All You Ever Wanted to Know About Frames  
(But Were Always Afraid to Ask)

1st Author

1st author's affiliation  
1st line of address  
2nd line of address  
Telephone number, incl. country code

1st author's email address

2nd Author

2nd author's affiliation  
1st line of address  
2nd line of address  
Telephone number, incl. country code

2nd E-mail

3rd Author

3rd author's affiliation  
1st line of address  
2nd line of address  
Telephone number, incl. country code

3rd E-mail

**ABSTRACT**

Abstract.

# INTRODUCTION

*Kristin*

# USE CASES

*Kristin*

# FRAME SPECIFICATION

To give a formal specification of frames, we first define the framing of a data stream and then specify the functions that can be used to define a framing. We also describe how a framing can be used to fill frames with tuples from a stream. Finally, we introduce example of commonly used framings and describe how they can be expressed based on the given specification.

**Definition:** A data stream is defined as an infinite sequence of tuples . All tuples of a data stream have the same schema. One attribute of the schema, the *progressing attribute*, is distinguished as is defines the logical order of the tuples. The stream progresses on this attribute, i.e. if is this attribute, then for any , there is an s.t. . Note that while the presence of a progressing attribute implies a logical ordering, it does not require that the tuples within the stream are physically ordered.

## Framing of a Data Stream

**Definition:** A *framing* of a data stream is represented as an infinite sequence of intervals over the domain of the progressing attribute , where .[[1]](#footnote-1) Each interval defines an *extent* as a set of tuples , where is the progressing attribute. Generally, a framing can be thought of as a function , where is the domain of all streams and is the domain of all framings.

Theoretically, all set of intervals that satisfy the conditions given in the definition are valid framings of a stream. In practice, however, it is useful to further constrain framings. We distinguish *local* and *global conditions* that can be applied to restrict which intervals are reported. A local condition is a property that is true for each interval extent in the framing. We introduce three types of local conditions. First, a framing can be restricted by specifying a **predicate** that needs to hold for all its interval extents. For example, the predicate restricts the framing to intervals where the value of attribute of all contained tuples is consistently larger than a constant .

If a framing is restricted by a predicate, the second local condition that can be applied is the **span** of the framing intervals. The span of an interval determines whether it is maximal or minimal. For example, a framing that is restricted by the threshold predicate above is not guaranteed to report unique set of intervals as for every interval that satisfies the predicate all sub-intervals down to individual tuples could be reported. In this case, requiring the framing to be maximal ensures that only non-overlapping intervals of maximal length are reported.

The third local property constrains the **starting points** of intervals in a framing. There are several ways how this property can be expressed. First, it can be specified in terms of the boundaries of a preceding interval. Second, it can be specified as a data-dependent condition over the tuples in the stream, for example a local minimum, a value above a threshold, or the observation of a certain value or punctuation (e.g. a “synchronization pulse”). Finally, the starting point of a new interval can be triggered w.r.t. to external event such as a wall-clock. Expressions combining all of the above using and are also permitted.

Global conditions are properties of the framing as a whole. For example, framings can be constrained by requiring a certain **framing scheme** that defines how the stream is segmented. Figure 2 summarizes important framing schemes.

1. **Partition.** Each interval begins where the previous frame ended, i.e. . Note that this segmentation of the stream is similar to tumbling windows. However, intervals still vary in length, whereas windows are of fixed length, either in terms of time or tuples.
2. **Cover.** , where is the progressing attribute, i.e. . Note that this segmentation of the stream is similar to sliding windows. However, neither the slide offset nor the interval length is fixed. A special case of this segmentation is the case where intervals overlap by fixed value , i.e. . We will refer to this stream segmentation as **adjacent**. Another special case are **advancing** intervals, where we require that .
3. **Disjoint.** , i.e. . Note that with this segmentation it is possible that some values , where is the progressing attribute, are not contained in any intervals.
4. **Unconstrained.**



Figure : Different framing schemes.

Note that some of the local properties imply global properties. For example, requiring intervals to be maximal at the local level will always lead to a disjoint framing scheme as overlapping intervals can be combined into a single interval. Similarly, certain constraints on the starting points of interval will also directly yield a certain framing scheme. For example, to specify an adjacent segmentation, the starting point condition would be defined as , while the condition yields a partitioning of the stream. As a consequence, care has to be taken in order not to specify conflicting conditions.

Furthermore, we allowing multiple possible interval starting points using a complex expression together with a non-overlapping framing scheme, the situation may arise where a starting point occurs *before* the ending point of the preceding interval. We use policies to indicate which condition will be violated by the framing if such a situation arises. There are three possibilities.

1. **Close and report the preceding interval:** This policy will lead to a framing that possibly violates the local condition defined by the predicate.
2. **Drop and suppress the previous interval:** This policy possibly leads to false negatives in terms of the local condition defined by the predicate. It will also possibly violate the global condition of a covering (or partitioning) framing scheme.
3. **Continue the previous interval:** This policy does not violate any of the local conditions, but possibly violates a requirement to generate a non-overlapping framing scheme.

## Framing Function

A framing as defined in the previous section is a function that returns a sequence of frame intervals for a given data stream. However, in order to process the data stream as it progresses, we require that this function can be computed incrementally, based on the previously computed frame intervals and the next tuple of the stream.

**Definition:** A *framing function* applies the th tuple to the current framing and returns a new framing . The framing is an extension of the framing , i.e. and denote the respective state before and after application of for the th tuple.

manages both the internal state of the framing function and the list of frames that are currently being processed, i.e. the *open* frames. For every arriving tuple, the framing function needs to decide whether (a) to open a new frame, (b) update an existing frame, or (c) close and emit a frame. In the latter case, the frame is removed from state and inserted into the framing . Depending on the framing scheme, all of these actions can be applied once or multiple times. In order to control the behavior of a framing function, we have identified the following frame properties.

We conclude this subsection by outlining an example template of a framing function that generates partitioning frames.

**partition**  
1: 2: **if** **then**3:   
4:   
5: **else**  
6:   
7:   
8: **end**  
9:   
10:

The function template begins by extracting the current frame from the state . Note that there is only one open frame in the state as this template follows the partition scheme. On line 2, the predicate is evaluated. If it still holds, the current frame is extended (lines 3 and 4). If not, the current frame is emitted (line 6) and a new frame is opened (line 7). Finally, the state is updated to reflect the updated or new frame (lines 9 and 10). The function is used to return the value of the progressing attribute of a tuple.

## Filling Frames

Once a framing of a stream has been defined in terms of a sequence of intervals, it can be used to fill frames. A framing that has computed based on one stream can be used to produce frames of that stream or any other stream. A frame filling is given as a set of tuple sequences .

**Definition:** A frame filling is defined by a function , where is the data stream furnishing the tuples, is the framing, and is a query.

The function allows intervals in the framing to be modified. For example, the function could be used to extend or reduce the lengths of intervals by a fixed amount. In particular, it can be used to produce adjacent frames. In contrast, the query defines a subset of the tuples that fall into the interval of the frame, or performs aggregation.

## Examples

Coming back to the use cases presented in Section 2, we now describe a series of types of frames that address the requirements outlined earlier. We also discuss how each of these types can be specified within the framework introduced in this section.

**Delta Frames:** This type of frame monitors a user-specified attribute of the tuples in the data stream. A frame is emitted whenever the delta between the minimum and maximum value of this attribute becomes greater than a predefined value . This type of frame partitions the stream. The frame predicate can therefore be given as . The frame starting point is the end point of the previous frame and the span of the frames is maximal.

**Threshold Frames:** This type of frame reports periods of the stream where the value of a user-defined attribute is greater (or smaller) than a given threshold value . As a consequence, this type of frame does not partition or cover the stream. It is defined by the frame predicate (or ). The start point is data dependent, i.e. a frame starts as soon as the predicate is true. The span is maximal in the sense that the frame keeps growing as long as the predicate holds true. An example of use for this type of frame is to report high-loss periods.

**Boundary Frames:** Boundary frames segment the stream whenever the value of a user-specified attribute crosses a (multiple of) a given boundary . This type of frames partitions the stream. The frame predicate for the nth frame is given by , whereas the data-dependent starting point is defined by . The span is minimal. Note that if attribute is a progressing attribute, this equivalent to windows.

**Aggregate Frames:** This type of frames monitor a predicate over an aggregation of an attribute. Aggregate frames segment the stream. The frame predicate is given by , where is an aggregation function and is a comparison operator. The start point of aggregate frames is the end point of the previous frame and its span can either be minimal or maximal, depending on the desired reporting scheme. An example of this type of frame is segmenting on the maximum sum of dye mass.

**Density Frames**

**Router-Drop Frames**

# IMPLEMENTATION

*Sharmada/Michael*

# EXPERIMENTS

*Sharmada*

# RELATED WORK

*Dave*

# CONCLUSION

*Kristin?*

# ACKNOWLEDGMENTS

Michael Grossniklaus’ work is funded by the Swiss National Science Foundation (SNSF) grant number PA00P2\_131452.

# REFERENCES

1. The Frog, K and Piggy, M. It’s time to play the music. *The Muppets Journal* (Jun. 1976), 123-456.

Columns on Last Page Should Be Made As Close As Possible to Equal Length

1. The names and stand for start and end point, respectively. [↑](#footnote-ref-1)