Bharat Jayaraman (Ed.)

Practical Aspects of Declarative Languages

LNCS 3057

6th International Symposium, PADL 2004
Dallas, TX, USA, June 2004
Proceedings

Springer

Bharat Jayaraman (Ed.)

Practical Aspects of Declarative Languages

6th International Symposium, PADL 2004
Dallas, TX, USA, June 18-19, 2004
Proceedings

Springer

Volume Editor

Bharat Jayaraman
University at Buffalo, The State University of New York
Department of Computer Science and Engineering
201 Bell Hall, Buffalo, NY 14260-2000, USA
E-mail: bharat@cse.buffalo.edu

Library of Congress Control Number: 2004107018

CR Subject Classification (1998): D.3, D.1, F.3, D.2

ISSN 0302-9743
ISBN 3-540-22253-7 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 PTP-Berlin, Protago-TeX-Production GmbH
Printed on acid-free paper SPIN: 11014027 06/3142 5 4 3 2 1 0

Lecture Notes in Computer Science 3057

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

Editorial Board

Preface

The International Symposium on Practical Aspects of Declarative Languages (PADL) is a forum for researchers and practitioners to present original work emphasizing novel applications and implementation techniques for all forms of declarative concepts, especially those emerging from functional, logic, and constraint languages. Declarative languages have been studied since the inception of computer science, and continue to be a vibrant subject of investigation today due to their applicability in current application domains such as bioinformatics, network configuration, the Semantic Web, telecommunications software, etc.

The 6th PADL Symposium was held in Dallas, Texas on June 18–19, 2004, and was co-located with the Compulog-Americas Summer School on Computational Logic. From the submitted papers, the program committee selected 15 for presentation at the symposium based upon three written reviews for each paper, which were provided by the members of the program committee and additional referees.

Two invited talks were presented at the conference. The first was given by Paul Hudak (Yale University) on "An Algebraic Theory of Polymorphic Temporal Media." The second invited talk was given by Andrew Fall (Dowlland Technologies and Simon Fraser University) on "Supporting Decisions in Complex, Uncertain Domains with Declarative Languages."

Following the precedent set by the previous PADL symposium, the program committee this year again selected one paper to receive the 'Most Practical Paper' award. The paper judged as the best in the criteria of practicality, originality, and clarity was "Simplifying Dynamic Programming via Tabling," by Hai-Feng Guo, University of Nebraska at Omaha, and Gopal Gupta, University of Texas at Dallas. This paper presents an elegant declarative way of specifying dynamic programming problems in tabled logic programming systems.

The PADL symposium series is sponsored in part by the Association of Logic Programming and Compulog Americas, a network of research groups devoted to the promotion of computational logic in North and South America.

We thank the University at Buffalo and the University of Texas at Dallas for their support. We gratefully acknowledge the contributions of Shriram Krishnamurthy and Pete Hopkins, Brown University, for maintaining the PADL website which provided much needed assistance in the submission and review process. Finally, we thank the authors who submitted papers to PADL 2004 and all who participated in the conference.

April 2004

Bharat Jayaraman
Program Chair

Program Committee

Maurice Bruynooghe	KU Leuven, Belgium
Veronica Dahl	Simon Fraser University, Canada
Olivier Danvy	University of Aarhus, Denmark
Stefan Decker	USC Information Sciences Institute, USA
Matthew Flatt	University of Utah, USA
Julia Lawall	DIKU, Denmark
Gopal Gupta	Univ. of Texas at Dallas, USA (General Chair)
Michael Hanus	University of Kiel, Germany
John Hughes	Chalmers University, Sweden
Joxan Jaffar	National University of Singapore
Bharat Jayaraman	University at Buffalo, USA (Program Chair)
Michael Leuschel	University of Southampton, UK
Gopalan Nadathur	University of Minnesota, USA
Enrico Pontelli	New Mexico State University, USA
C.R. Ramakrishnan	University at Stony Brook, USA
Tim Sheard	Oregon Graduate Institute, USA
Vitor Santos Costa	University of Rio de Janiero, Brazil
Paul Tarau	University of North Texas, USA

Referees

Alma Barranco-Mendoza	Martin Henz
Bernd Braßel	Glendon Holst
Stefan Bressan	Frank Huch
Manuel Carro	Maarten Mariën
Luis Castro	Ilkka Niemela
Alvaro Cortés-Calabuig	Nikolay Pelov
Stephen J. Craig	Ricardo Rocha
Diana Cukierman	Mads Torgersen
Dan Elphick	Son Cao Tran
Anderson Faustino	Remko Tronçon
Antonio Mario Florido	Jan Wielemaker

Lecture Notes in Computer Science

For information about Vols. 1–2994

please contact your bookseller or Springer-Verlag

Vol. 3096: G. Melnik, H. Holz (Eds.), Advances in Learning Software Organizations. X, 173 pages. 2004.

