Report from Dagstuhl Seminar 16402

Programming Language Techniques for Incremental and Reactive Computing

Edited by

Camil Demetrescu[1], Sebastian Erdweg[2], Matthew A. Hammer[3], and Shriram Krishnamurthi[4]

1 **Sapienza University of Rome, IT,** `demetres@dis.uniroma1.it`
2 **TU Delft, NL,** `s.t.erdweg@tudelft.nl`
3 **University of Colorado – Boulder, US,** `matthew.hammer@colorado.edu`
4 **Brown University – Providence, US,** `sk@cs.brown.edu`

Abstract

Incremental computations are those that process input changes faster than naive computation that runs from scratch, and reactive computations consist of interactive behavior that varies over time. Due to the importance and prevalence of incremental, reactive systems, ad hoc variants of incremental and reactive computation are ubiquitous in modern software systems.

In response to this reality, the PL research community has worked for several decades to advance new languages for systems that interface with a dynamically-changing environment. In this space, researchers propose new general-purpose languages and algorithms to express and implement efficient, dynamic behavior, in the form of incremental and reactive language systems.

While these research lines continue to develop successfully, this work lacks a shared community that synthesizes a collective discussion about common motivations, alternative techniques, current results and future challenges. To overcome this lack of community, this seminar will work towards building one, by strengthening existing research connections and by forging new ones. Developing a shared culture is critical to the future advancement of incremental and reactive computing in modern PL research, and in turn, this PL research is critical to developing the efficient, understandable interactive systems of the future.

Seminar October 3–7, 2016 – http://www.dagstuhl.de/16402
1998 ACM Subject Classification F.3.2 Semantics of Programming Languages, F.3.3 Studies of Program Constructs, F.2.2 Nonnumerical Algorithms and Problems
Keywords and phrases Incremental computing, reactive programming, memoization, change propagation, dynamic dependency graph, dataflow programming, live programming
Digital Object Identifier 10.4230/DagRep.6.10.1

1 Executive Summary

Matthew A. Hammer

We sought to hold a Dagstuhl Seminar that would bring together programming language (PL) researchers focusing on incremental and reactive computing behavior. The meta-level purpose of this seminar was to take an initial step toward developing a community of experts from the disparate threads of successful research. In that this seminar provoked discussion about common and differing motivations, techniques, and future challenges, this event was successful in starting to cultivate this culture.

Short-term concrete outcomes: Thus far, there are been two concrete outcomes of this seminar:

1. *Wikipedia article outlines and edits* (Section 3.3)
2. *First Workshop on Incremental Computation (IC) at PLDI 2017* (Section 5)

Section 3 gives an overview of the event structure of the seminar, and details some of the event's outcomes, including outline brainstorming and Wikipedia editing, and the creation of a new Workshop on Incremental Computing (IC). In later sections, this report gives further background on research in reactive and incremental computing (Section 4), and further details on the new IC Workshop (Section 5).

Acknowledgments. Co-organizing this seminar with Camil, Sebastian and Shriram was a pleasure. I am especially thankful to Shriram for organizing the event's structure, and moderating group discussions and group decision making during its execution. We organizers are all thankful to the participants, who all brought a unique insight to the seminar, which in my humble opinion, succeeded in its aims.

2 Table of Contents

3 Event Summary

The following table summarizes how we organized the three and a half day event. (Monday of that week is a German holiday). Rather than organize the event into talk sessions, we chose more interactive sessions: Posters and introductions (on day 1) and group discussions on the remaining days.

Day 1	Tuesday	*First session*	*Second session*
	AM	Poster session A	Poster session B
	PM	Poster session C	1-min introductions (1 slide each).
Day 2	Wednesday	*First session*	*Second session*
	AM	Demos	Group discussion: IC vs RP
	PM	Group discussion: Domain-specific	Break: Outside walks
Day 3	Thursday	*First session*	*Second session*
	AM	Group discussion: Run-time design	Group discussion: Meshing with non-IC/RP
	PM	Smaller group discussions: Algorithms, Semantics & Types & Verification	
Day 4	Friday	First session	Second session
	AM	Topic outlines, Wikipedia editing: *Incremental_computing* and *Reactive_programming*	

3.1 Poster Sessions

The following seminar participants presented posters about their research (the organizers evenly distributed themselves among these sessions, indicated in bold):

Session 1	**Demetrescu**, Haller, Khoo, Ley-Wild, Minsky, Salvaneschi, Szábo, Tangwonsan
Session 2	Burckhardt, Cicek and Garg, **Erdweg**, **Hammer**, Krishnaswami, Labich, McSherry, Newton, Shah
Session 3	Bhatotia, Courtney, Harkes, **Krishnamurthi**, Mezini, Pouzet, Shapiro

3.2 Group discussions

Before and during the seminar, we took surveys of the participants to find topics that for interesting discussions. In the end, the following topics were scheduled into the event.

- Incremental Computing (IC) vs. Reactive Programming (RP).
  Our first discussion centered on the differences between the domains of incremental and reactive systems, and I have paraphrased some conclusions from that discussion.
  - In common to both IC and RP, we broadly consider ad hoc approaches “harmful”, in the sense that they lack systematic abstractions, and consequently, may suffer from undefined or inconsistent behavior (“glitches”). The alternative to ad hoc approaches are programming language abstractions and carefully-designed libraries that offer reusable abstractions, with well-defined semantics.
  - In common to both IC and RP, there are common abstractions and implementations based on, e.g., dataflow graphs.
  - Reactive programming, unlike IC, encompasses programs whose behavior interacts with other systems in time, and generally does not terminate.

These computations consist of signal processing, aviotics and control systems, OS kernels, and financial analytics. All of these domains require time-dependent behavior that senses time-dependent inputs. In many cases, there may be real-time constraints. To a first approximation, they lack costly, redundant subcomputations that terminate and repeat over time.

- Incremental computation, unlike RP, is concerned with computational cost. It attempts to *improve the algorithmic efficiency* of computations that terminate, but repeat over time in a changing environment.
  The general aim of IC is to improve the asymptotic efficiency of repeated computations, and/or, to cache and reuse large portions of past computations. IC encompasses techniques that make the following tasks more efficient:
  - *data syncrhonization or versioning*, where the redundant subcomputation to avoid is communication of data that has not changed since the last pass;
  - *program analysis*, *HTML rendering* and *spreadsheet evaluation*, where the redundant subcomputation to avoid is the analysis, rendering or calculations that have not been affected by changes since the last pass.

Other discussion topics:

- Domain-specific techniques
- Run-time system design
- Meshing with non-IC/RP
- Semantics & Types: Small, break-out group
- Algorithms: Small, break-out group
- Verification: Small, break-out group

3.3 Topic outlines and Wikipedia edits

During the final morning of the seminar, the seminar participants brainstormed outlines for new Wikipedia articles on the topics of *Incremental computing* and *Reactive programming*. This exercise forced us to catalog the most important concepts in these fields, and how to best structure their exposition. To record these outlines, we edited Wikipedia collaboratively. The seminar participants broke into two groups, to focus on the incremental and reactive articles in a divide-and-conquer approach.

The outcome of this editing session are new article outlines, with some discussion, and some links to relevant literature. For both articles, much more work is warranted. However, we feel that the consensuses reached during the seminar will make future collaborative editing easier at a distance. (e.g., by using the Dagstuhl Seminar email list, or Wikipedia itself, to coordinate).

A recording of the new proposed outlines is included below, for posterity. (Of course, we give permission to Wikipedia to use this outline; in fact, we encourage them to continue to do so!). Below the outlines, we give links to the exact Wikipedia edits.

3.3.1 Article outline: Incremental Computing

- Static Versus Dynamic
- Specialized versus General-Purpose Approaches

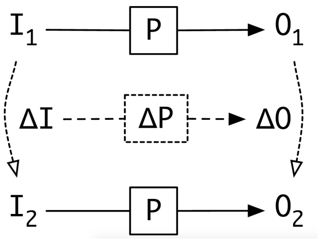

Figure 1 Incremental computing of P using ΔP is sound when it is commutative, as above. This diagram shows the relationships between a program P, successive inputs I_1 and I_2, successive outputs O_1 and O_2, their relative changes Δ_I and Δ_O, and the change propagation mechanism that performs Δ_O somehow from Δ_I.

- Static Methods
  - Program Derivatives
  - View Maintenance
- Dynamic Methods
- Existing systems
  - Compiler and Language Support
  - Frameworks and libraries
- Applications
  - Databases (view maintenance)
  - Build systems
  - Spreadsheets
  - Development Environments
  - Financial Computations
  - Attribute Grammar Evaluation
  - Graph Computations and Queries
  - GUIs (e.g., React and DOM diffing)
  - Scientific applications
- See also
  - Reactive programming
  - Memoization

We also contributed Figure 1, which we used throughout the seminar to orient our group discussions.

3.3.2 Article outline: Reactive programming

- Definition of Reactive Programming
- Approaches to Creating Reactive Programming Languages
  - Dedicated languages that are specific to some domain constraints (such as real-time or embedded computing or hardware description)
  - General-purpose languages that support reactivity
  - Libraries or embedded domain specific languages that enable reactivity alongside or on top of an existing general-purpose programming language
- Programming Models and Semantics
  - Synchrony: is the underlying model of time synchronous versus asynchronous?
  - Determinism: Deterministic versus non-deterministic in both evaluation process and results (the former does not necessarily imply the latter)
  - Update process: callbacks versus dataflow versus actors

- Implementation Challenges
  - Glitches
  - Cyclic Dependencies
  - Interaction with Mutable State
  - Dynamic Updating of the Graph of Dependencies

3.3.3 Wikipedia edits

Our collective edits, relative to the article before our collaborative editing session, are visible at the following Wikipedia URLs:

- **Incremental computing edits:**
  https://en.wikipedia.org/w/index.php?title=Incremental_computing&type=revision&diff=743022258&oldid=735698349
- **Reactive programming edits:**
  https://en.wikipedia.org/w/index.php?title=Reactive_programming&type=revision&diff=743074812&oldid=740655985

3.4 New workshop on incremental computation

Following the success of this Dagstuhl Seminar, organizers Sebastian Erdweg and Matthew Hammer proposed a new Workshop on Incremental Computing (IC) to PLDI 2017, in Barcelona, Spain. The content of this workshop proposal is included below, in Section 5. PLDI 2017 has since accepted this proposal.

4 Background

4.1 Incremental computing

Incremental computations are those that process input changes faster than a naive re-computation from scratch. Due to the importance and prevalence of incremental, reactive systems, ad hoc variants of incremental computation are ubiquitous in modern software systems. As an everyday example, spreadsheets such as Excel re-calculate selectively based on user interaction and the dynamic dependencies of formula. Incremental computation is needed in this domain, and many others, for efficiency and responsiveness. As other examples, build systems and integrated development environments re-compile selectively, based on the code dependencies. Hence, build systems strive to use a domain-specific form of incremental computation that is aware of compiler dependencies [14, 27]. Doing so is necessary to support interactive development, testing and debugging, which should be responsive to code changes. Meanwhile, the interactive behavior of the visual elements in the development tools, and their interaction with external tools, can be modeled with reactive computation. Finally, modern web browsers (such as Chrome, Firefox, Safari, etc.) house incremental computations that respond to mobile code, whose execution leads to re-computing layout and styling information (viz., dynamic variation of CSS attributes incrementally affect the placement and appearance of modern web pages). These scripts, as well as the browser platform in which they run, are incremental, reactive computations [6].

Due to their prevalence in practical systems used every day, notions of incremental computing abound in computer science broadly, and within research on programming languages (PL). In the area of PL, researchers are particularly interested in **language-based approaches** to incremental computation. In contrast to the algorithms community that often studies each incremental problem in isolation (e.g., incremental convex hull), PL researchers study large classes of incremental programs that are defined by a general language. Their typical goal is to provide a language and associated technique that is general enough to express the behavior of many incremental or reactive programs. For instance, many general-purpose techniques can derive the incremental behavior of two-dimensional convex hull from the expression of a textbook algorithm for the non-incremental algorithm (e.g., quickhull) [1, 16, 5, 19]. More generally, researchers have shown that for certain algorithms, inputs, and classes of input changes, IC delivers large, even *asymptotic* speed-ups over full reevaluation [4, 2]. IC has been developed in many different language settings [26, 17, 18, 9], and has even been used to address open problems, e.g., in computational geometry [3].

4.2 Reactive programming

Reactive programming languages offer abstractions for processing events generated by dynamic environments. Reactive abstractions encompass both event-driven and data-driven scenarios, relying on graphs to model dependencies in a program. In the former, events are explicitly modeled as a stream generated over time; computations respond to generated events and may trigger further computations along the dependency graph. In the latter, events are modeled as input data changes as in IC frameworks. A data-flow graph describes relationships between objects and changes are automatically propagated throughout the graph.

Similarly to IC environments, a critical aspect in reactive systems is to minimize the amount of recomputations triggered by external discrete events or input data changes. Early examples of event-based reactive environments for real-time systems in embedded software include Signal [15] and Lustre [8].

Functional Reactive Programming (FRP) is a declarative programming model for constructing interactive applications [13, 24, 28]. The chief aim of FRP is to provide a declarative means of specifying programs whose values are time-dependent (stored in signals), whereas the chief aim of IC is to provide time savings for small input changes (stored in special references). The different scope and programming model of FRP makes it hard to imagine using it to write an efficient incremental sorting algorithm, though it may be possible. On the other hand, IC would seem to be an appropriate mechanism for implementing an FRP engine, though the exact nature of this connection remains unclear.

FrTime [10] extends a purely functional subset of PLT Scheme with an instantiation of the FRP paradigm, supporting eager evaluation and *benign impurities* (e.g., imperative commands for drawing and for creating and varying mutable references). The problem of integrating FrTime and object-oriented graphics toolkits has been investigated by [20].

More recently [23] have introduced Flapjax, a reactive extension to JavaScript for Web applications, whose approach is mainly informed by FrTime. Frappé [11] integrates the FRP model with the Java Beans technology, allowing reactive programming in Java. FrTime has also served as a basis for MzTake [22], a scriptable debugger implementing a dataflow language in the tradition of Dalek [25], an earlier programmable debugger that also modeled events using a data-flow graph. SugarCubes [7] and ReactiveML [21] allow reactive programming in Java and OCaml, respectively.

A different data-driven line of research investigates how to mix the reactive paradigm with imperative and object-oriented languages, allowing programmers to declaratively express dataflow constraints between C/C++ objects allocated in a special “reactive memory” heap [12].

5 Event Outcome: Workshop on Incremental Computation (IC)

Sebastian Erdweg (TU Delft, NL) and Matthew A. Hammer (University of Colorado – Boulder, US)

The content below is from a successful workshop proposal to PLDI 2017. Elsewhere, this proposal referenced the success of this Dagstuhl Seminar as evidence of research community interest.

Due to its cross-cutting nature, research results on and experience with incremental computing is scattered throughout the PL community. The Workshop on Incremental Computing (IC) will provide a platform for researchers and users of incremental computing.

- 4 sessions featuring invited talks from academia and industry as well as contributed talks from the community.
- Type of submission: Contributed talks need to submit talk abstracts, which forms the basis for selection.
- Review process: We will invite a small program committee that selects contributed talks based on the submitted talk abstracts.
- Result dissemination: We will collect talk abstracts from all invited and selected contributed talks and publish them as an openly accessable technical report.

Since incremental computations are cross-cutting PL, we expect significant interest within the PLDI community. Traditionally, static analysis has seen numerous successful applications of incremental computing, and the workshop may spark especial interest in that part of the community. Since this is the first workshop on incremental computing, it is very difficult to estimate the number of attendees; anything between 20 and 80 people seems realistic.

References

1 Umut A. Acar, Guy E. Blelloch, Matthias Blume, Robert Harper, and Kanat Tangwongsan. A library for self-adjusting computation. *ENTCS*, 148(2), 2006.
2 Umut A. Acar, Guy E. Blelloch, Kanat Tangwongsan, and Duru Türkoğlu. Robust kinetic convex hulls in 3D. In *Proceedings of the 16th Annual European Symposium on Algorithms*, September 2008.
3 Umut A. Acar, Andrew Cotter, Benoît Hudson, and Duru Türkoğlu. Dynamic well-spaced point sets. In *Symposium on Computational Geometry*, 2010.
4 Umut A. Acar, Alexander Ihler, Ramgopal Mettu, and Özgür Sümer. Adaptive Bayesian inference. In *Neural Information Processing Systems (NIPS)*, 2007.
5 Umut A. Acar and Ruy Ley-Wild. Self-adjusting computation with Delta ML. In *Advanced Functional Programming*. Springer Berlin Heidelberg, 2009.
6 Brian Anderson, Lars Bergstrom, David Herman, Josh Matthews, Keegan McAllister, Manish Goregaokar, Jack Moffitt, and Simon Sapin. Experience report: Developing

the servo web browser engine using rust. *CoRR*, abs/1505.07383, 2015. URL: http://arxiv.org/abs/1505.07383.

7 Frédéric Boussinot and Jean-Ferdy Susini. The SugarCubes Tool Box: a Reactive Java Framework. *Software: Practice and Experience*, 28(14):1531–1550, 1998.

8 P. Caspi, P. Pilaud, N. Halbwachs, and J. Plaice. Lustre, a Declarative Language for Programming Synchronous Systems. In *POPL*, pages 178–188, 1987.

9 Yan Chen, Joshua Dunfield, Matthew A. Hammer, and Umut A. Acar. Implicit self-adjusting computation for purely functional programs. *J. Functional Programming*, 24(1):56–112, 2014.

10 Gregory H. Cooper and Shriram Krishnamurthi. Embedding dynamic dataflow in a call-by-value language. In *ESOP*, 2006.

11 Antony Courtney. Frappé: Functional Reactive Programming in Java. In *PADL*, pages 29–44, 2001.

12 Camil Demetrescu, Irene Finocchi, and Andrea Ribichini. Reactive imperative programming with dataflow constraints. *ACM Trans. Program. Lang. Syst.*, 37(1):3:1–3:53, 2014.

13 Conal Elliott and Paul Hudak. Functional Reactive Animation. In *ICFP*, pages 263–273, 1997.

14 Sebastian Erdweg, Moritz Lichter, and Manuel Weiel. A sound and optimal incremental build system with dynamic dependencies. In *OOPSLA'15*, pages 89–106. ACM, 2015.

15 P. Le Guernic, A. Benveniste, P. Bournai, and T. Gautier. SIGNAL – A Data Flow-Oriented Language for Signal Processing. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 34(2):362–374, 1986.

16 Matthew Hammer and Umut A. Acar. Memory management for self-adjusting computation. In *ISMM*, 2008.

17 Matthew Hammer, Umut A. Acar, Mohan Rajagopalan, and Anwar Ghuloum. A proposal for parallel self-adjusting computation. In *DAMP'07: Declarative Aspects of Multicore Programming*, 2007.

18 Matthew A. Hammer, Umut A. Acar, and Yan Chen. CEAL: a C-based language for self-adjusting computation. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2009.

19 Matthew A. Hammer, Joshua Dunfield, Kyle Headley, Nicholas Labich, Jeffrey S. Foster, Michael Hicks, and David Van Horn. Incremental computation with names (extended version). arXiv:`1503.07792 [cs.PL]`, 2015.

20 Daniel Ignatoff, Gregory H. Cooper, and Shriram Krishnamurthi. Crossing State Lines: Adapting Object-Oriented Frameworks to Functional Reactive Languages. In *FLOPS*, pages 259–276, 2006.

21 Louis Mandel and Marc Pouzet. ReactiveML, a Reactive Extension to ML. In *PPDP*, pages 82–93, 2005.

22 Guillaume Marceau, Gregory H. Cooper, Jonathan P. Spiro, Shriram Krishnamurthi, and Steven P. Reiss. The design and implementation of a dataflow language for scriptable debugging. *Automated Software Engg.*, 14(1):59–86, 2007.

23 Leo A. Meyerovich, Arjun Guha, Jacob Baskin, Gregory H. Cooper, Michael Greenberg, Aleks Bromfield, and Shriram Krishnamurthi. Flapjax: a Programming Language for Ajax Applications. In *OOPSLA*, pages 1–20, 2009.

24 Henrik Nilsson, Antony Courtney, and John Peterson. Functional reactive programming, continued. In *Proceedings of the 2002 ACM SIGPLAN Haskell Workshop (Haskell'02)*, pages 51–64, Pittsburgh, Pennsylvania, USA, October 2002. ACM Press.

25 Ronald A. Olsson, Richard H. Crawford, and W. Wilson Ho. A dataflow approach to event-based debugging. *Softw. Pract. Exper.*, 21(2):209–229, 1991.

26 Ajeet Shankar and Rastislav Bodik. DITTO: Automatic incrementalization of data structure invariant checks (in Java). In *Programming Language Design and Implementation*, 2007.

27 Tamás Szabó, Sebastian Erdweg, and Markus Völter. IncA: A DSL for the definition of incremental program analyses. In *Proceedings of International Conference on Automated Software Engineering (ASE)*. ACM, 2016.

28 Zhanyong Wan and Paul Hudak. Functional Reactive Programming from First Principles. In *PLDI*, pages 242–252, 2000.

Participants

- Pramod Bhatotia
TU Dresden, DE
- Sebastian Burckhardt
Microsoft Research – Redmond, US
- Ezgi Cicek
MPI-SWS – Saarbrücken, DE
- Antony Courtney
San Francisco, US
- Camil Demetrescu
Sapienza University of Rome, IT
- Sebastian Erdweg
TU Delft, NL
- Deepak Garg
MPI-SWS – Saarbrücken, DE
- Philipp Haller
KTH Royal Institute of Technology – Stockholm, SE
- Matthew A. Hammer
University of Colorado – Boulder, US
- Daco Harkes
TU Delft, NL
- Kyle Headley
University of Colorado – Boulder, US
- Yit Phang Khoo
The MathWorks Inc. – Natick, US
- Shriram Krishnamurthi
Brown University – Providence, US
- Neel Krishnaswami
University of Cambridge, GB
- Nicholas Labich
University of Maryland – College Park, US
- Ruy Ley-Wild
LogicBlox – Atlanta, US
- Frank McSherry
Richmond, US
- Mira Mezini
TU Darmstadt, DE
- Yaron Minsky
Jane Street – New York, US
- Ryan R. Newton
Indiana University – Bloomington, US
- Marc Pouzet
ENS – Paris, FR
- Guido Salvaneschi
TU Darmstadt, DE
- Rohin Shah
University of California – Berkeley, US
- R. Benjamin Shapiro
University of Colorado – Boulder, US
- Tamás Szabó
TU Delft, NL
- Kanat Tangwongsan
Mahidol University, TH