Licentiate Thesis Proposal

A resource-efficient event detection algebra

Jan Carlson
Department of Computer Science and Engineering
Mälardalen University, Sweden
jan.carlson@mdh.se

Abstract

Event detection is an important aspect of many application types, ranging from active databases over digital libraries and stock market agents, to reactive embedded systems. To allow systems to react to complex events patterns rather than to simple primitive events, an event algebra can be used.

We have developed an event algebra with formal semantics and a number of useful algebraic properties. An equivalent operational semantics has been developed, and we have identified an important subset of event expressions for which the detection mechanism can be implemented with constant memory.

1 Background and motivation

A wide range of applications, including active databases, traffic monitoring systems and rule based embedded systems, are based on the detection of events that trigger an appropriate response from the system. Events can be simple, e.g., sampled directly from the environment or occurring within the system, but it is often necessary to react to more sophisticated situations involving a number of simpler events that occur in accordance with some pattern.

A standard way in which to allow systems to react to sophisticated situations is to introduce complex events by means of an event algebra. These complex events can then be used to trigger actions just like simple events. A benefit of this method is that the mechanisms handling event detection are separated from the rest of the system logic.

In order to allow formal reasoning about the behaviour of the system, it is essential that the algebra has a well-defined semantics. This is particularly important when the algebra is used in safety-critical applications for which formal verification is required. In addition, reasoning on a high level of abstraction is facilitated if the designer is provided a number of formal properties that the algebra conforms to. Such properties include, for example, laws of associativity and distributivity.

For embedded applications and systems with strict timeliness requirements, it is essential that the event detection can be implemented with limited resources. For a given complex event, defined in the algebra, one would like to know the maximum memory usage, or at least a safe approximation thereof, as well as the worst case execution time of the detection mechanism for that event.

2 Event Algebras

Complex events are defined by expressions built from primitive events and the operators of the event algebra. The task of the event detection mechanism is to compute from the instances of primitive events, the instances of events defined by expressions.

In some applications the event detection is performed on a finite set of primitive event instances that were collected earlier by monitoring the environment or the system. This allows the detection mechanism to process the data in any order and possibly in several passes, and typically do not impose hard resource constraints. Contrasting these *off-line* methods, other applications require events to be detected continually during the entire system lifetime. This implies that the detection mechanism have no knowledge of future instances of primitive events, and typically due to resource restrictions, only limited information about past events can be stored. This type of event detection can be classified as *on-line*, and is the main focus of this work.

The following operations, or variants of them, are found in many event algebras. The *disjunction* of A and B represents that either of A and B occurs, here denoted $A \vee B$. *Conjunction* means that both events have occurred, possibly not simultaneously, and is denoted $A + B$. The *negation*, denoted $A - B$, occurs when there is an occurrence of A during which there is no occurrence of B. Finally, a *sequence* of A and B is an occurrence of A followed by an occurrence of B, and is denoted $A; B$.

2.1 Interval-based event detection

Single point detection means that every complex event, including those that require more than one occurrence of simpler events in order to occur, is associated with a single time point (the time of detection, i.e., the time of the last occurrence that was required). Galton and Augusto [GA02] showed that this results in unintended semantics for some operation compositions.

For example, using single point detection an instance of the event $B; (A; C)$ is detected if A occurs first, and then B followed by C. The reason is that these occurrences cause a detection of $A; C$ which is associated with the occurrence time of C. Since B occurred before this time point, an occurrence of $B; (A; C)$ is detected. Figure 1 shows this situation, together with the intuitively correct detection of $A; (B; C)$. In this diagram, time flows from left to right, each row showing the detected instances of the corresponding expression.

This problem can be solved by associating the occurrence of a complex event with the occurrence interval, i.e., the interval in which all required simpler events occurred, rather than the time of detection. In this setting, the sequence $A; B$ can be defined to occur only if the intervals of A and B are non-overlapping. In our example, no occurrence of $B; (A; C)$ is detected, since B occurs within the interval associated with the occurrence of $A; C$. The result of the interval-based version is depicted in Figure 2.