Vol. 3094: A. Nürnberger, M. Detyniecki (Eds.), Adaptive Multimedia Retrieval. VIII, 229 pages. 2004.

Vol. 3093: S.K. Katsikas, S. Gritzalis, J. Lopez (Eds.), Public Key Infrastructure. XIII, 380 pages. 2004.

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. 3086: M. Odersky (Ed.), ECOOP 2004 – Object-Oriented Programming. XIII, 611 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. 3068: E. André, L. Dybkj\ae r, W. Minker, P. Heisterkamp (Eds.), Affective Dialogue Systems. XII, 324 pages. 2004. (Subseries LNAI).

Vol. 3067: M. Dastani, J. Dix, A. El Fallah-Seghrouchni (Eds.), Programming Multi-Agent Systems. X, 221 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. 3057: B. Jayaraman (Ed.), Practical Aspects of Declarative Languages. VIII, 255 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. 3055: H. Christiansen, M.-S. Hacid, T. Andreasen, H.L. Larsen (Eds.), Flexible Query Answering Systems. X, 500 pages. 2004.

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. 3040: R. Conejo, M. Urretavizcaya, J.-L. Pérez-de-la-Cruz (Eds.), Current Topics in Artificial Intelligence. XIV, 689 pages. 2004. (Subseries LNAI).

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. 3018: M. Bruynooghe (Ed.), Logic Based Program Synthesis and Transformation. X, 233 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.

Table of Contents

An Algebraic Theory of Polymorphic Temporal Media

Paul Hudak

Department of Computer Science
Yale University
paul.hudak@yale.edu

Abstract. *Temporal media* is information that is directly consumed by a user, and that varies with time. Examples include music, digital sound files, computer animations, and video clips. In this paper we present a polymorphic data type that captures a broad range of temporal media. We study its syntactic, temporal, and semantic properties, leading to an algebraic theory of polymorphic temporal media that is valid for underlying media types that satisfy specific constraints. The key technical result is an axiomatic semantics for polymorphic temporal media that is shown to be both sound and complete.

1 Introduction

The advent of the personal computer has focussed attention on the *consumer*, the person who buys and makes use of the computer. Our interest is in the consumer as a person who *consumes information.* This information takes on many forms, but it is usually dynamic and time-varying, and ultimately is consumed mostly through our visual and aural senses. We use the term *temporal media* to refer to this time-varying information. We are interested in how to represent this information at an abstract level; how to manipulate these representations; how to assign a meaning, or interpretation, to them; and how to reason about such meanings.

To achieve these goals, we define a polymorphic representation of temporal media that allows combining media values in generic ways, independent of the underlying media type. We describe three types of operations on and properties of temporal media: (a) syntactic operations and properties, that depend only on the structural representation of the media, (b) temporal operations and properties, that additionally depend on time, and (c) semantic operations and properties, that depend on the meaning, or interpretation, of the media. The latter development leads to an axiomatic semantics for polymorphic temporal media that is both sound and complete.

Examples of temporal media include music, digital sound files, computer animations, and video clips. It also includes representations of some other concepts, such as dance [10] and a language for humanoid robot motion [5]. In this paper we use two running examples throughout: an abstract representation of *music* (analogous to our previous work on *Haskore* and *MDL*, DSLs for computer music

B. Jayaraman (Ed.): PADL 2004, LNCS 3057, pp. 1–15, 2004.

[9,6,7,8]), and an abstract representation of *continuous animations* (analogous to our previous work on *Fran* and *FAL* [4,3,7]).

The key new ideas in the current work are the polymorphic nature of the media type, the exploration of syntactic and temporal properties of this media type that parallel those for lists, the casting of the semantics in a formal algebraic framework, the definition of a normal form for polymorphic temporal media, and a completeness result for the axiomatic semantics. The completeness result relies on a new axiom for swapping terms in a serial/parallel construction.

We present all of our results using Haskell [12] syntax that, in most cases, is executable. Haskell's type classes are particularly useful in specifying constraints, via implicit laws, that constituent types must obey. Proofs of most theorems have been omitted in this extended abstract.

2 Polymorphic Media

We represent temporal media by a polymorphic data type:

```
data Media a = Prim a
             | Media a :+: Media a
             | Media a :=: Media a
```

We refer to `T` in `Media T` as the *base media type*. Intuitively, for values `x :: T` and `m1, m2 :: Media T`, a value of type `Media T` is either a primitive value `Prim x`, a sequential composition `m1 :+: m2`, or a parallel composition `m1 :=: m2`. Although simple in structure, this data type is rich enough to capture quite a number of useful media types.

Example 1 (Music): Consider this definition of an abstract notion of a *musical note*:

```
data Note = Rest Dur | Note Pitch Dur
type Dur    = Real
type Pitch  = (NoteName, Octave)
type Octave = Int
data NoteName = Cf | C  | Cs | Df | D  | Ds | Ef | E  | Es | Ff | F
              | Fs | Gf | G  | Gs | Af | A  | As | Bf | B  | Bs
```

In other words, a `Note` is either a pitch paired with a duration, or a `Rest` that has a duration but no pitch. `Dur` is a measure of time (duration), which ideally would be a real number; in a practical implementation a suitable approximation such as `Float`, `Double`, or `Ratio Int` would be used. A `Pitch` is a pair consisting of a note name and an octave, where an octave is just an integer. The note name `Cf` is read as "C-flat" (normally written as C♭), `Cs` as "C-sharp" (normally written as C♯), and so on.[1] Then the type:

```
type Music = Media Note
```

[1] This representation corresponds well to that used in music theory, except that in music theory note names are called *pitch classes*.

is a temporal media for music. In particular, a value `Prim (Rest d)` is a rest of duration `d`, `Prim (Note p d)` is a note with pitch `p` played for duration `d`, `m1 :+: m2` is the music value `m1` followed sequentially in time by `m2`, and `m1 :=: m2` is `m1` played simultaneously with `m2`. This representation of music is a simplified version of that used in the Haskore computer music library [9,6], which has been used successfully in several computer music applications. As a simple example:

```
let dMinor  =  Note (D,3) 1 :=: Note (F,3) 1 :=: Note (A,3) 1
    gMajor  =  Note (G,3) 1 :=: Note (B,3) 1 :=: Note (D,4) 1
    cMajor  =  Note (C,3) 2 :=: Note (E,3) 2 :=: Note (G,3) 2
in dMinor :+: gMajor :+: cMajor
```

is a ii-V-I chord progression in C major.

Example 2 (Animation): Consider this definition of a base media type for *continuous animations*:

```
type Anim = (Dur, Time -> Picture)
type Dur  = Real
type Time = Real
data Picture = EmptyPic | Circle Radius Point
             | Square Length Point | Polygon [Point]
type Point = (Real, Real)
```

A `Picture` is either empty, a circle or square of a given size and located at a particular point, or a polygon having a specific set of vertices. An `Anim` value `(d, f)` is a continuous animation whose image at time $0 \leq \mathtt{t} \leq \mathtt{d}$ is the `Picture` value `f t`. Then the type:

```
type Animation = Media Anim
```

is a temporal media for continuous animations. This representation is a simplified version of that used in Fran [4,3] and FAL [7]. As a simple example:

```
let ball1 = (10, \t -> Circle t origin)
    ball2 = (10, \t -> Circle (10-t) origin
    box   = (20, \t -> Square 1 (t,t))
in (ball1 :+: ball2) :=: box
```

is a box sliding diagonally across the screen, together with a ball located at the origin that first grows for 10 seconds and then shrinks.

3 Syntactic Properties

Before studying semantic properties, we first define various operations on the *structure* (i.e. syntax) of polymorphic temporal media values, many of which are analogous to operations on lists (and thus we borrow similar names when the analogy is strong). We also explore various *laws* that these operators obey, laws that are also analogous to those for lists [2,7].

