Queries. ACHTUNG NICHT FERTIG!

Thure Nebendahl

May 24, 2020

After the last chapter covered in great detail how the data stream is processed, this chapter has a closer look on the temporal queries (TQs) used for this investigation of the implementation of the bounded history encoding from [1]. It will further discuss if the queries specified here are expressible in the language from [1].

1 What Queries are used for the investigation?

1.1 Practical TQs

There are two different kinds of TQs, practical and meaningful TQs as well as randomized TQs build on the idea presented in [2]. The practical TQs show if the bounded history encoding from [1] can give relevant answers to queries. This is necessary to find out if it has practical relevance and can be used for example to observe markets as done in this work. Practical TQs can further be distinguished into database specific, e.g., "BY HOW MUCH DOES THE NUMBER OF OBSERVED OFFERS CHANGE?", car specific, e.g., "Which cars cost less than 10,000 at the last point in time and cost more than 10,000 at this point in time?" and brand or model specific queries, e.g., "Which brands (or models) have an increased average price compared to the last point in time?". For the sake of readability, only selected queries of each distinction are discussed below as examples. THIS DISTINCTION IS MAINLY FOR THE PURPOSE OF LATER REFERENCING .

1.2 Randomized TQs

The randomized TQs are used to verify the impact of the queries on the overall research result. They are necessary because the practical TQs might not be sufficiently diverse to yield reliable results. Randomized TQs can be constructed in any given size.

Definition 1.1 (randomized TQs similar to [2]). Let $n \in \mathbb{N} \setminus \{0\}$. Let Ops be a set of available operators. A randomized TQ of size n can be constructed by recursively constructing its subqueries as follows:

```
randomTQ($n$)
1 if $n = 1$ then
2   return $\mathcal{Q}$ – query $\psi$
3 else
4   select a random operator $op$
5   if $op$ is unary then
6     return $op$(randomTQ($n - 1$))
7   else // $op$ is binary
8     select a random $m \in \mathbb{N} \setminus \{0\}, m < n - 1$
9     return (randomTQ($m$) $op$ randomTQ($n - m - 1$))
```

2 Are these queries expressible in the language from [1]?

2.1 Practical TQs

Because the attribute ”url” is distinct for each dataset it will serve as an identifier for a car. Thus, when the query asks for cars it will actually return a set of urls. For example the previously mentioned car specific query can be translated into the language of [1] as follows:

- ”[. . .] and [. . .]” indicates the ”conjunction” operator from [1]
- ”[. . .] at the last point in time [. . .]” indicates either a ”strong previous” or a ”weak previous” operator from [1]
- ”Which cars cost less than 10,000 [. . .]” indicates the SQL-query ”SELECT *url* FROM *autos* WHERE *price* $<$ 10000”
- ”[. . .] more than 10,000 [. . .]” indicates the SQL-query ”SELECT *url* FROM *autos* WHERE *price* $>$ 10000”

As mentioned before, ”autos” is the table in which all offers are stored. Thus, a resulting query in the language from [1] is ”$\bigcirc^-$(SELECT *url* FROM *autos* WHERE *price* $<$ 10000) $\wedge$ ”SELECT *url* FROM *autos* WHERE *price* $>$ 10000”.

THIS CONSTRUCTION IS NOT ALWAYS SO STRAIGHTFORWARD. The brand specific query can be rewritten as follows:

- "[...] compared to the last point in time?" indicates the use of the "conjunction" operator from [1] with either a "strong previous" or a "weak previous" operator as one argument from [1]

- Since the query needs to compare the value of one attribute at two different points in time it needs a restriction like "[...] WHERE *price*1 < *price*2" with *price*1 being the average price of the first point in time and *price*2 the average price of the second point in time.

Thus, a resulting query might be "$\bigcirc^-$(SELECT *marke*, (AVG(*price*) AS *price*1) FROM *autos*) $\wedge$ "SELECT *marke*, (AVG(*price*) AS *price*2) FROM *autos* WHERE *price*1 < *price*2". Notice that this kind of restriction is not considered in [1] and therefore is also not considered in the implementation.

This also applies to the operators $\Box\phi$ (always), $\Box^-\phi$ (always in the past), $\Diamond\phi$ (eventually), $\Diamond^-\phi$ (some time in the past) that were presented in the paper but not considered when defining the algorithm and are therefore also not considered in the implementation.

Lastly a query of the type "Which cars cost more than 10.000 some time in the last 6 points in time?" is not expressible in the language from [1]. Such a query would need a temporal limitation in the form of numerical predicates.

Consequently, it is necessary to expand the implementation of the bounded history encoding from [1] by the missing operators. Subsequently, it can be checked whether cross-referencing of query-variables is supported and if the language from [1] can be expanded by numerical predicates.

2.2 Randomized TQs

Since only a predefined set of operators is used in the construction of randomized TQs, it is obvious that any randomized TQ can be expressed in any desired language, i.e., the language from [1], if Ops is exactly the set of operators available there. In the following $\mathcal{Q}-$query ψ will be defined as the SQL query "SELECT * FROM autos", which returns every offer without any restriction.

Some claim [1]

Definition 2.1 (size of TQs)**.** Let $\mathcal{Q}$ be a query language and ϕ be a TQ. The size of queries is defined by induction on the structure of ϕ as follows:

ϕ	$\mathsf{sz}(\phi)$
$\mathcal{Q}$ − query ψ	1
$\bigcirc\phi_1, \bigcirc^-\phi_1, \bullet\phi_1, \bullet^-\phi_1, \Box\phi_1, \Box^-\phi_1, \Diamond\phi_1, \Diamond^-\phi_1$	$1 + \mathsf{sz}(\phi_1)$
$\phi_1 \wedge \phi_2, \phi_1 \vee \phi_2, \phi_1 \mathsf{U} \phi_2, \phi_1 \mathsf{S} \phi_2$	$\mathsf{sz}(\phi_1) + \mathsf{sz}(\phi_2)$

Table 1: size of TQs

References

[1] Stefan Borgwardt, Marcel Lippmann, and Veronika Thost. Temporalizing rewritable query languages over knowledge bases. *Journal of Web Semantics*, 33:50–70, 2015.

[2] Joshua Schneider, David Basin, Sran Krstić, and Dmitriy Traytel. A formally verified monitor for metric first-order temporal logic. In *International Conference on Runtime Verification*, pages 310–328. Springer, 2019.