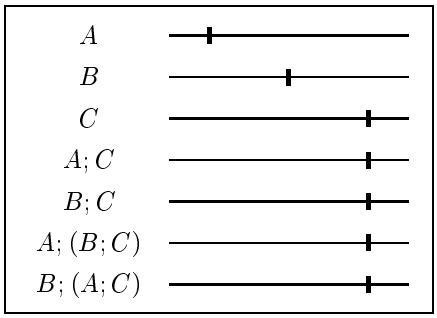

Figure 1: Single point semantics

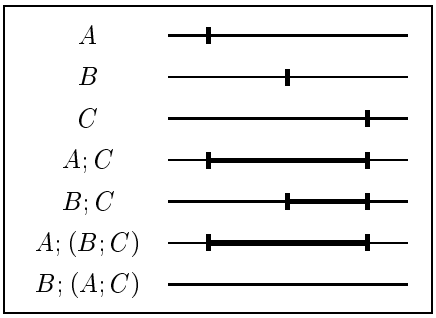

Figure 2: Interval semantics

2.2 Restricted detection

A very straightforward definition of the sequence operator is that the sequence $A; B$ should occur whenever A occurs and then B occurs. Using this definition, three occurrences of A

followed by two occurrences of B would generate six occurrences of the sequence. While this may be acceptable, or even desirable, in some applications, the memory requirements (each occurrence of A must be remembered forever) and the increasing number of simultaneous events means that it is unsuitable in many cases. Also, it is argued that many applications are interested only in a subset of the instances that are generated by this definition.

One way to deal with this is to define the event algebra in two steps. The operations are defined in an unrestricted, straightforward way like in our example above. Then a restriction policy is defined, that acts like a filter so that only a subset of the occurrences allowed by the unrestricted definition are detected. For example, a restriction policy could state that only the latest occurrence of A is allowed to create an occurrence of $A; B$ when B occurs, and that once an instance of A is used it can not be used again in the future.

The restriction policy is sometimes divided into two separate concepts. An instance selection policy decides which of multiple possible instances to use when constructing a complex instance, and the consumption policy states whether the same instance can be used again the in future.

3 Related work

A lot of work, especially formal approaches, on event algebras has been done in the context of active databases. In addition, work in knowledge representation and general event notification is also of relevance.

3.1 Active databases

One area where event algebras are used is active databases which, unlike passive databases, react automatically to situations that arise within or outside the database. The reactions are specified by so called *event-condition-action rules* (ECA rules) stating that when a certain event occurs, and the condition is satisfied, the given action should be performed. The event part of a ECA rule can be expressed by an event algebra to allow the database to react to complex events.

The event expression language uses in the object database Ode has the same expressive power as regular expressions, which allows the detection mechanism to be implemented by finite state automata [GJS93]. The definition is based on a global, totally ordered set of primitive event occurrences, implying that primitive events can not occur simultaneously. To allow event occurrences to carry values and composite events that occur only under given restrictions on the values of the constituent events, the automata mechanism is extended with data structures that store the values of events that have occurred.

In the active database SAMOS, event detection is implemented using Petri nets [GD93, GD94]. Event occurrences are associated with a number of parameter-value pairs, and it can be specified that a complex event should occur only if the constituent event occurrences have the same value for a given parameter. SAMOS does not allow simultaneous primitive event occurrences.

Snoop [CM94, CKAK94] is an event specification language for active databases. It defines four different restriction policies (called parameter contexts) that can be applied to the operators of the algebra. The unrestricted context is defined formally, but for the restriction policies only informal descriptions are given. The detection mechanism is based on trees corresponding to the event expressions, where primitive event occurrences are inserted at the leaves and propagate upwards in the tree as they cause more complex events to occur.

None of the algebras described above provide algebraic properties for their respective operators, and little is said about the memory and time complexity associated with the detection of complex events.

A formalized schema for this type of event detection, including a definition of the operations and restriction policies of Snoop using this schema, has been defined by Mellin and Andler [MA02]. The operators have definitions parameterised on restriction policies, which facilitates formal reasoning about the operators with different restriction policies applied, without requiring explicit definitions for each operator- restriction combination. They propose, as future work, to extend the operators with temporal constraints, which would allow an investigation of the temporal complexity of the detection algorithm.

Zimmer and Unland present a formal restriction framework in which the event algebras of Snoop, SAMOS, Ode and a few other systems are compared [ZU99]. They also highlight a number of ambiguities and inconsistencies of the various approaches.

Liu et al. uses Real Time Logic to define a system where composite events are expressed as timing constraints and handled by general timing constraint monitoring techniques [LMK98]. They present a mechanism for early detection of timing constraint violation, and show that it is possible to calculate a upper bound on the length of the structures needed to detect an event. In general, the time complexity of detection is in $O(n^3)$, but for a certain subset of expressions, a $O(n)$ algorithm is possible [ML97a, ML97b].

The algebra presented by Baily and Mikulás [BM01], including four restriction policies, is defined formally in temporal logic. They identify a class of complex events for which testing whether two complex events are equivalent is decidable, and show that testing for implication is undecidable.

3.2 Event notification systems

Many systems and frameworks have been developed where clients register their interest in certain types of events with a central server. The server monitors the environment and, upon the detection of an event, notifies the concerned clients. As the clients typically perform monitoring tasks, they are generally interested in particular event sequences rather than single occurrences. Rather that having each application implement this separately, it is beneficial to extend the server to allow registration of complex event descriptions. The system presented by Hayton et al. [HBBM96] contains a simple event algebra, implemented using a push down automata.

The event algebra developed by Hinze and Voisard is designed to suit event notification service systems in general [HV02b, HV02a]. Their algebra contains time restricted sequence and conjunction, which permits events like A *occurs less than* t *time units before* B to be expressed. Following the framework presented by Zimmer and Unland [ZU99], the algebra is parameterised with respect to policies for event instance selection and consumption.

Additional examples of event notification systems that allow composite events expressed by an event algebra are the READY system [GKP99] and the event specification language developed by Zang and Unger [ZU96].

3.3 Knowledge representation

In the area of knowledge representation, similar techniques are used to reason about event occurrences. Interval Calculus introduce formalised concepts for properties, actions and events, where events are expressed in terms of conditions for their occurrence [AF94]. Event Calculus [KS86, Kow92] also deals with the occurrences of events, but, as in the Interval Calculus, the motivation is slightly different from ours. Rather than detecting complex events as they occur, the focus of Event Calculus is to express formally the fact that some event has occurred, and to allow inferences to be made from it.

In the area of temporal data mining, event operators similar to those in event detection are used. A crucial difference is that the task of event detection is to detect patterns that match a given event description, while in data mining the data is analysed in search for trends and patterns for which matching descriptions are to be derived [AO01].

4 Research description

We have developed an event algebra suitable for applications where the ability to reason formally about the system is required, and where resources such as memory and time are limited. The following outlines the proposed approach and the main contributions.

4.1 The approach

In order to allow formal reasoning we believe that a fully formal definition of the algebra is essential. We use techniques such as interval-based semantics and restriction policies to handle complexity issues while retaining an intuitive semantics for the operators.

A key factor is the development of a suitable restriction policy. It should be restrictive enough to permit the formulation of memory consumption bounds, while at the same time allowing a simple formal relation between the restricted semantics and the unrestricted version. Since these two objectives are contradictory to some extent, careful investigation is required to find an optimal balance.

Once the declarative semantics was completed, we developed an operational semantics for the algebra in order to reason about resource requirements. The operational semantics will be used to implement the algebra in some suitable language used for embedded system design.

4.2 The contributions

The main contributions of the work can be summarised as follows.

The algebra The semantics of the event algebra, including the restriction policy, is defined formally. This is a basic requirement in order to perform any formal reasoning about the algebra or a system that uses utilises it.

Our restriction policy is carefully designed to retain many of the properties of operators that are intuitive. These include, for example, associativity of sequence, conjunction and disjunction; commutativity of conjunction and disjunction; and distributative laws. Additionally, we have investigated the relation between the restricted and the unrestricted semantics. This facilitates formal reasoning since it allows, for example, properties to be proved for the simple, unrestricted semantics and then translated into the restricted version using this relation.

Operational semantics An imperative algorithm has been developed, and we will prove that it is equivalent to the declarative semantics of the algebra. This algorithm is analysed with respect to resource requirements, and criteria under which it can be guaranteed to execute with bounded resources will be given.

Implementation The J2ME Personal Profile implementation illustrates how the algebra can be used together with existing languages for embedded systems.

4.3 Published and planned papers

A paper describing the declarative semantics of the algebra, together with many algebraic properties and an imperative algorithm, was presented at the first international workshop on formal modeling and analysis of timed systems (FORMATS 2003) [CL03].

In a second paper, we intend to extend the algebra with time limited operators. It will also contain correctness proofs for the algorithm, additional properites of the operators, and a description of a class of expressions that can be detected by a constant memory implementation.

Additionally, we plan to write a paper that focuses on the Java (or J2ME Personal Profile) implementation. Whether this is done before the licentiate thesis is finished, or afterwards, depends on the time it takes to finish the other activities.

5 Thesis outline

The proposed title of the thesis is *"A resource-efficient event detection algebra"*. This section presents the planned table of contents.

1 Introduction

This section addresses the relevance of the work, and states the key objectives. Further, it describes the approach by which the objectives have been investigated, and summarises the main contributions of the work.

2 Composite event detection

This section introduces some basic techniques used in event algebras.

3 The event algebra

3.1 Declarative semantics

This section defines the algebra.

3.2 Algebraic properties

Properties that relate the restricted algebra to the unrestricted version are presented, as well as equality properties of expressions.

3.3 Operational semantics

This section presents an algorithm, and proves that it is equivalent to the declarative semantics.

3.4 Resource requirements

The algorithm is analysed with respect to memory requirements and time, and a class of event expressions is identified for which the algorithm requires constant memory.

4 J2ME Personal Profile implementation

This section describes an implementation of the algebra for *J2ME Personal Profile*, a Java application environment for embedded devices.

5 Related work

The section gives an overview of related work, and investigates similarities and differences to the proposed algebra.

6 Discussion and future work

The main contributions are summarised and the possible directions of future research are outlined.

6 Time plan

This section describe the milestones of the licentiate thesis project, and when they are planned. The declarative and operational semantics of the program, as well as a majority of the property proofs, are completed [CL03]. The licentiate degree requirement of passed courses of at least 30 credits is already fulfilled.

2003-11-14	Licentiate proposal presented
2003-11-21	Algorithm proven correct and a class of constant expressions defined
2003-12-04	Second paper finished
2004-01-23	J2ME implementation finished
2004-02-06	Experiments finished
2004-02-27	Third paper finished (might be postponed until after the thesis defense)
2004-03-19	Thesis draft ready for review
2004-04-30	Thesis finished and defended

7 Future work

The main weakness of the developed algebra is the memory requirements. The detection can be performed with constant memory and time for a large subset of events expressions. For expressions outside this set, it is often possible to derive memory bounds from information about the frequencies of primitive events. This idea should be formalised and investigated further. Additional operators might be introduced to increase the expressiveness of the algebra. Possible extensions include simultaneous conjunction, periodic events, etc.

In this work we have not paid much attention to the values associated with the primitive occurrences. Values of primitive event occurrences are composed and propagated as values of complex events. In some systems it might be desirable to filter the event streams with respect to the values. Performing filtering on the primitive streams before they are handled by the algebra is not always an option, for example if we want to detect occurrences of $A; B$ such that the value of the two instances are the same. When filtering is performed on the result of the proposed algebra, some of the properties do not hold after filtering. It might be possible to define filtering options that can be performed on subexpressions within an event expression, while still retaining the algebraic properties.

The event detection algorithm has been implemented in Java, and we would like to incorporate it in additional languages. The functional reactive language suite AFRP, including the robot programming language FROB [PNH02], is based on time varying behaviours and discrete events [WH00, NCP02]. We belive that an efficient implementation of the event algebra in AFRP would facilitate development as well as formal reasoning about AFRP programs.

Timber is an object-oriented reactive language with a purely functional core, designed to target embedded systems in particular [CNK03]. It should be reasonably straightforward to develope a Timber implementation of the algebra from the current, object-oriented, Java implementation.

Another possibility is to combine the algebra with the synchronous language Esterel. Esterel is built around a notion of discrete time intervals during which a number of events can occur that should be reacted to [Ber99, Ber00]. Incorporating the event algebra so that the Esterel program reacts to occurrences of complex events rather than primitive would separate pure event detection from the rest of the application logic, which would hopefully increase readability and provability of programs.

References

[AF94] J. F. Allen and G. Ferguson. Actions and events in interval temporal logic. *Journal of Logic and Computation*, 4(5):531–579, October 1994.

[AO01] C. M. Antunes and A. L. Oliveira. Temporal data mining: An overview. In *KDD Workshop on Temporal Data Mining*, pages 1–13, San Francisco, CA, 26 August 2001.

[Ber99] G. Berry. *The Esterel-V5 Language Primer*. CMA and Inria, Sophia-Antipolis, France, v 5.21, release 2.0 edition, May 1999.

[Ber00] G. Berry. The foundations of esterel. In G. Plotkin, C. Stirling, and M. Tofte, editors, *Proof, Language, and Interaction: Essays in Honour of Robin Milner*, pages 425–454. MIT Press, 2000.

[BM01] J. Bailey and S. Mikulás. Expressiveness issues and decision problems for active database event queries. In *Database Theory - ICDT 2001, 8th International Conference*, volume 1973 of *Lecture Notes in Computer Science*, pages 68–82, London, UK, 4–6 January 2001. Springer.

[CKAK94] S. Chakravarthy, V. Krishnaprasad, E. Anwar, and S.-K. Kim. Composite events for active databases: Semantics, contexts and detection. In *20th International Conference on Very Large Data Bases*, pages 606–617, Santiago, Chile, 12–15 September 1994. Morgan Kaufmann Publishers.

[CL03] J. Carlson and B. Lisper. An interval-based algebra for restricted event detection. In *Proceedings of First International Workshop on Formal Modeling and Analysis of Timed Systems (FORMATS 2003)*, Marseille, France, 6–7 September 2003.

[CM94] S. Chakravarthy and D. Mishra. Snoop: An expressive event specification language for active databases. *Data Knowledge Engineering*, 14(1):1–26, 1994.

[CNK03] M. Carlsson, J. Nordlander, and D. Kieburtz. The semantic layers of Timber. In *Proceedings of the First Asian Symposium on Programming Languages and Systems (APLAS'2003)*, Beijing, China, 26–29 November 2003. To appear.

[GA02] A. Galton and J. C. Augusto. Two approaches to event definition. In *Proc. of Database and Expert Systems Applications 13th International Conference (DEXA'02)*, volume 2453 of *Lecture Notes in Computer Science*, pages 547–556, Aix-en-Provence, France, 2–6 September 2002. Springer-Verlag.

[GD93] S. Gatziu and K. R. Dittrich. Events in an active object-oriented database system. In *Proc. 1st Intl. Workshop on Rules in Database Systems (RIDS)*, Edinburgh, UK, September 1993. Springer-Verlag.

[GD94] S. Gatziu and K. R. Dittrich. Detecting composite events in active database systems using petri nets. In *Research Issues in Data Engineering (RIDE '94)*, pages 2–9, Los Alamitos, Ca., USA, February 1994. IEEE Computer Society Press.

[GJS93] N. Gehani, H. V. Jagadish, and O. Shmueli. COMPOSE: A system for composite specification and detection. In *Advanced Database Systems*, volume 759 of *Lecture Notes in Computer Science*. Springer, 1993.

[GKP99] R.E. Gruber, B. Krishnamurthy, and E. Panagos. The architecture of the READY event notification service. In *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems, Middleware Workshop*, Austin, TX, USA, May 1999.

[HBBM96] R. Hayton, J. Bacon, J. Bates, and K. Moody. Using events to build large scale distributed applications. In *Proceedings of the ACM SIGOPS European Workshop*, 1996.

[HV02a] A. Hinze and A. Voisard. Composite events in notification services with application to logistics support. Technical Report tr-B-02-10, Freie Universitaet Berlin, 2002.

[HV02b] A. Hinze and A. Voisard. A parameterized algebra for event notification services. In *Proceedings of the 9th International Symposium on Temporal Representation and Reasoning (TIME 2002)*, Manchester, UK, July 2002. Springer-Verlag.

[Kow92] R. Kowalski. Database updates in the event calculus. *The Journal of Logic Programming*, 12:121, January 1992.

[KS86] R. A. Kowalski and M. J. Sergot. A logic-based calculus of events. *New Generation Computing*, 4:67–95, 1986.

[LMK98] G. Liu, A. Mok, and P. Konana. A unified approach for specifying timing constraints and composite events in active real-time database systems. In *4th IEEE Real-Time Technology and Applications Symposium (RTAS '98)*, pages 199–209, Washington - Brussels - Tokyo, June 1998. IEEE.

[MA02] J. Mellin and S. F. Adler. A formalized schema for event composition. In *Proc. 8th Int. Conf on Real-Time Computing Systems and Applications (RTCSA 2002)*, pages 201–210, Tokyo, Japan, 18–20 March 2002.

[ML97a] A. Mok and G. Liu. Early detection of timing constraint violation at runtime. In *The 18th IEEE Real-Time Systems Symposium (RTSS '97)*, pages 176–186, Washington - Brussels - Tokyo, December 1997. IEEE.

[ML97b] A. Mok and G. Liu. Efficient run-time monitoring of timing constraints. In *Proceedings of the Third IEEE Real-Time Technology and Applications Symposium (RTAS '97)*, pages 252–262, Washington - Brussels - Tokyo, June 1997. IEEE.

[NCP02] H. Nilsson, A. Courtney, and J. Peterson. Functional reactive programming, continued. In *Proceedings of the 2002 ACM SIGPLAN Haskell Workshop (HASKELL-02)*, pages 51–64, New York, October 3 2002. ACM Press.

[PNH02] I. Pembeci, H. Nilsson, and G. Hager. Functional reactive robotics: an exercise in principled integration of domain-specific languages. In *Proc. 4th ACM SIGPLAN Conference on Principles and Practice of Declarative Programming (PPDP-02)*, pages 168–179. ACM Press, October 6–8 2002.

[WH00] Z. Wan and P. Hudak. Functional reactive programming from first principles. *ACM SIGPLAN Notices*, 35(5):242–252, May 2000.

[ZU96] R. Zhang and E. Unger. Event specification and detection. Technical Report TR CS-96-8, Department of Computing and Information Sciences, Kansas State University, June 1996.

[ZU99] D. Zimmer and R. Unland. On the semantics of complex events in active database management systems. In *Proceedings of the 15th International Conference on Data Engineering*, pages 392–399. IEEE Computer Society Press, 1999.