An improved algebra for restricted event detection

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

MDH-MRTC-???/2004-?-SE

February, 2004

Abstract

This paper extends the event detection algebra proposed in our previous work with a temporally restricted sequence that allows a large class of expressions to be detected with limited resources.

1 Introduction

In reactive systems, execution is driven by external events to which the system should respond with appropriate actions. Such events can be simple, but systems are often supposed to react to sophisticated situations involving a number of simpler events occurring in accordance with some pattern. A systematic approach to handle this type of systems is to separate the mechanism for detecting composite events from the rest of the application logic.

This paper extends the event detection algebra presented in [3] with a temporally restricted sequence that allows a large class of expressions to be detected with limited resources. We also give a formal proof that establishes the relation between the declarative and operational semantics.

The rest of this paper is organised as follows: Section 2 gives a bried survey of related work. The algebra is defined in Section 3, followed by a presentation of the algebraic properties in Section 4. Section 5 presents the algorithm, and Section 6 discusses resource bounds. Finally, Section 7 concludes the paper.

2 Related work

The operators of our algebra, as well as the use of interval semantics and restricted detection, are influenced by work in the area of active databases. Snoop [5], Ode [8] and SAMOS [7] are examples of active database systems where an event algebra is used to specify the reactive behaviour. These systems differ primarily in the choice of detection mechanism. SAMOS is based on Petri nets, while Snoop uses event graphs. In Ode, event definitions are equivalent to regular expressions and can be detected by state automata. In the area of active databases, event algebras are often not given a formal semantics, and algebraic properties of the operators are not presented. Also, resource efficiency is not a main concern.

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. They present a mechanism for early detection of timing constraint violation, and show that upper bounds on memory and time can be derived [10].

Common to all these systems is that they consider composite events to be instantaneous, i.e., an occurrence is associated with a single time instant. Galton and Augusto have shown that this results in unintended semantics for some operation compositions [6]. For example, an occurrence of A followed by B and then C, is accepted as an occurrence of the composite event $B;(A;C)$, since B occurs before the occurrence of $A;C$. They also present the core of an alternative, interval-based, semantics to handle these problems. We use a similar semantic base for our algebra, but we extend it with a restriction policy to allow the algebra to be implemented with limited resources while retaining useful algebraic properties.

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 [1]. Event Calculus [9] 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, they focus on how to express formally the fact that some event has occurred, and to allow inferences to be made from it.

3 Declarative semantics

We assume a discrete time model. The declarative semantics of the algebra can be used with a continuous time model as well, under restrictions that prevent primitive events that occur infinitely many times in a finite time interval.

Definition 3.1 *The temporal domain $\mathcal{T}$ is the set of all natural numbers. Also, we define $\mathcal{T}^\infty = \mathcal{T} \cup \{\infty\}$, with $i < \infty$ for all $i \in \mathcal{T}$.*

3.1 Primitive events

We assume that the system has a pre-defined set of primitive event types to which it should be able to react. These events can be external (sampled from the environment or originating from another system) or internal (such as the violation of a condition over the system state, or a timeout), but the detection mechanism does not distinguish between these categories.

Events are also allowed to carry values. These values are not manipulated in any way by the detection mechanism, but simply forwarded to the part of the system that reacts to the detected events. For example, the occurrence of a temperature alarm might carry the measured temperature value, to be used in the responding action.

Definition 3.2 *Let $\mathcal{P}$ be a set of identifiers that represent the primitive event types that are of interest to the system. For each identifier $p \in \mathcal{P}$, let $\mathrm{dom}(p)$ denote the domain from which the values of p are taken.*

Occurrences of primitive events are assumed to be instantaneous and atomic. Formally, we represent a primitive instance as a singleton set, to allow primitive and complex instances to be treated uniformly.

Definition 3.3 *If $p \in \mathcal{P}$, $v \in \mathrm{dom}(p)$ and $\tau \in \mathcal{T}$, then the singleton set $\{\langle p, v, \tau\rangle\}$ is a primitive event instance.*

The occurrences of a certain event type form an event stream. We assume that there are no simultaneous occurrences of the same primitive event.

Definition 3.4 *A primitive event stream is a set of primitive event instances all of which have the same identifier, and different times.*

In order to provide a semantic meaning to the algebra, there must be an association between the identifiers of the algebra and the real-world. This is supplied by an interpretation function as follows:

Definition 3.5 *An interpretation is a function that maps each identifier $p \in \mathcal{P}$ to a primitive event stream containing instances with identifier p.*

Example: Let $\mathcal{P} = \{A, B\}$ with $\mathrm{dom}(A) = \mathbb{N}$ and $\mathrm{dom}(B) = \{\text{high, low}\}$. Now $S = \{\{\langle A, 12, 2\rangle\},\ \{\langle A, 14, 3\rangle\},\ \{\langle A, 8, 5\rangle\}\}$ and $T = \{\{\langle B, \text{low}, 4\rangle\}\}$ are examples of primitive event streams, and $\mathcal{I}$ such that $\mathcal{I}(A) = S$ and $\mathcal{I}(B) = T$ is a possible interpretation.

