IntervalEvents in EScala: Design

Michael Kutschke und Frank Englert

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Inhaltsverzeichnis

1 Grundlegende Definitionen **1**

2 Operatoren **1**
2.1 Komplement comp(int) : Intervall 1
2.2 Vereinigung int1 || int2 : Intervall 2
2.3 Schnitt int1 && int2 : Intervall 2
2.4 Differenz int1 \ int2 : Interval 2
2.5 within(int,e) : Event 2
2.6 !within(int,e) : Event 2
2.7 StrictlyWithin(int,e) : Event 2
2.8 !strictlyWithin(int,e) : Event 2
2.9 weitere Anmerkungen 3

Abbildungsverzeichnis **3**

Todo:

before, after, complement als Member des Intervall-Events, dabei ist das complement ein Lazy object,

das before und das After werden Ãijberladen, wenn das after-Event nicht das Intervall beenden soll. Dies ist z.B. beim ExecutionEvent der Fall.

1 Grundlegende Definitionen

Sei before(int) : Event das Ereignis, das den Beginn des Intervalls int angibt und after(int) : Event jenes, welches das Ende von int angibt. Dies kann in unterschiedlicher Form erfolgen:

- einmal before(_) zu Beginn und einmal after(_) am Ende (Standard in der jetzigen Fassung)
- mehrfaches before(_) und ein after(_) am Ende (Use case? Wohl meist nicht erwünscht)
- verschachtelte, paarweise passende before(_) und after(_) Ereignisse (Bsp. Exec)
- beliebige before(_) und after(_) Events innerhalb des Intervals

Ausserdem gebe active(int) : boolean an, ob das Intervall äktiv¨ ist. In der jetzigen Fassung bedeutet dies einfach, ob man sich zeitlich innerhalb des Intervalls befindet. Das Umschalten wude bisher als Sink realisiert. Dies führt jedoch zu Indeterminismus, sodass e && within(between(e,e')) beim ersten Vorkommen von e nicht unbedingt immer ausgeführt wird. Dies ist unschön und sollte vermieden werden.

Daher sollte active eine andere Semantik erhalten und als reaction aktualisiert werden. So kann sichergestellt werden, dass der Zustand von active zu jedem Zeitpunkt bekannt ist. Die Semantik ändert sich also von [—-int—-] zu]——int—]. Dies ist keine Einschränkung, wie im weiteren gezeigt werden wird.

FÃijr alle Intervalle muss gelten, dass active nur bei einem before(_) oder after(_) Event umgeschaltet wird, wobei active nur von einem before(_) Event gesetzt und von einem after(_) Event zurÃijckgesetzt werden kann. Ausserdem sollten ausserhalb des Intervalls keine isolierten Events auftreten.

2 Operatoren

Im folgenden werden einige Operatoren sowie ihre mögliche Imlementierung und evtl. Schwächen des jeweiligen Ansatzes diskutiert.

Spezielle Operatoren:

- merge(int) : macht ein Intervall zu einem Standard-Intervall

2.1 Komplement comp(int) : Intervall

- before(comp(int)) <= after(int)
- after(comp(int)) <= before(int) && ! active(int)
- active(comp(int)) <= ! active(int)

Probleme: An den Schnittpunkten ist ein Event laut dieser Definition sowohl in int als auch comp(int). Bei verschachtelten Events gibt es unerwünschte, isolierte before(comp(int))-Events.

mögliche Lösungen: Komplement als Methode von Intervall Event überladbar machen, und in Spezialfällen gesondert behandeln. Für das Schnittpunkt-problem gibt es (noch) keine Lösung, ist evtl. auch positiv (siehe Differenz)

Bemerkung: die zusätzlichen Bedingungen an after und active existieren aufgrund der Nicht-Standard-Intervalle. Die explizite Definition von active ist wichtig damit das Komplement auch anfangs aktiv sein kann.

2.2 Vereinigung int1 || int2 : Intervall

Die Semantik der ||-Vereinigung soll eine Vereinigung der Zeitpunkte von int1 und int2 sein, also wäre, falls int1 ein Exec Event wäre, int1 || int1 ein Standard Intervall.

- before(int1 || int2) <= (before(int1) || before(int2)) && ! active(int1 || int2)
- after(int1 || int2) <= ((before(comp(int1)) && ! active(int2)) || (before(comp(int2)) && ! active(int1)) || (before(comp(int1)) && before(comp(int2)))) \before(int1 || int2)

Anmerkung: statt after(int1) wurde hier before(comp(int1)) verwendet. In der jetzigen Fassung ist dies äquivalent, allerdings bietet diese Formulierung den Vorteil, dass falls die Probleme für Komplement gelöst werden, auch zum größten Teil die Probleme mit Vereinigung verschwinden.

Problem: Verschachtelte Events, ähnlich wie bei Komplement. Eine Lösung für das Problem mit Komplement löst auch dieses Problem.

2.3 Schnitt int1 && int2 : Intervall

- before(int1 && int2) <= (((before(int1) && active(int2)) || (before(int2) && active(int1)) || (before(int1) && before(int2))) \(after(int1) || after(int2))) && ! active(int1 && int2)
- after(int1 && int2) <= before(comp(int1)) || before(comp(int2))
- active(int1 && int2) <= active(int1) && active(int2)

Anmerkungen: zu before(comp(_)) siehe Anm. zur Vereinigung

2.4 Differenz int1 \ int2 : Interval

Alle Zeitpunkte, die in int1 liegen, nicht aber in int2 (ausser Start bzw. Endzeitpunkt, siehe Anm.). Wegen dieser Einschränkung kann es sinnvoll sein, eine Bedingung an einen Zeitpunkt über eine Verbindung von within und !within auszudrücken.

int1 \ int2 <= int1 && comp(int2)

Anmerkung: Wie bei Komplement erwähnt, sind die Eckpunkte von int1 und int2 evtl. fälschlicherweise mit in int1 \ int2. Dies lässt sich nicht lösen, solange keine Möglichkeit gefunden wird, offene Intervalle zu modellieren (z.B. über explizite before-Trigger o.ä.). U.U. ist dies aber auch keine schlechte Eigenschaft, so dass ein Wechsel int -> comp(int) atomar zu einem bestimmten Zeitpunkt stattfindet.

2.5 within(int,e) : Event

within(int,e) <= (e && active(int)) || (e && before(int))

2.6 !within(int,e) : Event

!within(int,e) <= (e && ! active(int)) \ before(int)

2.7 StrictlyWithin(int,e) : Event

StrictlyWithin(int,e) <= (e && active(int)) \ after(int)

Problem: Bei verschachtelten Intervallen werden einzelne Zeitpunkte fälschlicherweise herausgeschnitten. Deshalb möglichst nur auf StandardIntervalle anwenden. z.B. via comp(comp(_)) oder einen Standard wrapper, siehe unten.

2.8 !strictlyWithin(int,e) : Event

!strictlyWithin(int,e) <= (e && ! active(int)) || (e && after(int))

Problem: Auch hier werden bei verschachtelten Intervallen multiple after(_) Events mit aufgenommen.

2.9 weitere Anmerkungen

Eine andere mögliche Lösung für das before(comp(int)) Problem wäre es, statt Komplement zu überladen, einen Operator std(int) zu definieren, dessen Semantik ist, dass er die Standard-Entsprechung eines Intervalls zurückgibt, d.h. dass erfüllt sein muss, dass before(int) niemals aktiviert wird, wenn das Intervall schon aktiv ist und das active(int) niemals true zurückliefert nach einem after(int) und vor einem before(int). So ließe sich Komplement leicht implementieren (after(std(int)); before(std(int))) und entsprechend die anderen Operatoren.

Der Umstand, dass sich int und comp(int) zwei Zeitpunkte teilen, mag unintuitiv erscheinen, allerdings bietet dies nicht nur Nachteile. So ist int || comp(int) wie erwartet immer aktiv, beginnt und endet niemals. int && comp(int) ist niemals aktiv und lÃűst auch niemals before oder after Ereignisse aus.

Abbildungsverzeichnis