Map. For starters, it is easy to define a polymorphic *map* on temporal media, which we do by declaring `Media` to be an instance of the `Functor` class:

```
instance Functor Media where
  fmap f (Prim n)     = Prim (f n)
  fmap f (m1 :+: m2) = fmap f m1 :+: fmap f m2
  fmap f (m1 :=: m2) = fmap f m1 :=: fmap f m2
```

`fmap` shares many properties with `map` defined on lists, most notably the standard laws for the `Functor` class:

Theorem 1. For any finite `m :: Media T1` and functions `f, g :: T1 -> T2`:

```
fmap (f . g) = fmap f . fmap g
fmap id      = id
```

`fmap` allows us to define many useful operations on specific media types, thus obviating the need for a richer data type as used, for example, in our previous work on Haskore, MDL, Fran, and Fal. For example, tempo scaling and pitch transposition of music, and size scaling and position translation of animation.

Fold (i.e. catamorphism). A fold-like function can be defined for media values, and will play a critical role in our subsequent development of the semantics of temporal media:

```
foldM :: (a->b) -> (b->b->b) -> (b->b->b) -> Media a -> b
foldM f g h (Prim x) = f x
foldM f g h (m1 :+: m2) = foldM f g h m1 `g` foldM f g h m2
foldM f g h (m1 :=: m2) = foldM f g h m1 `h` foldM f g h m2
```

Theorem 2. For any `f :: T1 -> T2`:

```
foldM (Prim . f) (:+:) (:=:) = fmap f
foldM Prim (:+:) (:=:)       = id
```

More interestingly, we can also state a *fusion law* for `foldM`:

Theorem 3. (Fusion Law) For `f :: T1->T2`, `g, h :: T2->T2->T2`, `k :: T2->T3`, and `g', h' :: T1->T3`, if:

```
f' x = k (f x)
g' (k x) (k y) = k (g x y)
h' (k x) (k y) = k (h x y)
```

then: `k . foldM f g h = foldM f' g' h'`.

Example: In the discussion below a reverse function, and in Section 4 a duration function, are defined as catamorphisms. In addition, in Section 5 we define the standard interpretation, or semantics, of temporal media as a catamorphism.

Reverse. We would like to define a function `reverseM` that *reverses*, in time, any temporal media value. However, this will only be possible if the base media type is itself reversible, a constraint that we enforce using type classes:

```
class Reverse a where
  reverseM :: a -> a
instance Reverse a => Reverse (Media a) where
  reverseM (Prim a)    = Prim (reverseM a)
  reverseM (m1 :+: m2) = reverseM m2 :+: reverseM m1
  reverseM (m1 :=: m2) = reverseM m1 :=: reverseM m2
```

Note that `reverseM` can be defined more succinctly as a catamorphism:

```
instance Reverse a => Reverse (Media a) where
  reverseM = foldM (Prim . reverseM) (flip (:+:)) (:=:)
```

Analogous to a similar property on lists, we have:

Theorem 4. For all finite `m`, if the following law holds for `reverseM :: T -> T`, then it also holds for `reverseM :: Media T -> Media T`:

```
reverseM (reverseM m) = m
```

We take the constraint in this theorem to be a law for all valid instances of a base media type `T` in the class `Reverse`. It is straightforward to prove this theorem using structural induction. However, one can also carry out an inductionless proof by using the fusion law of Theorem 3.

Example 1 (Music): We declare `Note` to be an instance of class `Reverse`:

```
instance Reverse Note where
  reverseM = id
```

In other words, a single note is the same whether played backwards or forwards. The constraint in Theorem 4 is therefore trivially satisfied, and it thus holds for music media.[2]

Example 2 (Animation): We declare `Anim` to be an instance of `Reverse`:

```
instance Reverse Anim where
  reverseM (d, f) = (d, \t -> f (d-t))
```

Note that `reverseM (reverseM (d, f)) = (d, f)`, therefore the constraint in Theorem 4 is satisfied, and the theorem thus holds for continuous animations.

[2] The reverse of a musical passage is called its *retrograde*. Used sparingly by traditional composers (two notable examples being J.S. Bach's "Crab Canons" and Franz Joseph Haydn's Piano Sonata No. 26 in A Major (Menueto al Rovescio)), it is a standard construction in modern twelve-tone music.

4 Temporal Properties

As a data structure, the `Media` type is fairly straightforward. Complications arise, however, when *interpreting* temporal media. The starting point for such an interpretation is an understanding of temporal properties, the most basic of which is *duration*. Of particular concern is the meaning of the parallel composition `m1 :=: m2` when the durations of `m1` and `m2` are different. In this paper we simply disallow this situation: i.e. `m1` and `m2` must have the same duration in a "well-formed" `Media` value. This approach does not lack in generality, since other approaches can be expressed by padding the media values appropriately (for example with rests in music, or empty images in animation).

Duration. To compute the *duration* of a temporal media value we first need a way to compute the duration of the underlying media type, which we enforce as before using type classes:

```
class Temporal a where
  dur  :: a -> Dur
  none :: Dur -> a
instance Temporal a => Temporal (Media a) where
  dur  = foldM dur (+) max
  none = Prim . none
```

The `none` method allows one to express the absence of media for a specified duration, as discussed earlier.

We take the constraint in the following lemma to be a law for any valid instance of a base media type `T` in the class `Temporal`:

Lemma 1. If the property `dur (none d) = d` holds for `dur :: T -> Dur`, then it also holds for `dur :: Media T -> Dur`.

Note that, for generality, the duration of a parallel composition is defined as the maximum of the durations of its arguments. However, as discussed earlier, we wish to restrict parallel coompositions to those whose two argument durations are the same. Thus we define:

Definition 1. A *well-formed* temporal media value `m :: Media T` is one that is finite, and for which each parallel composition `m1 :=: m2` has the property that `dur m1 = dur m2`.

Example 1 (Music): We declare `Note` to be `Temporal`:

```
instance Temporal Note where
  dur (Rest d)   = d
  dur (Note p d) = d
  none d         = Rest d
```

Example 2 (Animation): We declare `Anim` to be `Temporal`:

```
instance Temporal Anim where
  dur (d, f) = d
  none d     = (d, const EmptyPic)
```

Take and Drop. We now define two functions `takeM` and `dropM` that are analogous to Haskell's `take` and `drop` functions for lists. The difference is that instead of being parameterized by a number of elements, `takeM` and `dropM` are parameterized by *time*. As with other operators we have considered, this requires the ability to take and drop portions of the base media type, so once again we use type classes to structure the design. The expression `takeM d m` is a media value corresponding to the first `d` seconds of `m`. Similarly, `dropM d m` is all but the first `d` seconds. Both of these are very useful in practice.

```
class Take a where
  takeM :: Dur -> a -> a
  dropM :: Dur -> a -> a
instance (Take a, Temporal a) => Take (Media a) where
  takeM d m | d <= 0  = none 0
  takeM d (Prim x)    = Prim (takeM d x)
  takeM d (m1 :+: m2) = let d1 = dur m1
          in if d <= d1 then takeM d m1 else m1 :+: takeM (d-d1) m2
  takeM d (m1 :=: m2) = takeM d m1 :=: takeM d m2
  dropM ... = ... (details omitted) ...
```

Perhaps surprisingly, `takeM` and `dropM` share many properties analogous to their list counterparts, except that indexing is done in time, not in the number of elements:

Theorem 5. For all non-negative `d1, d2 :: Dur`, if the following laws hold for `takeM, dropM :: Dur -> T -> T`, then they also hold for `takeM, dropM :: Dur -> Media T -> Media T`:

```
takeM d1 . takeM d2 = takeM (min d1 d2)
dropM d1 . dropM d2 = dropM (d1+d2)
takeM d1 . dropM d2 = dropM d2 . takeM (d1+d2)
dropM d1 . takeM d2 = takeM (d2-d1) . dropM d1     -- if d2>=d1
```

There is one other theorem that we would *like* to hold, whose corresponding version for lists in fact does hold:

Theorem 6. For all finite well-formed `m :: Media a` and non-negative `d :: Dur <= dur m`, if the following law holds for `takeM, dropM :: Dur->T->T`, then it also holds for `takeM, dropM :: Dur -> Media T -> Media T`:

```
takeM d m :+: dropM d m = m
```

However, this theorem is false; in fact it does not hold for the base case:

```
takeM d (Prim x) :+: dropM d (Prim x)
=  Prim (takeM d x) :+: Prim (dropM d x)
/= Prim x
```

We cannot even state this as a constraint on the base media type, because it involves an interpretation of `(:+:)`. We will return to this issue in a later section.

Example 1 (Music): We declare `Note` to be an instance of `Take`: