Stefano Berardi
Mario Coppo
Ferruccio Damiani (Eds.)

LNCS 3085

Types for Proofs and Programs

International Workshop, TYPES 2003
Torino, Italy, April/May 2003
Revised Selected Papers

Springer

Stefano Berardi Mario Coppo
Ferruccio Damiani (Eds.)

Types for Proofs and Programs

International Workshop, TYPES 2003
Torino, Italy, April 30 - May 4, 2003
Revised Selected Papers

Springer

Volume Editors

Stefano Berardi
Mario Coppo
Ferruccio Damiani
Università di Torino, Dipartimento di Informatica
C. Svizzera 185, 10149 Torino, Italy
E-mail: {berardi, coppo, damiani}@di.unito.it

Library of Congress Control Number: 2004106869

CR Subject Classification (1998): F.3.1, F.4.1, D.3.3, I.2.3

ISSN 0302-9743
ISBN 3-540-22164-6 Springer-Verlag Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by PTP-Berlin, Protago-TeX-Production GmbH
Printed on acid-free paper SPIN: 11012856 06/3142 5 4 3 2 1 0

Lecture Notes in Computer Science 3085

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

Lecture Notes in Computer Science

For information about Vols. 1–2985

please contact your bookseller or Springer-Verlag

Vol. 3092: J. Eckstein, H. Baumeister (Eds.), Extreme Programming and Agile Processes in Software Engineering. XVI, 358 pages. 2004.

Vol. 3091: V. van Oostrom (Ed.), Rewriting Techniques and Applications. X, 313 pages. 2004.

Vol. 3089: M. Jakobsson, M. Yung, J. Zhou (Eds.), Applied Cryptography and Network Security. XIV, 510 pages. 2004.

Vol. 3085: S. Berardi, M. Coppo, F. Damiani (Eds.), Types for Proofs and Programs. X, 409 pages. 2004.

Vol. 3084: A. Persson, J. Stirna (Eds.), Advanced Information Systems Engineering. XIV, 596 pages. 2004.

Vol. 3083: W. Emmerich, A.L. Wolf (Eds.), Component Deployment. X, 249 pages. 2004.

Vol. 3078: S. Cotin, D.N. Metaxas (Eds.), Medical Simulation. XVI, 296 pages. 2004.

Vol. 3077: F. Roli, J. Kittler, T. Windeatt (Eds.), Multiple Classifier Systems. XII, 386 pages. 2004.

Vol. 3076: D. Buell (Ed.), Algorithmic Number Theory. XI, 451 pages. 2004.

Vol. 3074: B. Kuijpers, P. Revesz (Eds.), Constraint Databases and Applications. XII, 181 pages. 2004.

Vol. 3073: H. Chen, R. Moore, D.D. Zeng, J. Leavitt (Eds.), Intelligence and Security Informatics. XV, 536 pages. 2004.

Vol. 3070: L. Rutkowski, J. Siekmann, R. Tadeusiewicz, L.A. Zadeh (Eds.), Artificial Intelligence and Soft Computing - ICAISC 2004. XXV, 1208 pages. 2004. (Subseries LNAI).

Vol. 3066: S. Tsumoto, R. S lowiński, J. Komorowski, J.W. Grzymala-Busse (Eds.), Rough Sets and Current Trends in Computing. XX, 853 pages. 2004. (Subseries LNAI).

Vol. 3065: A. Lomuscio, D. Nute (Eds.), Deontic Logic in Computer Science. X, 275 pages. 2004. (Subseries LNAI).

Vol. 3064: D. Bienstock, G. Nemhauser (Eds.), Integer Programming and Combinatorial Optimization. XI, 445 pages. 2004.

Vol. 3063: A. Llamosí, A. Strohmeier (Eds.), Reliable Software Technologies - Ada-Europe 2004. XIII, 333 pages. 2004.

Vol. 3062: J.L. Pfaltz, M. Nagl, B. Böhlen (Eds.), Applications of Graph Transformations with Industrial Relevance. XV, 500 pages. 2004.

Vol. 3060: A.Y. Tawfik, S.D. Goodwin (Eds.), Advances in Artificial Intelligence. XIII, 582 pages. 2004. (Subseries LNAI).

Vol. 3059: C.C. Ribeiro, S.L. Martins (Eds.), Experimental and Efficient Algorithms. X, 586 pages. 2004.

Vol. 3058: N. Sebe, M.S. Lew, T.S. Huang (Eds.), Computer Vision in Human-Computer Interaction. X, 233 pages. 2004.

Vol. 3056: H. Dai, R. Srikant, C. Zhang (Eds.), Advances in Knowledge Discovery and Data Mining. XIX, 713 pages. 2004. (Subseries LNAI).

Vol. 3054: I. Crnkovic, J.A. Stafford, H.W. Schmidt, K. Wallnau (Eds.), Component-Based Software Engineering. XI, 311 pages. 2004.

Vol. 3053: C. Bussler, J. Davies, D. Fensel, R. Studer (Eds.), The Semantic Web: Research and Applications. XIII, 490 pages. 2004.

Vol. 3052: W. Zimmermann, B. Thalheim (Eds.), Abstract State Machines 2004. Advances in Theory and Practice. XII, 235 pages. 2004.

Vol. 3051: R. Berghammer, B. Möller, G. Struth (Eds.), Relational and Kleene-Algebraic Methods in Computer Science. X, 279 pages. 2004.

Vol. 3050: J. Domingo-Ferrer, V. Torra (Eds.), Privacy in Statistical Databases. IX, 367 pages. 2004.

Vol. 3047: F. Oquendo, B. Warboys, R. Morrison (Eds.), Software Architecture. X, 279 pages. 2004.

Vol. 3046: A. Laganà, M.L. Gavrilova, V. Kumar, Y. Mun, C.K. Tan, O. Gervasi (Eds.), Computational Science and Its Applications – ICCSA 2004. LIII, 1016 pages. 2004.

Vol. 3045: A. Laganà, M.L. Gavrilova, V. Kumar, Y. Mun, C.K. Tan, O. Gervasi (Eds.), Computational Science and Its Applications – ICCSA 2004. LIII, 1040 pages. 2004.

Vol. 3044: A. Laganà, M.L. Gavrilova, V. Kumar, Y. Mun, C.K. Tan, O. Gervasi (Eds.), Computational Science and Its Applications – ICCSA 2004. LIII, 1140 pages. 2004.

Vol. 3043: A. Laganà, M.L. Gavrilova, V. Kumar, Y. Mun, C.K. Tan, O. Gervasi (Eds.), Computational Science and Its Applications – ICCSA 2004. LIII, 1180 pages. 2004.

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. 3039: M. Bubak, G.D.v. Albada, P.M. Sloot, J.J. Dongarra (Eds.), Computational Science - ICCS 2004. LXVI, 1271 pages. 2004.

Vol. 3038: M. Bubak, G.D.v. Albada, P.M. Sloot, J.J. Dongarra (Eds.), Computational Science - ICCS 2004. LXVI, 1311 pages. 2004.

Vol. 3037: M. Bubak, G.D.v. Albada, P.M. Sloot, J.J. Dongarra (Eds.), Computational Science - ICCS 2004. LXVI, 745 pages. 2004.

Vol. 3036: M. Bubak, G.D.v. Albada, P.M. Sloot, J.J. Dongarra (Eds.), Computational Science - ICCS 2004. LXVI, 713 pages. 2004.

Vol. 3035: M.A. Wimmer (Ed.), Knowledge Management in Electronic Government. XII, 326 pages. 2004. (Subseries LNAI).

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

Vol. 3033: M. Li, X.-H. Sun, Q. Deng, J. Ni (Eds.), Grid and Cooperative Computing. XXXVIII, 1076 pages. 2004.

Vol. 3032: M. Li, X.-H. Sun, Q. Deng, J. Ni (Eds.), Grid and Cooperative Computing. XXXVII, 1112 pages. 2004.

Vol. 3031: A. Butz, A. Krüger, P. Olivier (Eds.), Smart Graphics. X, 165 pages. 2004.

Vol. 3030: P. Giorgini, B. Henderson-Sellers, M. Winikoff (Eds.), Agent-Oriented Information Systems. XIV, 207 pages. 2004. (Subseries LNAI).

Vol. 3029: B. Orchard, C. Yang, M. Ali (Eds.), Innovations in Applied Artificial Intelligence. XXI, 1272 pages. 2004. (Subseries LNAI).

Vol. 3028: D. Neuenschwander, Probabilistic and Statistical Methods in Cryptology. X, 158 pages. 2004.

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. 3024: T. Pajdla, J. Matas (Eds.), Computer Vision - ECCV 2004. XXVIII, 621 pages. 2004.

Vol. 3023: T. Pajdla, J. Matas (Eds.), Computer Vision - ECCV 2004. XXVIII, 611 pages. 2004.

Vol. 3022: T. Pajdla, J. Matas (Eds.), Computer Vision - ECCV 2004. XXVIII, 621 pages. 2004.

Vol. 3021: T. Pajdla, J. Matas (Eds.), Computer Vision - ECCV 2004. XXVIII, 633 pages. 2004.

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

Vol. 3016: C. Lengauer, D. Batory, C. Consel, M. Odersky (Eds.), Domain-Specific Program Generation. XII, 325 pages. 2004.

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

Vol. 3014: F. van der Linden (Ed.), Software Product-Family Engineering. IX, 486 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. 3008: S. Heuel, Uncertain Projective Geometry. XVII, 205 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. 2990: J. Leite, A. Omicini, L. Sterling, P. Torroni (Eds.), Declarative Agent Languages and Technologies. XII, 281 pages. 2004. (Subseries LNAI).

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.

Preface

These proceedings contain a selection of refereed papers presented at or related to the 3rd Annual Workshop of the Types Working Group (Computer-Assisted Reasoning Based on Type Theory, EU IST project 29001), which was held during April 30 to May 4, 2003, in Villa Gualino, Turin, Italy. The workshop was attended by about 100 researchers. Out of 37 submitted papers, 25 were selected after a refereeing process. The final choices were made by the editors.

Two previous workshops of the Types Working Group under EU IST project 29001 were held in 2000 in Durham, UK, and in 2002 in Berg en Dal (close to Nijmegen), The Netherlands. These workshops followed a series of meetings organized in the period 1993–2002 within previous Types projects (ESPRIT BRA 6435 and ESPRIT Working Group 21900). The proceedings of these earlier workshops were also published in the LNCS series, as volumes 806, 996, 1158, 1512, 1657, 2277, and 2646. ESPRIT BRA 6453 was a continuation of ESPRIT Action 3245, Logical Frameworks: Design, Implementation and Experiments. Proceedings for annual meetings under that action were published by Cambridge University Press in the books "Logical Frameworks", and "Logical Environments", edited by G. Huet and G. Plotkin.

We are very grateful to the members of the research group "Semantics and Logics of Computation" of the Computer Science Department of the University of Turin, who helped organize the Types 2003 meeting in Torino. We especially want to thank Daniela Costa and Claudia Goggioli for the secretarial support, Sergio Rabellino for the technical support, and Ugo de' Liguoro for helping out in various ways.

We also acknowledge the support from the Types Project, EU IST 29001, which makes the Types workshops possible.

March 2004

Stefano Berardi
Mario Coppo
Ferruccio Damiani

Referees

We would like to thank the following people for their kind work in reviewing the papers submitted to these proceedings:

Michael Abbott
Peter Aczel
Robin Adams
Yohji Akama
Fabio Alessi
Thorsten Altenkirch
Chris Andersen
Steffen van Bakel
Clemens Ballarin
Franco Barbanera
Gianpaolo Bella
Gianluigi Bellin
Stefano Berardi
Chantal Berline
Yves Bertot
Frédéric Blanqui
Kim Bruce
Iliano Cervesato
Alberto Ciaffaglione
Norman Danner
Ugo de' Liguoro
Fer-Jan de Vries
Pietro Di Gianantonio
Peter Dybjer
Maribel Fernandez
Jean-Christophe Filliatre
Matthew Flatt
Daniel Fridlender
Herman Geuvers
Pola Giannini
Elio Giovannetti
Adam Grabowsky
Hugo Herbelin
Roger Hindley
Daniel Hirkschoff
Marieke Huisman
Pierre Lescanne
Cedric Lhoussaine
Yong Luo
Zhaohui Luo
Simone Martini
James McKinna
Marino Miculan
Christine Paulin-Mohring
Jens Palsberg
Randy Pollak
Francois Pottier
Frédéric Prost
Christophe Raffalli
Aarne Ranta
Eike Ritter
Simona Ronchi
Pino Rosolini
Luca Roversi
Frédéric Ruyer
Ivan Scagnetto
Vincent Simonet
Jan Smith
Sergei Soloviev
Bas Spitters
Dan Synek
Paul Taylor
Tarmo Uustalu
Femke van Raamsdonk
Jen von Plato
Hongwei Xi
Yamagata Yoriyuki

Table of Contents

A Modular Hierarchy of Logical Frameworks

Robin Adams

University of Manchester
robin.adams@ma.man.ac.uk

Abstract. We present a method for defining logical frameworks as a collection of features which are defined and behave independently of one another. Each feature is a set of grammar clauses and rules of deduction such that the result of adding the feature to a framework is a conservative extension of the framework itself. We show how several existing logical frameworks can be so built, and how several much weaker frameworks defined in this manner are adequate for expressing a wide variety of object logics.

1 Introduction

Logical frameworks were invented because there were a large number of differing systems of logic, with no common language or environment for their investigation and implementation. However, we now find ourselves in the same situation with the frameworks themselves. There are many systems that are used as logical frameworks, and it is often difficult to compare them or share results between them. It is often much work to discover whether two frameworks can express the same class of object logics, or whether one is stronger or weaker than the other. If we are interested in metavariables, and we compare Pientka and Pfenning's work [1] with Jojgov's [2], it is difficult to see which differences are due to the different handling of metavariables, and which are due to differences in the underlying logical framework.

To redress this situation somewhat, I humbly present the first steps towards a common scheme within which a surprising number of different frameworks can be fitted. We take a modular approach to the design of logical frameworks, defining a framework by specifying a set of *features*, each of which is defined and behaves independently of the others. Together, all the frameworks that can be built from a given set of features form a *modular hierarchy* of logical frameworks.

We may give an informal definition of a feature thus:

> A *feature* F is a set of grammar clauses and rules of deduction such that, for any logical framework L, the result of adding F to L is a conservative extension of L.

(This cannot be made a formal definition, as we do not (yet) have a notion of "any logical framework".)

S. Berardi, M. Coppo, and F. Damiani (Eds.): TYPES 2003, LNCS 3085, pp. 1–16, 2004.

It is not surprising that features exist — one would expect, for example, that adding a definitional mechanism to a typing system should yield a conservative extension. Perhaps more surprising is the fact that such things as lambda-abstraction can be regarded as features. In fact, we shall show how a logical framework can be regarded as being nothing but a set of features. More precisely, we shall define a system that we call the *basic framework* **BF**, and a number of features that can be added to it, and we shall show how a number of existing frameworks can be built by selecting the appropriate features.

We shall also show that most of these features are unnecessary from the theoretical point of view — that is, a much smaller set of features suffices to express a wide variety of object logics. These 'unnecessary' features may well be desirable for implementation, of course.

It may be asked why we insist that our features always yield conservative extensions. This would seem to be severely limiting; in one's experience with typing systems, rarely are extensions conservative. For typing systems in general, this is true. But I would argue that logical frameworks are an exception. The fact that all the features presented here yield conservative extensions is evidence to this effect. And it would seem to be desirable when working with a logical framework — if we add a feature to widen the class of object logics expressible, for example, we still want the old object logics to behave as they did before.

We suggest that, if this work were taken further, it would be possible and desirable to define mechanisms such as metavariables or subtyping as features, and investigate their properties separately from one another and from any specific framework. If we did this for metavariables, for example, we would then know immediately what the properties of ELF with metavariables were, or Martin-Löf's Theory of Types with metavariables, or ...

2 Logical Frameworks

Let us begin by being more precise as to what we mean by a logical framework.

Broadly speaking, logical frameworks can be used in two distinct ways. The first is to define an object logic by means of a *signature*, a series of declarations of constants, equations, etc. The typable terms under that signature should then correspond to the terms, derivations, etc. of the object logic, using contexts to keep track of free variables and undischarged hypotheses. Examples include the Edinburgh Logical Framework [3] and Martin-Löf's Theory of Types [4]. We shall call a framework used in this way a *logic-modelling* framework.

The second is to use the logical framework as a *book-writing* system, as exemplified by the AUTOMATH family of systems [5]. The most important judgement form in such a framework is that which declares a book correct; the other judgement forms are only needed as auxiliaries for deriving this first form of judgement.

These two kinds of system behave in very similar ways. Any system of one kind can be used as a system of the other, by simply reading 'signature' for 'book', or vice versa. This is a striking fact, considering the difference in use. In

a system of the first kind, deriving that a signature is valid is just the first step in using an object logic; in a book-writing system, it is the only judgement form of importance. We shall take advantage of this similarity. Our features shall be written with logic-modelling frameworks in mind; it shall turn out that they are equally useful for building book-writing frameworks.

We consider a *logical framework* to consist of:

1. Disjoint, countably infinite sets of *variables* and *constants*.
2. A number of *syntactic classes* of *expressions*, defined in a BNF-style grammar by a set of *constructors*, each of which forms a member of one class from members of other classes, possibly binding variables.
3. Three syntactic classes that are distinguished as being the classes of *signature declarations*, *context declarations* and *judgement bodies*. Each signature declaration is specified to be either a declaration of a particular constant, or of none. Similarly, each context declaration is specified to be either a declaration of a particular variable or of none.
   We now define a *signature* to be a finite sequence of signature declarations, such that no two declarations are of the same constant. The *domain* of the signature Σ, $\mathrm{dom}\,\Sigma$, is then defined to be the sequence consisting of the constants declared in Σ, in order. Similarly, we define a *context* to be a finite sequence of context declarations, no two of the same variable, and we define its domain similarly.
   Finally, we define a *judgement* to be a string of one of two forms: either
   $$\Sigma\ \mathrm{sig}$$
   or
   $$\Gamma \vdash_\Sigma J$$
   where Σ is a signature, Γ a context, and J a judgement body.
4. A set of *defined operations and relations* on terms. Typically, these shall include one or more relations of *reducibility* and *convertibility*.
5. The final component of a logical framework is a set of *rules of deduction* which define the set of *derivable* judgements.

2.1 The Basic Framework BF

As is to be expected, **BF** is a very simple system. It allows: the declaration of variable and constant types; the declaration of variables and constants of a previously declared type; and the assertion that a variable or constant has the type with which it was declared, or is itself a type.

The grammar of **BF** is as follows:

$$\begin{aligned}
\text{Term} \quad & a ::= x \mid c \\
\text{Kind} \quad & A ::= \mathbf{Type} \mid \mathrm{El}(a) \\
\text{Signature Declaration} \quad & \delta ::= c : A \text{ of } c \\
\text{Context Declaration} \quad & \gamma ::= x : A \text{ of } x \\
\text{Judgement Body} \quad & J ::= \mathrm{valid} \mid A\ \mathrm{kind} \mid a : A
\end{aligned}$$

The rules of deduction of **BF** are given in Figure 1.

$$\frac{}{\langle\rangle \text{ sig}} \qquad \frac{\vdash_\Sigma A \text{ kind}}{\Sigma, c : A \text{ sig}} \ (c \notin \operatorname{dom} \Sigma)$$

$$\frac{\Sigma \text{ sig}}{\vdash_\Sigma \text{ valid}} \qquad \frac{\Gamma \vdash_\Sigma A \text{ kind}}{\Gamma, x : A \vdash_\Sigma \text{ valid}} \ (x \notin \operatorname{dom} \Gamma)$$

$$\frac{\Gamma \vdash_\Sigma \text{ valid}}{\Gamma \vdash_\Sigma c : A} \ (c : A \in \Sigma) \qquad \frac{\Gamma \vdash_\Sigma \text{ valid}}{\Gamma \vdash_\Sigma x : A} \ (x : A \in \Gamma)$$

$$\frac{\Gamma \vdash_\Sigma \text{ valid}}{\Gamma \vdash_\Sigma \mathbf{Type} \text{ kind}} \qquad \frac{\Gamma \vdash_\Sigma a : \mathbf{Type}}{\Gamma \vdash_\Sigma \mathrm{El}(a) \text{ kind}}$$

Fig. 1. The basic framework **BF**

3 Features and the Modular Hierarchy

A *feature* that *depends* on the logical framework L consists of any number of new entities: new syntactic classes, new constructors, new defined operations and relations and new rules of deduction. The new constructors may take arguments from new classes or those of L, bind new variables or those of L, and return expressions in new classes or those of L. In particular, they may create new signature declarations, context declarations and judgement bodies. Likewise, the new defined operations and relations should be defined on both old and new expressions, and the new rules of deduction may use both old and new judgement forms.

A feature may also introduce *redundancies*. A *redundancy* takes an old constructor and declares that it is to be replaced by a certain expression. That is, the constructor is no longer part of the grammar; wherever it appeared in a defined operation or relation or a rule of deduction, its place is to be taken by the given expression.

Now, if L' is any logical framework that extends L, we define the logical framework $L' + F$ in the obvious manner.

It should be noted that these rules of deduction are assumed to automatically extend themselves when future features are added. For example, if a feature contains the rule of deduction

$$\frac{\Gamma \vdash_\Sigma M : A}{\Gamma \vdash_\Sigma M = M : A}$$

and we later introduce a new constructor for terms M, this rule is assumed to hold for the *new* terms M as well as the old.

(Formally defining features in such a way that this is possible requires explicitly defining classes of *meta-expressions* in the manner of [6]. We shall not go into such details here.)

Finally, we define:

Definition 1. *A feature F that depends on the set of features $\{F_1, F_2, \dots\}$ is a feature that depends on the logical framework*

$$\mathbf{BF} + F_1 + F_2 + \cdots .$$

Thus, if F depends on $\{F_1, F_2, \dots\}$, we can add F to any framework in the hierarchy that contains all of F_1, F_2, ... Note that we do not stipulate in this definition whether the set $\{F_1, F_2, \dots\}$ is finite or infinite.

3.1 Parametrization

The first, and most important, of our features are those which allow the declaration of variables and constants with *parameters*. This mechanism is taken as fundamental by the systems of the AUTOMATH [5] family as well as PAL$^+$ [9], but can be seen as a subsystem of almost all logical frameworks. Parametrization provides a common core, above which the different forms of abstraction (λ-abstraction with typed or untyped domains, and with β- or $\beta\eta$-conversion, as well as PAL$^+$-style abstraction by let-definition) can be built as conservative extensions.

We define a series of features: **SPar** (1), **SPar** (2), **SPar** (3), ..., and also **LPar** (1), **LPar** (2), **LPar** (3), ... These extend one another in the manner shown in Figure 2.

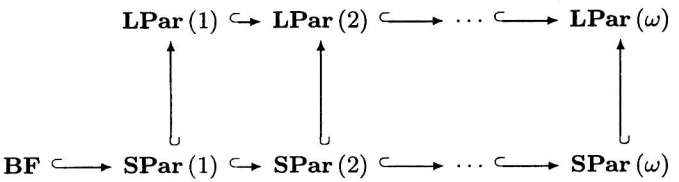

Fig. 2. The initial fragment of the modular hierarchy

BF already allows the declaration of constants in kinds: $c_1 : A$.

LPar (1) allows the declaration of constants in *first-order kinds*: $c_2 : (x_1 : A_1, \dots, x_n : A_n)A$. This declaration indicates that c is a constant that takes *parameters* x_1 of kind A_1, ..., x_n of kind A_n, and returns a term $c_2[x_1, \dots, x_n]$ of kind A.

LPar (2) allows the parameters themselves to have parameters: $c_3 : (x_1 : (x_{11} : A_{11}, \dots, x_{1k_1} : A_{1k_1})A_1, \dots, x_n : (x_{n1} : A_{n1}, \dots, x_{nk_n} : A_{nk_n})A_n)A$. **LPar** (3) allows these second-order parameters to have parameters, and so on. Similarly for declaration of variables.

We also define the feature **LPar** (ω) to be the union of all these features, allowing any level of parametrization.

The sequence of features **SPar** (n) is similar; the only difference is that, in **SPar** (n), every parameter must be in a *small* kind; that is, each A_i, A_{ij}, ... above must be of the form $\mathrm{El}(a)$; it cannot be **Type**. (In **SPar** (n), A itself, the rightmost kind, can be **Type** in the declaration of a constant, but not in a declaration of a variable.)

The full details of these features are as follows:

Parameters in Small Kinds, SPar (n)

Grammar Before we can introduce the new grammar constructors, we need to make a few definitions.

We define an m-th order *pure context* by recursion on m as follows. An m-th order pure context is a string of the form

$$(x_1 : (\Delta_1)\,\mathrm{El}(a_1), \ldots, x_k : (\Delta_k)\,\mathrm{El}(a_k))$$

where each x_i is a variable, all distinct, Δ_i a pure context of order $< m$, and a_i a term. Its *domain* is $(x_1, \ldots, x_k)$.

We define an *abstraction* to be a string of the form

$$[\boldsymbol{x}]M$$

where $\boldsymbol{x}$ is a sequence of distinct variables, and M a term. We take each member of $\boldsymbol{x}$ to be bound within M in this abstraction, and we define free and bound variables and identify all our expressions up to α-conversion in the usual manner. We write $\hat{}M, \hat{}N, \ldots$ for arbitrary abstractions. It is important to note that these are *not* first-class objects of every framework that contains **SPar** (n).

Now, we add the following clause to the grammar:

$$z[\hat{}\boldsymbol{M}]$$

is a term, where z is a variable or constant, and $\hat{}\boldsymbol{M}$ a sequence of abstractions.. This clause subsumes the grammar of **BF**, for $x()$ and $c()$ are terms when x is a **0**-ary variable and c a **0**-ary constant.

We also allow declarations of the form

$$c : (\Delta)A$$

in the signature, where c is a constant, Δ a pure context of order $\leq n$, and A a kind; and those of the form

$$x : (\Delta)\,\mathrm{El}(a)$$

in the context, where x is a variable, Δ a pure context of order $\leq n$, and a a term. Again, these subsume those of **BF**.