A Linear Temporal Logic with Heartbeat

MOHAMMAD-ALI A'RÂBI, University of Freiburg, Germany

ReactiveX has become a standard model for event-driven programming and is used widely in the industry. Still, abstractions such as observable have not been studied adequately. We will introduce a new modality in linear temporal logic to represent sequences of event. This modality is a generalization of the future/eventually modality in Girard's linear logic setting.

CCS Concepts: • **Theory of computation → Modal and temporal logics**; **Linear logic**.

Additional Key Words and Phrases: observable, temporal logic, type system

1 INTRODUCTION

Describing events with the future/eventually modality of linear-time temporal logic was already done by Paykin et al. [5]. But a theoretical study on streams of events has not yet been carried out, although most event-driven production codes are wrapped into streams of events rather than single events. Different ports of ReactiveX such as RxJava and RxJS are good evidents. We will first introduce a new modality called *heartbeat* in a linear version [2] of the linear-time temporal logic [6], and then try to complete the Curry–Howard correspondent [3] with an extension of linear/non-linear logic [1].

2 MOTIVATION

Assume $\Diamond\alpha$ describes the type of a value that will become available eventually, e.g. a promise in JavaScript or a single in RxJava. We call such a value an event. An observable in the Rx sense is a sequence of events. The sequence may hold just one event or more. Let's denote the type of a sequence of n events with $\heartsuit_n\alpha$ where $\heartsuit_1\alpha := \Diamond\alpha$ and $\heartsuit_{n+1}\alpha := \heartsuit_n\alpha \otimes \alpha$. We would also like to introduce a type for infinite sequence of events: $\heartsuit_\omega\alpha := !\Diamond\alpha$. Finally, a general type is defined for all kinds of event sequences:

$$\heartsuit\alpha := \left(\bigoplus_{i=1}^{\omega} \heartsuit_i\alpha\right) \oplus \heartsuit_\omega\alpha. \tag{1}$$

Note that right-hand side of the equation 1 does not belong to the language of temporal logic, as it has an infinite summation on formulae. Hence, we will try to add $\heartsuit$ as distinct symbol to the language and give it meaning. Furthermore, note that all this work is only valuable in linear logic, i.e. a logic without weakening and contraction, otherwise $\heartsuit = \heartsuit_\omega = \heartsuit_n = \Diamond$.

3 THE LOGIC

The language of our logic is defined with the following grammar:

$$\alpha, \beta ::= 0 \mid a \mid \alpha \multimap \beta \mid \alpha \oplus \beta \mid \alpha \otimes \beta \mid !\alpha \mid \Diamond\alpha \mid \heartsuit\alpha.$$

Here 0 is a constant and the linear version of contradiction, the connective $\multimap$ is called *lollipop* which is the bilinear version of implication, $\oplus$ is called *additive disjunction* and is a bilinear version of or, $\otimes$ is called *multiplicative conjunction* or *tensor* and is a bilinear version of and, and the modality ! is called *of course* or *bang* and makes the formulae non-linear (*i.e.* discardable or duplicable). So far the symbols were drawn from Girard's linear logic. The modality $\Diamond$ is called *future* or *eventually* and is drawn from linear-time temporal logic. The last symbol $\heartsuit$ is a modality called *heartbeat* which we is a generalization of $\Diamond$ and we will explain extensively. Two syntactic sugars are *negation* $\neg\alpha := \alpha \multimap 0$ and the *global* modality which is the dual of future: $\neg\Box\alpha = \Diamond\neg\alpha$ and $\Box\neg\alpha = \neg\Diamond\alpha$.

Author's address: Mohammad-Ali A'râbi, University of Freiburg, Freiburg, Germany, mohammadaliarabi@acm.org.

In the setting of LTL, $\Diamond\alpha$ is read "eventually α" or "α will become true in the future". We can think of $\heartsuit\alpha$ as "occasionally α".

$$\frac{\Gamma \vdash \Diamond\alpha}{\Gamma \vdash \heartsuit\alpha}\ \heartsuit\text{-I} \qquad \frac{\Gamma_1 \vdash \heartsuit\alpha \quad \Gamma_2, \Diamond\alpha \vdash \neg\Diamond\alpha}{\Gamma_1, \Gamma_2 \vdash \Diamond\alpha}\ \heartsuit\text{-E}$$

Two more elimination rules are called *fork-join* and *flat-map*:

$$\frac{\Gamma \vdash \heartsuit\alpha}{\Gamma \vdash \Diamond!\alpha}\ \text{FJ} \qquad \frac{\Gamma_1 \vdash \heartsuit\alpha \quad \Gamma_2, \alpha \vdash \heartsuit\beta}{\Gamma_1, \Gamma_2 \vdash \heartsuit\beta}\ \text{FM}$$

Fork-join successfully shows that there is no difference between $\Diamond$ and $\heartsuit$ in the non-linear world.

4 COMPLETING THE CURRY–HOWARD CORRESPONDENCE

As discussed earlier, $\Diamond\alpha$ describes the type of single events, e.g. `Single<`α`>` in the RxJava sense. In contrast, $\heartsuit\alpha$ describes the type of an event stream, e.g. `Observable<`α`>`.

Fork-join. When t is a finite stream of events, one can wait for all of them to resolve, and then get an array of the resolved values. This action is called `forkJoin`:

$$\frac{\Gamma \vdash t : \heartsuit\alpha}{\Gamma \vdash \texttt{forkJoin}\ t : \Diamond!\alpha}\ \text{FJ}$$

The bang modality can be used to describe arrays. So, in the observable terminology, a fork-join is basically transforming a finite stream of events into a single event that contains an array, by waiting for all the events to resolve.

Flat-map. Flat-map is the transformation that converts each event in the stream into another stream, and then unpacks all of the resulting events and puts them into a linear timeline [4].

$$\frac{\Gamma_1 \vdash t_1 : \heartsuit\alpha \quad \Gamma_2, x : \alpha \vdash t_2 : \heartsuit\beta}{\Gamma_1, \Gamma_2 \vdash t_1\ \texttt{flatMap}\ (\lambda x.t_2) : \heartsuit\beta}\ \text{FM}$$

From single. A single event can be assumed a singleton stream of events:

$$\frac{\Gamma \vdash t : \Diamond\alpha}{\Gamma \vdash \texttt{fromSingle}\ t : \heartsuit\alpha}\ \heartsuit\text{-I}$$

REFERENCES

[1] P. N. Benton. 1994. A Mixed Linear and Non-Linear Logic: Proofs, Terms and Models (Extended Abstract). In *Computer Science Logic, 8th International Workshop, CSL '94, Kazimierz, Poland, September 25-30, 1994, Selected Papers (Lecture Notes in Computer Science, Vol. 933)*, Leszek Pacholski and Jerzy Tiuryn (Eds.). Springer, 121–135. https://doi.org/10.1007/BFb0022251

[2] Jean-Yves Girard. 1987. Linear Logic. *Theor. Comput. Sci.* 50 (1987), 1–102. https://doi.org/10.1016/0304-3975(87)90045-4

[3] W. Howard. 1969. The formulae-as-types notion of construction.

[4] Ben Lesh. 2017. mergeMap. https://rxjs.dev/api/operators/mergeMap

[5] Jennifer Paykin, N. Krishnaswami, and S. Zdancewic. 2016. The Essence of Event-Driven Programming.

[6] Amir Pnueli. 1977. The Temporal Logic of Programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977.* IEEE Computer Society, 46–57. https://doi.org/10.1109/SFCS.1977.32