. 2977: G. Di Marzo Serugendo, A. Karageorgos, O.F. Rana, F. Zambonelli (Eds.), Engineering Self-Organising Systems. X, 299 pages. 2004. (Subseries LNAI).

Vol. 2976: M. Farach-Colton (Ed.), LATIN 2004: Theoretical Informatics. XV, 626 pages. 2004.

Vol. 2973: Y. Lee, J. Li, K.-Y. Whang, D. Lee (Eds.), Database Systems for Advanced Applications. XXIV, 925 pages. 2004.

Vol. 2972: R. Monroy, G. Arroyo-Figueroa, L.E. Sucar, H. Sossa (Eds.), MICAI 2004: Advances in Artificial Intelligence. XVII, 923 pages. 2004. (Subseries LNAI).

Vol. 2971: J.I. Lim, D.H. Lee (Eds.), Information Security and Cryptology -ICISC 2003. XI, 458 pages. 2004.

Vol. 2970: F. Fernández Rivera, M. Bubak, A. Gómez Tato, R. Doallo (Eds.), Grid Computing. XI, 328 pages. 2004.

Vol. 2968: J. Chen, S. Hong (Eds.), Real-Time and Embedded Computing Systems and Applications. XIV, 620 pages. 2004.

Vol. 2967: S. Melnik, Generic Model Management. XX, 238 pages. 2004.

Vol. 2966: F.B. Sachse, Computational Cardiology. XVIII, 322 pages. 2004.

Vol. 2965: M.C. Calzarossa, E. Gelenbe, Performance Tools and Applications to Networked Systems. VIII, 385 pages. 2004.

Vol. 2964: T. Okamoto (Ed.), Topics in Cryptology – CT-RSA 2004. XI, 387 pages. 2004.

Vol. 2963: R. Sharp, Higher Level Hardware Synthesis. XVI, 195 pages. 2004.

Vol. 2962: S. Bistarelli, Semirings for Soft Constraint Solving and Programming. XII, 279 pages. 2004.

Vol. 2961: P. Eklund (Ed.), Concept Lattices. IX, 411 pages. 2004. (Subseries LNAI).

Vol. 2960: P.D. Mosses (Ed.), CASL Reference Manual. XVII, 528 pages. 2004.

Vol. 2958: L. Rauchwerger (Ed.), Languages and Compilers for Parallel Computing. XI, 556 pages. 2004.

Vol. 2957: P. Langendoerfer, M. Liu, I. Matta, V. Tsaoussidis (Eds.), Wired/Wireless Internet Communications. XI, 307 pages. 2004.

Vol. 2956: A. Dengel, M. Junker, A. Weisbecker (Eds.), Reading and Learning. XII, 355 pages. 2004.

Vol. 2954: F. Crestani, M. Dunlop, S. Mizzaro (Eds.), Mobile and Ubiquitous Information Access. X, 299 pages. 2004.

Vol. 2953: K. Konrad, Model Generation for Natural Language Interpretation and Analysis. XIII, 166 pages. 2004. (Subseries LNAI).

Vol. 2952: N. Guelfi, E. Astesiano, G. Reggio (Eds.), Scientific Engineering of Distributed Java Applications. X, 157 pages. 2004.

Vol. 2951: M. Naor (Ed.), Theory of Cryptography. XI, 523 pages. 2004.

Vol. 2949: R. De Nicola, G. Ferrari, G. Meredith (Eds.), Coordination Models and Languages. X, 323 pages. 2004.

Vol. 2948: G.L. Mullen, A. Poli, H. Stichtenoth (Eds.), Finite Fields and Applications. VIII, 263 pages. 2004.

Vol. 2947: F. Bao, R. Deng, J. Zhou (Eds.), Public Key Cryptography – PKC 2004. XI, 455 pages. 2004.

Vol. 2946: R. Focardi, R. Gorrieri (Eds.), Foundations of Security Analysis and Design II. VII, 267 pages. 2004.

Vol. 2943: J. Chen, J. Reif (Eds.), DNA Computing. X, 225 pages. 2004.

Vol. 2941: M. Wirsing, A. Knapp, S. Balsamo (Eds.), Radical Innovations of Software and Systems Engineering in the Future. X, 359 pages. 2004.

Vol. 2940: C. Lucena, A. Garcia, A. Romanovsky, J. Castro, P.S. Alencar (Eds.), Software Engineering for Multi-Agent Systems II. XII, 279 pages. 2004.

Vol. 2939: T. Kalker, I.J. Cox, Y.M. Ro (Eds.), Digital Watermarking. XII, 602 pages. 2004.

Vol. 2937: B. Steffen, G. Levi (Eds.), Verification, Model Checking, and Abstract Interpretation. XI, 325 pages. 2004.