3.2 Composite events

Composite events are represented by expressions built from the identifiers and the operators of the algebra.

Definition 3.6 *If $A \in \mathcal{P}$, then A is an event expression. If A and B are event expressions, and $\tau \in \mathcal{T}^{\infty}$, then $A \vee B$, $A + B$, $A - B$, and $A ;_{\tau} B$ are event expressions.*

Next, we extend the concepts of instances and streams to composite events as well as primitive.

Definition 3.7 *An event instance is a union of n primitive event instances, where $0 < n$.*

Informally, an instance of a composite event represents the primitive event occurrences that caused an occurrence of the composite event. Since we want the detection to be interval-based, we associate each instance with an interval, through the following definition.

Definition 3.8 *For an event instance a we define*

$$\begin{aligned} \mathrm{start}(a) &= \mathrm{Min}_{\langle i,v,\tau\rangle \in a}\ (\tau) \\ \mathrm{end}(a) &= \mathrm{Max}_{\langle i,v,\tau\rangle \in a}\ (\tau) \end{aligned}$$

The interval $[\mathrm{start}(a), \mathrm{end}(a)]$ can be thought of as the smallest interval which contains all the occurrences of primitive events that caused the occurrence of a. Note that a primitive event instance is an event instance, and if a is a primitive instance then $\mathrm{start}(a) = \mathrm{end}(a)$.

Example: Let $a = \{\langle A, 12, 2\rangle,\ \langle B, \text{low}, 4\rangle,\ \langle A, 8, 5\rangle\}$, then a is an event instance, and we have $\mathrm{start}(a) = 2$ and $\mathrm{end}(a) = 5$.

We also need a definition of general event streams. These will be used to represent all instances of a composite event. By this definition, a primitive event stream is an event stream, just as the names suggest.

Definition 3.9 *An event stream is a set of event instances.*

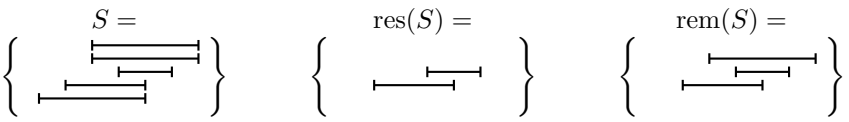

Figure 1: The effect of applying res and rem to an event stream S

3.3 Semantics

The role of the semantics is to associate, for a given interpretation, with each event expression an event stream. We will define two versions of the semantics. A simple, unrestricted version which is infeasible for resource-conscious applications, and a restricted version which can be efficiently implemented.

The following functions over event streams form the core of the algebra semantics, as they define the basic characteristics of the four operators.

Definition 3.10 *For event streams S and T, and $\tau \in \mathcal{T}^\infty$, define:*

$$\begin{aligned}
\mathrm{dis}(S,T) &= S \cup T \\
\mathrm{con}(S,T) &= \{s \cup t \mid s \in S \wedge t \in T\} \\
\mathrm{neg}(S,T) &= \{s \mid s \in S \wedge \neg\exists t(t \in T \wedge \mathrm{start}(s) \leq \mathrm{start}(t) \wedge \mathrm{end}(t) \leq \mathrm{end}(s))\} \\
\mathrm{seq}(S,T,\tau) &= \{s \cup t \mid s \in S \wedge t \in T \wedge \mathrm{end}(s) < \mathrm{start}(t) \wedge \mathrm{end}(t) - \mathrm{start}(s) \leq \tau\}
\end{aligned}$$

The main task of the restriction policy is to make the algebra effectively implementable, while retaining important algebraic properties. As these goals are somewhat contradictory, finding a suitable restriction policy has been a key issue when developing the algebra.

The following restriction function, or family of restriction functions to be more precise, does not allow instances that are fully overlapping, and removes all but one innermost instance.

Definition 3.11 *Let* res *denote an arbitrary unary function over event streams such that the following holds. For an event stream S,* $\mathrm{res}(S)$ *is a minimal subset of S for which* $\forall s(s \in S \Rightarrow \exists s'(s' \in \mathrm{res}(S) \wedge \mathrm{start}(s) \leq \mathrm{start}(s') \wedge \mathrm{end}(s') \leq \mathrm{end}(s)))$ *holds.*

For some operators of the algebra, it suffices to use a weaker restriction, where overlapping instances are accepted, as long as they have different end time. From instances with the same end time, all but one innermost is removed. This is formalised by the following family of functions.

Definition 3.12 *Let* rem *denote an arbitrary unary function over event streams such that the following holds. For an event stream S,* $\mathrm{rem}(S)$ *is a minimal subset of S for which* $\forall s(s \in S \Rightarrow \exists s'(s' \in \mathrm{res}(S) \wedge \mathrm{start}(s) \leq \mathrm{start}(s') \wedge \mathrm{end}(s') = \mathrm{end}(s)))$ *holds.*

Figure 1 illustrates the difference between the two restriction functions. Each event stream is represented graphically by the instance intervals, with time flowing from left to right in each picture.

Using the semantic functions, and the two types of restriction functions, we can define the two versions of the algebra semantics.

Definition 3.13 *The unrestricted and the restricted meaning of an event expres-*

sion for a given interpretation $\mathcal{I}$ is defined as follows:

$$
\begin{array}{lcl@{\qquad}lcl}
[A]^{\mathcal{I}} & = & \mathcal{I}(A) \text{ if } A \in \mathcal{P} & [\![A]\!]^{\mathcal{I}} & = & \mathcal{I}(A) \text{ if } A \in \mathcal{P} \\
[A \vee B]^{\mathcal{I}} & = & \mathrm{dis}([A]^{\mathcal{I}}, [B]^{\mathcal{I}}) & [\![A \vee B]\!]^{\mathcal{I}} & = & \mathrm{rem}(\mathrm{dis}([\![A]\!]^{\mathcal{I}}, [\![B]\!]^{\mathcal{I}})) \\
[A + B]^{\mathcal{I}} & = & \mathrm{con}([A]^{\mathcal{I}}, [B]^{\mathcal{I}}) & [\![A + B]\!]^{\mathcal{I}} & = & \mathrm{rem}(\mathrm{con}([\![A]\!]^{\mathcal{I}}, [\![B]\!]^{\mathcal{I}})) \\
[A - B]^{\mathcal{I}} & = & \mathrm{neg}([A]^{\mathcal{I}}, [B]^{\mathcal{I}}) & [\![A - B]\!]^{\mathcal{I}} & = & \mathrm{neg}([\![A]\!]^{\mathcal{I}}, [\![B]\!]^{\mathcal{I}}) \\
[A ;_\tau B]^{\mathcal{I}} & = & \mathrm{seq}([A]^{\mathcal{I}}, [B]^{\mathcal{I}}, \tau) & [\![A ;_\tau B]\!]^{\mathcal{I}} & = & \mathrm{res}(\mathrm{seq}([\![A]\!]^{\mathcal{I}}, [\![B]\!]^{\mathcal{I}}, \tau)
\end{array}
$$

Note that, in the restricted version, we do not require that the same rem (and res) function is used for all subexpressions, only that all functions fulfill the criteria of definitions 3.11 and 3.12. To simplify the presentation, we will use the notation $[A]$ and $[\![A]\!]$ instead of $[A]^{\mathcal{I}}$ and $[\![A]\!]^{\mathcal{I}}$ when the choice of $\mathcal{I}$ is obvious or arbitrary.

4 Properties

The restriction policy should allow an efficient implementation of the algebra while retaining algebraic properties. However, there are additional, implicit, requirements on the restriction policy, as these two would be trivially satisfied by a restriction policy that simply removes every instance. Informally, we want a restricted version that removes as few instances as possible, and never detects occurrences that are not detected by the unrestricted version. In particular, we want a formulation of the conditions under which an instance is detected by the unrestricted version but not by the restricted version. In this section, proofs have been left out due to space limitations, but the proofs in [3] are easily extended to cover temporally restricted sequence as well.

The following theorem justifies the proposed restriction policy. The subset result is not trivial, since with a different restriction policy $[\![B]\!] \subset [B]$ could easily result in $[\![A - B]\!] \supset [A - B]$. The second statement ensures that every removed instance leaves some trace in the restricted version, as the interval between the start and end time of the removed instance must be non-empty.

Theorem 4.1 *For any event expression A, the following holds:*

i) $[\![A]\!] \subseteq [A]$

ii) $a \in [A] \;\Rightarrow\; \exists a'(a' \in [\![A]\!] \wedge \mathrm{start}(a) \leq \mathrm{start}(a') \wedge \mathrm{end}(a') \leq \mathrm{end}(a))$

The following corollary is useful when only the first occurrence of an event, rather then every occurrence, is of interest to the application. For example, in a system where a temperature alarm followed by a pressure alarm should result in an immediate emergency shutdown, there is no reason to consider subsequent occurrences of this composite event.

Corollary 4.1 *For any event expression, the end time of the first unrestricted detection is the same as that of the first restricted detection.*

The previous theorem allows reasoning, both formally and informally, about the restricted semantics without a complete understanding of the restriction policy. Similarly, we should be able to compare expressions, for example to perform simplifications, without having to consider the details of the restriction policy. For this purpose, we define a weaker concept of equivalence, as ordinary set equivalence is not always achieved in the restricted version.

For example, if we have an occurrence of A followed by two occurrences of B and one occurrence of C, the set $[\![(A ;_\tau B) ;_\tau C]\!]$ contains a single instance built from the A instance, the first B instance and the C instance. The instance in $[\![A ;_\tau (B ;_\tau C)]\!]$, however, is built from the A and C instances and the second B instance. Evidently,

the two sets $[\![A;_\tau(B;_\tau C)]\!]$ and $[\![(A;_\tau B);_\tau C]\!]$ are not equal for all interpretations, but we can show that they always contain instances with the same start and end times. This is formalised by the following definition of expression equivalence.

Definition 4.1 *For event expressions A and B we define $A \cong B$ to hold if for any interpretation $\mathcal{I}$ we have*

$$\{\langle \mathrm{start}(a), \mathrm{end}(a)\rangle \mid a \in [\![A]\!]^{\mathcal{I}}\} = \{\langle \mathrm{start}(b), \mathrm{end}(b)\rangle \mid b \in [\![B]\!]^{\mathcal{I}}\}$$

Trivially, $\cong$ is an equivalence relation, and the following theorem states that it in fact defines structural congruence over event expressions.

Theorem 4.2 *If $A \cong A'$ and $B \cong B'$ then we have $(A \vee B) \cong (A' \vee B')$, $(A+B) \cong (A'+B')$, $(A-B) \cong (A'-B')$ and $(A;_\tau B) \cong (A';_\tau B')$.*

Now, the following algebraic laws can be formulated.

Theorem 4.3 *For any event expressions A, B and C, and $\tau \in \mathcal{T}$, the following laws hold:*

$$\begin{array}{rcl@{\qquad}rcl}
A \vee B & \cong & B \vee A & A;_\tau(B;_\tau C) & \cong & (A;_\tau B);_\tau C \\
A \vee A & \cong & A & A+(B \vee C) & \cong & (A+B) \vee (A+C) \\
A \vee (B \vee C) & \cong & (A \vee B) \vee C & (A \vee B)+C & \cong & (A+C) \vee (B+C) \\
A+B & \cong & B+A & (A \vee B)-C & \cong & (A-C) \vee (B-C) \\
A+A & \cong & A & (A-B)-B & \cong & A-B \\
A+(B+C) & \cong & (A+B)+C & A-(B \vee C) & \cong & (A-B)-C
\end{array}$$

5 Operational semantics

Next, we will present an imperative implementation of the algebra, and prove that it is equivalent to the declarative semantics presented previously. The algorithm is executed once every time instant, and computes the current instance of a given composite event from the current instances of the primitive events.

Let E be the event expression that is to be detected. Assign the numbers $1 \ldots m$ to the subexpressions of E in bottom-up order, and let E^i denote subexpression number i.

Each operator occurrence in the expression requires its own state variables, and thus variables are indexed from 1 to m. The variable a_i is used to store the current instance of E^i, and the variables l_i, r_i, t_i and q_i are auxiliary variables to store information about the past needed to detect E^i properly. In l_i and r_i, a single event instance is stored, t_i stores a time instant and q_i contains a set of event instances. The symbol $\langle\rangle$ represents a non-occurrence, and we use τ^c to access the current time.

The algorithm is presented in Figure 2. The rest of this section describes the implementation of each operator in more detail, including a correctness lemma for each operator. Finally, the correctness of the whole algorithm is addressed.

The following predicate will be used in the analysis of the algorithm. Informally, $P(i,\tau)$ states that the variable a_i contains the correct instance of E^i at time τ.

Definition 5.1 *For $1 \le i \le m$ and $\tau \in \mathcal{T}$, we define $P(i,\tau)$ to hold if:*

$$(a_i \in [\![E^i]\!] \wedge \mathrm{end}(a_i) = \tau) \ \vee \ (a_i = \langle\rangle \wedge \neg\exists e(e \in [\![E^i]\!] \wedge \mathrm{end}(e) = \tau))$$

```
for i from 1 to m
    if E^i ∈ P then
        a_i := the current instance of E^i, or ⟨⟩ if there is none.
    if E^i = E^j ∨ E^k then
        if a_j = ⟨⟩ or (a_k ≠ ⟨⟩ and start(a_j) ≤ start(a_k)) then a_i := a_k
            else a_i := a_j
    if E^i = E^j + E^k then
        if a_j ≠ ⟨⟩ and (l_i = ⟨⟩ or start(l_i) < start(a_j)) then l_i := a_j
        if a_k ≠ ⟨⟩ and (r_i = ⟨⟩ or start(r_i) < start(a_k)) then r_i := a_k
        if l_i = ⟨⟩ or r_i = ⟨⟩ or (a_j = ⟨⟩ and a_k = ⟨⟩) then a_i := ⟨⟩
            else if a_j ≠ ⟨⟩ and (a_k = ⟨⟩ or start(a_k) ≤ start(a_j))
                then a_i := a_j ∪ r_i
                else a_i := l_i ∪ a_k
    if E^i = E^j − E^k then
        if a_k ≠ ⟨⟩ and t_i < start(a_k) then t_i := start(a_k)
        if a_j ≠ ⟨⟩ and t_i < start(a_j) then a_i := a_j else a_i := ⟨⟩
    if E^i = E^j ;_τ' E^k then
        a_i := ⟨⟩
        if a_k ≠ ⟨⟩ then
            foreach e in q_i
                if end(e) < start(a_k) then
                    q_i := q_i − {e}
                    if a_i = ⟨⟩ or start(a_i) < start(e) then a_i := e
            if a_i ≠ ⟨⟩ then a_i := a_k ∪ a_i
        if a_j ≠ ⟨⟩ and t_i < start(a_j) then
            q_i := q_i ∪ {a_j}
            t_i := start(a_j)
        foreach e in q_i
            if start(e) ≤ τ^c − τ' then q_i := q_i − {e}
```

Figure 2: Algorithm for detecting E

5.1 Primitive expressions

The algorithm for primitive expressions establishes the relation between the algorithm input and the interpretation function of the declarative semantics.

Lemma 5.1 *Let $E^i \in \mathcal{P}$ and let τ^c be the current time. Then $P(i, \tau^c)$ holds after executing the loop body once.*

Proof: Assuming that the interpretation represents the real-world, this holds trivially. □

5.2 Disjunction

The disjunction operator is fairly simple and requires no auxiliary variables.

Lemma 5.2 *Let $E^i = E^j \vee E^k$ and assume that $P(j, \tau) \wedge P(k, \tau)$ holds. Then $P(i, \tau)$ holds after executing the loop body once.*

Proof: From $P(j, \tau)$ and $P(k, \tau)$ it is straightforward to show that $P(i, \tau)$ holds when one or both of a_j and a_k are $\langle\rangle$. Otherwise, both a_j and a_k belong to $\text{dis}([\![E^j]\!], [\![E^k]\!])$, and thus the one with maximum start time must be in $[\![E^i]\!]$. If

the start times are equal, selecting a_k is in accordance with the definition of rem. □

5.3 Conjunction

For conjunctions, it is necessary to store the instance with maximum start time so far from each of the two subexpressions. This is formalised by the following definition, which states that l_i and r_i have correct values at the start of time instant τ.

Definition 5.2 *For $1 \leq i \leq m$ such that $E^i = E^j + E^k$, and for $\tau \in \mathcal{T}$, we define $C(i,\tau)$ to hold if the following holds:*

- *l_i is an element in $\{e \mid e \in [\![E^j]\!] \wedge \mathrm{end}(e) < \tau\}$ with maximum start time, or $\langle\rangle$ if that set is empty.*
- *r_i is an element in $\{e \mid e \in [\![E^k]\!] \wedge \mathrm{end}(e) < \tau\}$ with maximum start time, or $\langle\rangle$ if that set is empty.*

Lemma 5.3 *For $E^i = E^j + E^k$ we have*

i) $C(i,0)$ holds for an initial state where $l_i = \langle\rangle$ and $r_i = \langle\rangle$.

ii) Assume that $P(j,\tau) \wedge P(k,\tau) \wedge C(i,\tau)$ holds. Then $P(i,\tau) \wedge C(i,\tau+1)$ holds after executing the loop body once.

Proof: As *i)* follows trivially from the definition of C, we focus on showing *ii)*. By $C(i,\tau)$ and $P(k,\tau)$ we can see that l_i will contain the value specified by $C(i,\tau+1)$ after the execution of the first conditional in the conjunction part of the algorithm. The same holds for the second conditional and r_i, so $C(i,\tau+1)$ holds after executing the first two conditionals.

If the guard of the third conditional is satisfied, then $\neg\exists e(e \in \mathrm{con}([\![E^j]\!], [\![E^k]\!]) \wedge \mathrm{end}(e) = \tau)$, and thus $P(i,\tau)$ holds after assigning $\langle\rangle$ to a_i. If the guard is false, we know that $\exists e(e \in [\![E^i]\!] \wedge \mathrm{end}(e) = \tau)$. For the case when one of a_j and a_k are $\langle\rangle$, there is only one $e \in \mathrm{con}([\![E^j]\!], [\![E^k]\!])$ with $\mathrm{end}(e) = \tau$, and thus this e is in $[\![E^i]\!]$.

For the other case, when neither a_j nor a_k is $\langle\rangle$, both $a_j \cup r_i$ and $l_i \cup a_k$ belong to $\mathrm{con}([\![E^j]\!], [\![E^k]\!])$. If the inner conditional holds, we have $\mathrm{start}(a_k) \leq \mathrm{start}(a_j)$ and by $C(i,\tau+1)$ we also have $\mathrm{start}(a_k) \leq \mathrm{start}(r_i)$. Thus $\mathrm{start}(l_i \cup a_k) \leq \mathrm{start}(a_j \cup r_i)$ which means that $a_j \cup r_i \in [\![E^i]\!]$ does not violate the constraints in the definition of rem. Similarly, if the inner conditional is false, we have $l_i \cup a_k \in [\![E^i]\!]$. □

5.4 Negation

According to the semantics of the negation operator, an instance of B is an instance of $B-C$ unless it is invalidated by some instance of C occurring within its interval. If the current instance of B is invalidated at all, it is invalidated by the instance of C with maximum start time (of those that have occured so far). Thus, it is sufficient to store a single start time, since the end time is trivially known to be less than the end time of the current instance of B.

Definition 5.3 *For $1 \leq i \leq m$ such that $E^i = E^j - E^k$, and for $\tau \in \mathcal{T}$, we define $N(i,\tau)$ to hold if t_i is the maximum start time of the elements in $\{e \mid e \in [\![E^k]\!] \wedge \mathrm{end}(e) < \tau\}$, or -1 if this set is empty.*

Lemma 5.4 *For $E^i = E^j - E^k$ we have*

i) $N(i,0)$ holds for an initial state where $t_i = -1$.

ii) Assume that $P(j,\tau) \wedge P(k,\tau) \wedge N(i,\tau)$ *holds. Then* $P(i,\tau) \wedge N(i,\tau{+}1)$ *holds after executing the loop body once.*

Proof: *i*) holds trivially, and for *ii*) it is straightforward to show that the first conditional updates t_i to the value specified by $N(i,\tau+1)$. If the guard of the second conditional holds, then by $P(j,\tau)$ we have $a_j \in [\![E^j]\!]$. According to $N(i,\tau{+}1)$ there is no e in $[\![E^k]\!]$ with $\mathrm{start}(a_j) \leq \mathrm{start}(e)$ and $\mathrm{end}(e) < \mathrm{start}(a_j) = \tau$, and thus $a_j \in \mathrm{neg}([\![E^j]\!], [\![E^k]\!])$, which means that $P(i,\tau)$ holds. □

5.5 Sequence

The sequence operator requires the most complex algorithm. The reason for this is that in order to detect a sequence $B;_\tau C$ correctly, we must store several instances of B. Once C occurs, the start time of that instance determines with which of the stored instances of B it should be combined to form the instance of $B;_\tau C$.

Definition 5.4 *For* $1 \leq i \leq m$ *such that* $E^i = E^j;_{\tau'} E^k$*, and for* $\tau \in \mathcal{T}$*, we define* $S(i,\tau)$ *to hold if all of the following conditions hold:*

- t_i *is the maximum start time of the elements in* $\{e \mid e \in [\![E^j]\!] \wedge \mathrm{end}(e) < \tau\}$*, or* -1 *if this set is empty.*
- q_i *is the set of every instance* $e \in [\![E^j]\!]$ *with* $\mathrm{end}(e) < \tau$ *except when* e *is invalidated by one (or more) of the following conditions:*
  1. $\mathrm{start}(e) < \tau - \tau'$
  2. *There is an instance* $e' \in [\![E^k]\!]$ *with* $\mathrm{end}(e') < \tau$ *and* $\mathrm{end}(e) < \mathrm{start}(e')$*.*
  3. *There is another instance* $e' \in [\![E^j]\!]$ *with* $e' \neq e$*,* $\mathrm{start}(e) \leq \mathrm{start}(e')$ *and* $\mathrm{end}(e') < \mathrm{start}(e)$*.*

Lemma 5.5 *For* $E^i = E^j;_{\tau'} E^k$ *we have*

i) $S(i,0)$ holds for an initial state where $q_i = \emptyset$ and $t_i = -1$.

ii) Assume that $P(j,\tau) \wedge P(k,\tau) \wedge S(i,\tau)$ holds. Then $P(i,\tau) \wedge S(i,\tau{+}1)$ holds after executing the loop body once.

Proof: We focus on *ii*) as *i*) holds trivially. The first conditional removes from q_i those instances of $[\![E^j]\!]$ that are invalidated, according to condition 2, by the current instance of $[\![E^k]\!]$. After the foreach statement, a_i contains the removed instances with maximum start time (or $\langle\rangle$ if no instances were removed). If $a_i = \langle\rangle$, then by $P(k,\tau)$ and $S(i,\tau)$ we have $\neg\exists e(e \in [\![E^i]\!] \wedge \mathrm{end}(e) = \tau)$, so $P(i,\tau)$ holds. If $a_i \neq \langle\rangle$, then $a_i \cup a_k \in \mathrm{seq}([\![E^j]\!], [\![E^k]\!], \tau')$ and then by $S(i,\tau)$ we have that any $e \in \mathrm{seq}([\![E^j]\!], [\![E^k]\!], \tau')$ such that $\mathrm{start}(a_i \cup a_k) \leq \mathrm{start}(e)$ and $\mathrm{end}(e) \leq \mathrm{end}(a_i \cup a_k)$ would imply $\mathrm{start}(a_i \cup a_k) = \mathrm{start}(e)$ and $\mathrm{end}(e) = \mathrm{end}(a_i \cup a_k)$. Thus, $a_i \cup a_k \in [\![E^i]\!]$ satisfies the constraint in the definition of res, so $P(i,\tau)$ holds after assigning $a_i \cup a_k$ to a_i.

The second conditional adds the current instance of $[\![E^j]\!]$ to q_i, unless it is invalidated by some $e' \in [\![E^j]\!]$ with $\mathrm{start}(e') = t_i$ and $\mathrm{end}(e') < \tau$, as specified by condition 3. Additionally, t_i is updated to the value specified by $S(i,\tau+1)$. Finally, the lastforeach statement removes all elements from q_i that are invalidated by condition 1 in $S(i,\tau{+}1)$. □

5.6 Putting it all together

The following theorem formalises the correctness of the algorithm.

Theorem 5.1 *After executing the algorithm at time instants* 0 *to* τ, $P(i,\tau)$ *holds for* $1 \leq i \leq m$.

Proof: Induction over i using the lemmas shows that each subexpression is correctly detected (since they are numbered bottom-up) provided that the state is correct at the start of the time instant. Induction over time instants 0 to τ shows that the state is correct at the start of each time instant, and thus the theorem holds. □

The algorithm is described for the case when the expression is unknown at compile time, in which case the main loop selects dynamically which algorithm to execute for each subexpression. If the expression is known at compile time, the main loop can be unrolled and the top-level conditionals, as well as all indices, can be statically determined. A concrete example of this is given in Figure 3.

Initialisation:
    $t_5 := -1$
Executed every time instant:
    $a_1 :=$ the current instance of A, or $\langle\rangle$ if there is none.
    $a_2 :=$ the current instance of B, or $\langle\rangle$ if there is none.
    if $a_1 = \langle\rangle$ or ($a_2 \neq \langle\rangle$ and $\text{start}(a_1) \leq \text{start}(a_2)$) then $a_3 := a_2$
        else $a_3 := a_1$
    $a_4 :=$ the current instance of C, or $\langle\rangle$ if there is none.
    if $a_4 \neq \langle\rangle$ and $t_5 < \text{start}(a_4)$ then $t_5 := \text{start}(a_4)$
    if $a_3 \neq \langle\rangle$ and $t_5 < \text{start}(a_3)$ then $a_5 := a_3$ else $a_5 := \langle\rangle$

Figure 3: Statically simplified algorithm for detecting $(A \vee B) - C$

6 Resource bounds

From the algorithm in the previous section, it is obvious that each disjunction, conjunction and negation in the event expression requires a constant amount of storage, and contributes with a constant factor to the computation time of the whole detection algorithm. For the sequence operator, the memory usage depends on the maximum size of q_i. For a temporally restricted sequence, i.e., $B;_\tau C$ with $\tau \in \mathcal{T}$, it follows from the definition of T that q_i never contains more than $\tau+1$ elements.

The remaining case is the temporally unrestricted sequence $B;_\infty C$, for which no memory bound exists in the general case. For an important subset of expressions, however, the sequence operator can be implemented more efficiently than in the general case. If we know that all instances of C are instantanous (i.e., start time is equal to end time), for example because C is primitive or a disjunction of primitives, then there is no need to store several instances of B to detect $B;_\tau C$ correctly.

Definition 6.1 *For an event expression A, let* $ins(A)$ *hold iff*

$$(A \in \mathcal{P}) \ \vee \ (A = B \vee C \wedge ins(B) \wedge ins(C)) \ \vee \ (A = B - C \wedge ins(B))$$

The improved algorithm for $B;_\tau C$ under the assumption that $ins(C)$ holds, is shown in Figure 4.

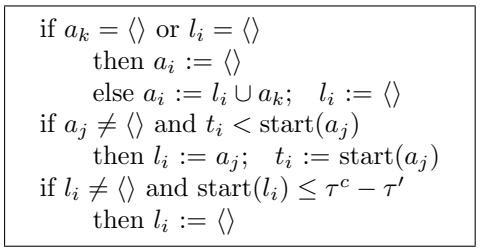

Figure 4: Algorithm for $E^i = E^j ;_{\tau'} E^k$ when $ins(E^k)$ holds

Definition 6.2 *For $1 \leq i \leq m$ such that $E^i = E^j ;_{\tau'} E^k$, and for $\tau \in \mathcal{T}$, we define $Q(i, \tau)$ to hold if all of the following conditions hold:*

- *t_i is the maximum start time of elements in $\{e \mid e \in [\![E^j]\!] \wedge \text{end}(e) < \tau\}$, or -1 if this set is empty.*
- *l_i is an instance with maximum start time from the instances that would belong to q_i by Definition 5.4, or $\langle\rangle$ if that set is empty.*

Lemma 6.1 *For $E^i = E^j ;_{\tau'} E^k$ such that $ins(E^k)$ holds, we have*

i) $Q(i, 0)$ holds for an initial state where $l_i = \langle\rangle$ and $t_i = -1$.

ii) Assume that $P(j, \tau) \wedge P(k, \tau) \wedge Q(i, \tau)$ holds. Then $P(i, \tau) \wedge Q(i, \tau+1)$ holds after executing the algorithm in Figure 4.

Proof: *i)* holds trivially. For *ii)* we see that if the first conditional holds then $\neg\exists e(e \in [\![E^i]\!] \wedge \text{end}(e) = \tau)$, so $P(i, \tau)$ holds after assigning $\langle\rangle$ to a_i. Otherwise, $l_i \cup a_k \in \text{seq}([\![E^j]\!], [\![E^k]\!], \tau')$ since $\text{end}(l_i) < \tau = \text{start}(a_k)$ according to $Q(i, \tau)$ and $ins(E^k)$. By $Q(i, \tau)$ and $ins(E^k)$ we have that $l_i \cup a_k \in [\![E^i]\!]$ satisfies the constraint in the definition of res, so $P(i, \tau)$ holds after assigning $l_i \cup a_k$ to a_i. Since the current instance of $[\![E^k]\!]$ invalidates all instances of $[\![E^j]\!]$ that have occured earlier (including l_i) according to condition 2, $\langle\rangle$ is assigned to l_i.

The second conditional checks whether the current instance of $[\![E^j]\!]$ satisfies condition 3, and updates l_i if neccesary. Finally, if l_i does not satisfy condition 1, no other instance of $[\![E^j]\!]$ satisfies all three conditions, so $\langle\rangle$ is assigned to l_i. □

Trivially, the new sequence algorithm requires a constant amount of storage, and thus the memory needed to detect an expression E is constant provided that for any subexpression of E on the form $B ;_\tau C$ for which $ins(C)$ does not hold, we have $\tau \neq \infty$.

Once the size limits of the q_i variables have been established, the worst case temporal complexity of the algorithm can be derived. A straightforward implementation of the q_i variables gives a total complexity of $O(mn)$, where m is the number of subexpressions in E and n is the maximum size limit of the q_i variables.

7 Conclusions and future work

We have presented a fully formal event algebra with operators for disjunction, conjunction, negation and sequence. The algebra is defined in terms of a restriction policy to allow an efficient implementation. This policy is carefully designed to ensure that important algebraic properties correspond to the intuitive behaviour of the operators also in the restricted version of the semantics.

An implementation of the algebra was presented and its semantics was proved equal to the declarative semantics. Also, we have identified a subset of expressions for which detection can be performed with bounded resources.

Our ongoing work includes investigating how to combine the algebra with languages that specifically target reactive systems, in particular Esterel [2], AFRP [12, 11] and Timber [4]. We are also investigating how information about primitive event frequencies can be used to lower the size limits of the q_i variables, resulting in more precise worst case memory and time estimates.

References

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

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

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

[4] 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)*, Lecture Notes in Computer Science, Beijing, China, 26–29 November 2003. Springer-Verlag.

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

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

[7] 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.

[8] 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.

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

[10] 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.

[11] 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.

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