Vol. 2936: P. Liardet, P. Collet, C. Fonlupt, E. Lutton, M. Schoenauer (Eds.), Artificial Evolution. XIV, 410 pages. 2004.

Vol. 2934: G. Lindemann, D. Moldt, M. Paolucci (Eds.), Regulated Agent-Based Social Systems. X, 301 pages. 2004. (Subseries LNAI).

Vol. 2930: F. Winkler (Ed.), Automated Deduction in Geometry. VII, 231 pages. 2004. (Subseries LNAI).

Vol. 2929: H. de Swart, E. Orlowska, G. Schmidt, M. Roubens (Eds.), Theory and Applications of Relational Structures as Knowledge Instruments. VII, 273 pages. 2003.

Vol. 2926: L. van Elst, V. Dignum, A. Abecker (Eds.), Agent-Mediated Knowledge Management. XI, 428 pages. 2004. (Subseries LNAI).

Vol. 2923: V. Lifschitz, I. Niemelä (Eds.), Logic Programming and Nonmonotonic Reasoning. IX, 365 pages. 2004. (Subseries LNAI).

Vol. 2919: E. Giunchiglia, A. Tacchella (Eds.), Theory and Applications of Satisfiability Testing. XI, 530 pages. 2004.

Vol. 2917: E. Quintarelli, Model-Checking Based Data Retrieval. XVI, 134 pages. 2004.

Vol. 2916: C. Palamidessi (Ed.), Logic Programming. XII, 520 pages. 2003.

Vol. 2915: A. Camurri, G. Volpe (Eds.), Gesture-Based Communication in Human-Computer Interaction. XIII, 558 pages. 2004. (Subseries LNAI).

Vol. 2914: P.K. Pandya, J. Radhakrishnan (Eds.), FST TCS 2003: Foundations of Software Technology and Theoretical Computer Science. XIII, 446 pages. 2003.

Vol. 2913: T.M. Pinkston, V.K. Prasanna (Eds.), High Performance Computing - HiPC 2003. XX, 512 pages. 2003. (Subseries LNAI).

Vol. 2911: T.M.T. Sembok, H.B. Zaman, H. Chen, S.R. Urs, S.H. Myaeng (Eds.), Digital Libraries: Technology and Management of Indigenous Knowledge for Global Access. XX, 703 pages. 2003.

Table of Contents

A Grand Unified Theory for Structural Computing

Peter J. Nürnberg*, Uffe K. Wiil, and David L. Hicks

Department of Computer Science, Aalborg University Esbjerg
Niels Bohrs Vej 8, DK-6700 Esbjerg, Denmark
{pnuern,ukwiil,hicks}@cs.aue.auc.dk

1 Introduction

Structural computing, in one sense, seeks to unify the notions of data and structure under a synthesized abstraction, by which data and structure become *views* to be applied as the need or desire arises. Indeed, one way of looking at structural computing is that the notions of data and structure are contextual, not essential. Any entity may be data to one person (application, agent, whatever) at one moment, and structure to another. Data and structure are matters of interpretation, not essence. What exactly this has bought us is discussed at length elsewhere [7, 10, 11].

However, despite the presence of the term "computing," structural computing research has left behavior (computation) out in the cold. Many agree that "behavior is important" [2, 13], but exactly how this abstraction should be integrated into the broader picture is unclear.

In this paper, we embark on a thought experiment: how can the notion of behavior be integrated into the data/structure synthesis, to derive a "grand unified abstraction?" Is behavior, like data and structure, non-essential – merely the product of a view to be applied as the need arises? If so, how can unifying data, structure, and behavior potentially benefit us?

It must be said up front that this thought experiment is in its early stages. Nonetheless, we present this work in the belief that it is sufficiently mature to benefit from active discussion within this field. Furthermore, we feel that the applications to metainformatics are clear: if this unification can succeed and is beneficial, it stands to impact a broad variety of fields, and potentially draw in researchers from hereto "foreign" fields such as computational theory.

The remainder of the paper is organized as follows: Section 2 presents a review of the data/structure unification undertaken thus far within structural computing. Section 3 describes one extremely simple (but Turing complete) behavior model and a prototypic implementation of this model in Java. Section 4 considers the synthesis of behavior and the data/structure abstraction. Section 5 considers some connections between the work presented here and other fields.

* The first author kindly acknowledges the Styrian Competence Center for Knowledge Management Research for partially supporting the work presented here. Please see http://www.know-center.at/ for more details.

D.L. Hicks (Ed.): MIS 2003, LNCS 3002, pp. 1–16, 2004.

Section 6 concludes the paper with discussions of future directions for this research.

2 Data/Structure Synthesis

In [8], it is claimed that hypermedia is about structure, but that most hypermedia systems treat structure as a derived abstraction. Data underlies most hypermedia systems. Structure is something built out of data. In this view of the world (prevalent not only in hypermedia but in almost every field) data is atomic. Some instances of data abstractions might be interpreted by some systems in certain ways. This implies that, at the backend, what is needed are generalized data management systems. Infrastructure that manages data is sufficient for hypermedia (and, indeed, any) "higher-level" application.

Structural computing disagrees with this assertion. Hypermedia is not just an application, like spreadsheet building or document management. It is an expression of an epistemological point of view – that *all* knowledge work is structuring work. Since hypermedia is about structure, it represents not as much an application as a radical rethink of the way systems (intended for human users, anyway) should be designed. Such systems need to account for the omnipresence of structure. Furthermore, this structure is arbitrary in nature. Associations built by humans among concepts are arbitrary (not regular in an obviously formalizable and computable way). Systems should support this non-formal, non-computable, arbitrary structure at the backend. Doing this is an attempt to make the technology of the computer fit the way people think, instead of forcing people to think as the computer is designed.

This has certain fairly radical implications for the way in which structure and data are related to one another. In the traditional "data-centric" view of systems construction, structure is an abstraction generated at the middleware layer at runtime. Data is an essential characteristic of all entities managed by systems. In the structural computing view, both data and structure are runtime views. As such, any entity may stand in relation to other entities, or even represent relations among other entities, at any time. Indeed, system infrastructure must assume that any entity is at all times (at least potentially) both *structured* and *structuring*.

What does this mean? Examples of interpretations of the structured and structuring roles (or in short, the structural nature) of entities are numerous. A short list here can serve to refresh the reader's memory: versioning; notification; access control; and, concurrency control – all have been shown to be non-trivially more complicated when managed entities are handled structurally.

What is the "atomic" abstraction in structural computing systems? Traditionally, different research groups have named this abstraction differently, but most names imply a highly structural view: abstract structural element [16] or structural unit [18] are two fine examples. This belies the fact, however, that the atomic abstraction is more correctly seen as a *synthesis* of data and structure. It is not as if structural interpretations of, say, versioning *replace* data inter-

pretations, but rather that they augment them. In this view, more appropriate names for the basic atomic abstraction might be the "dataic/structural atom" or "dataic/structural unit" – with the obvious caveat that this is, to say the least, awkward. Nonetheless, the point is clear: structural computing advocates a data/structure synthesis as the system atomic abstraction to be managed at the infrastructure layer of systems.

There are benefits that are immediately derived from such a viewpoint. Indeed, much structural computing research focuses on these benefits rather than any theoretical reasoning for such system constructions in the first place [1, 17]. Some researchers have gone so far as to claim that finding and articulating such benefits should be the main priority of the field [3]. Such claimed benefits usually fall into one of three categories: omnipresence of structural support (at the middleware layer) – the so-called "convenience" arguments; efficiency of structural operation implementation – the aptly named "efficiency" arguments; or, the guaranteed basic structural interpretation of entities – the so-called "interoperability" arguments.

Convenience arguments, such as those presented in [1], claim that by providing structural abstractions at the infrastructure layer, middleware services, such as software management systems, that export structural abstractions (and operations) are more easily (and consistently) implemented.

Efficiency arguments, such as those presented in [9], claim that by sinking the implementation of structural operations into the infrastructure, certain operations may execute more efficiently, even operation that are seemingly (or traditionally viewed as) non-structural, such as prefetching and caching.

Interoperability arguments, such as those presented in [6], claim that by guaranteeing some basic structural interpretation to be present at all times in the infrastructure, recasting of structural entities specialized for one application domain (e.g., navigational hypermedia) into those for another (e.g., spatial hypermedia) may be made easier.

Behind all of these arguments lies the basic claim that the data/structure synthesis is a better atomic abstraction that a solely data-oriented abstraction. Yet despite the nearly universal claim of the fundamental importance of behavior, the nearly exclusive focus on the data/structure synthesis has left behavior as a "residual," an unanalyzed concept of second-class importance.

In this paper, then, we make the following claim: analogous to the rather curious second-class, derived status of structure in hypermedia (a field ostensibly about structure), behavior occupies a second-class, derived status in structural computing (a field arguably as much about behavior as structure or data). Here, we look at some first steps toward remedying this state of affairs.

3 OISC – A Simple Turing-Complete Behavior Model

In this section, we present a very simple behavior model, which has the desirable characteristic of being Turing-complete. It is not yet clear whether this is a sufficient behavior model in practice (almost certainly this is not the case), but

it clearly is sufficient in theory. Any theoretically complete model can serve as a basis for more practical and useful models through transformation of these more complex models into their simple counterpart.

3.1 Defining subleq

OISC stands for "one instruction set computer" – analogous to the well-known CISC and RISC abbreviations [19]. The OISC instruction "set" indeed consists of only one instruction, but is nonetheless Turing-complete[1]. This instruction takes three parameters and is concisely described as "subtract and branch on result less than or equal to zero," abbreviated `subleq a b c`. (Hereafter, both the instruction set and any language based upon it are referred to as "subleq.") In "C" style pseudocode, this instruction does the following:

```
*b-= *a;
if (*b <= 0) goto c;
```

Oddly enough, subleq "programs" consist of parameters only, since no instruction operation codes are needed. "Well-formed" subleq programs, then, consist of triples of parameters, in which the first two parameters name locations to be manipulated algebraically, and the third names a potential jump location. We don't define "well-formed" here, but intuitively, we mean programs for which no parameter operates both as a algebraic location and as a jump location (though of course nothing prevents this from potentially being the case in arbitrary subleq programs.)

Like most minimal Turing-complete languages, subleq isn't very useful by itself. For example, there is no "halt" instruction, meaning the "program counter" (tape head) must be run off the end of the tape to halt the machine. There are no I/O instructions either, meaning the "memory" (tape) on which a program runs must be initialized (somehow). These, theoretically speaking, are details, however.

For simplicity and brevity, the `subleq a b` is to be interpreted as implying a "c" value that corresponds to whatever value to the PC would be if the jump condition failed. That is, the shorthand `subleq a b` implies that no jump is executed, regardless of the value of "b."

3.2 A Prototypic Implementation of subleq

We have built a prototypic implementation of a subleq virtual machine and a simple operating system and assembler for use with this virtual machine to experiment with subleq. This implementation is available under the GNU GPL from the Department of Computer Science at Aalborg University Esbjerg. Please see `<http://cs.aue.auc.dk/~pnuern/subleq/>` for more details. A description of this implementation is presented here.

[1] The proof of this is left as an exercise for the reader!

Virtual Machine. There is little to say about the virtual machine (VM) – it has only one instruction! When a VM instance is created, it has a fixed amount of memory. This memory is conceptually a one-dimensional array of integers, initially set to all zeroes. Program images (integer arrays) may be loaded into memory at arbitrary locations. The VM is able to execute the subleq instruction at any arbitrary given location in memory and supports "core dumping" for any range of memory. The VM will halt if asked to execute subleq at any "out of bounds" memory location.

Operating System. The operating system (OS) is substantially more complicated than the VM. Strictly speaking, the OS is an operating system "simulation," since it is neither implemented in subleq, nor runs on the VM. Nonetheless, it performs many of the functions of a very simple OS, and "appears" as an OS for the subleq VM to subleq programmers.

The basic abstraction of the OS is that of *process*, which consists of zero (sic) or more instances of the *thread* abstraction. A process is defined by: a range in memory; a set of threads; a name; and, a symbol table. A thread is defined by a program counter. Note that these abstractions are somewhat "poorer" than traditional notions of these same terms. Processes are not, for example, divided into text, data, stack, and heap sections. Processes may freely(!) modify their own "text" section memory, have fixed sizes (no dynamic allocation), and are responsible for all "free space" management themselves. Threads do not have register sets (there are no registers in the subleq VM), call stacks (these, also, do not exist in the VM) or any thread specific data. Processes with zero threads are also non-traditional – such "processes" are essentially global data.

The OS (intentionally) enforces no protection for process memory (with exceptions, see below). Instructions may freely reference memory locations outside the process (memory range) in which the instruction itself is executed.

There are two "special" processes – the *I/O* process and the *system* process. In the current implementation, both are 48 integers large, and are automatically loaded into locations 0-47 and 48-95, respectively. Program counter values inside these processes' memory space cause the VM to halt. (Technically, the OS translates all instructions that attempt to set the PC to a location inside these processes to "equivalent" instructions that set the PC to -1, which will cause the VM to halt.) These special processes are explained in more detail below.

I/O Process. Read and write operations to I/O locations behave specially. The 48 locations of the I/O block are to be interpreted as references to (up to) 48 I/O streams. The first 3 locations in the I/O block are hardwired to standard in, standard out, and standard error, respectively. References to I/O locations in the "a" parameter location result in a "read" from the appropriate I/O stream – the value read substitutes for the the value stored at "a". References to I/O block locations in the "b" parameter location result in a "write" of the "a" value to the appropriate I/O stream. More explicitly, consider the following subleq instructions and their "C" pseudocode counterparts: