Real-Time Monitoring Enhanced by Automata-based Skipping

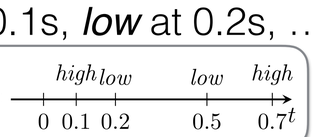
Masaki Waga, SOKENDAI / NII / ERATO MMSD project

Timed Pattern Matching

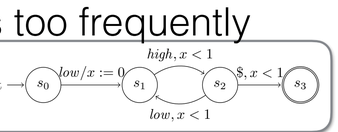
[Ulus+, FORMATS'14]

Input

- **Time-series data** (Logs of a car/ a robot)
  e.g., The gear of a car is *high* at 0.1s, *low* at 0.2s, …

Timed Words (Sequence of events with timestamps) $w =$ 

- **Real-time spec.** (Spec. useful for debugging)
  e.g., The gear of a car changes too frequently

Timed Automata [Alur & Dill, TCS'94] $\mathcal{A} =$ 

Output

- The intervals where the spec. is satisfied in the data
  e.g., The gear changes too frequently in 0.2s-0.7s

$$\mathcal{M}(w, \mathcal{A}) = \{(t, t') \mid w|_{(t,t')} \in L(\mathcal{A})\} = \{(t, t') \mid 0.2 \leq t < 0.5, 0.7 < t'\}$$

State of the Art

- There is a tool Montre [Ulus, CAV'17]
- However it has the following issues.
  - The offline algorithm [Ulus+, FORMATS'14] requires **huge RAM**.
    - linear to the size of the log
  - The online algorithm [Ulus+, TACAS'16] is **slow**.

Our Solution

Idea from string matching

- Construct an **automata-based** algorithm
  - [Waga+, FORMATS'16]
- Enhance the automata-based algorithm by ***Skipping***
  - [Waga+, FORMATS'16], [Waga+, FORMATS'17]

Skipping for String Matching

Brute-Force Search

Find "STRING"(pattern)
from "STRONG_STRING"(target)

```
STRONG_STRING
STRING
 STRING
  ⋮
      STRING
```

KMP Search

[Knuth+, SIAM J. Comput. '77]

Match until the **3rd** char.

Table for **length 3** partial match

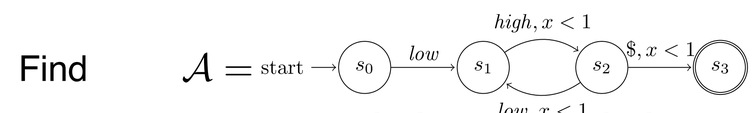
	S	T	R	:	I	N	G
✗	*	S	T	:	R	I	N
✗	*	*	S	:	T	R	I
✔	*	*	*	:	S	T	R

Skip 2 trials!!

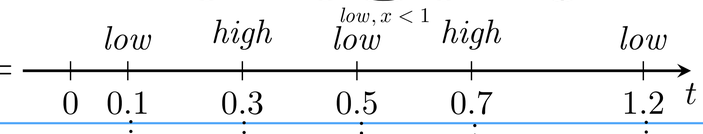
```
STRONG_STRING
STRING
  STRING
   STRING
    STRING
     ⋮
        STRING
```

Timed Pattern Matching with Skipping

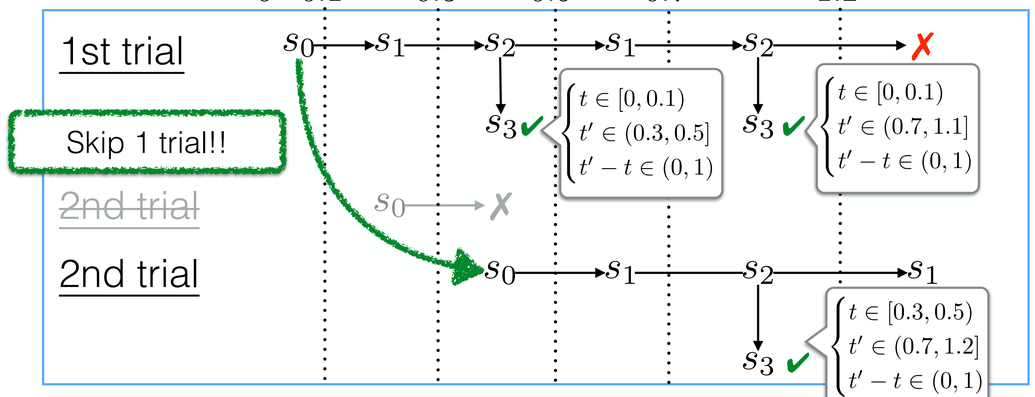
[Waga+, FORMATS'16], [Waga+, FORMATS'17]

Find $\mathcal{A} =$ 

(pattern)

from $w =$ 

(target)



Adaptation for Timed Pattern Matching

- The length of partial match is unbounded.

String Matching

Table for **length 3** partial match

	S	T	R	:	I	N	G
✗	*	S	T	:	R	I	N
✗	*	*	S	:	T	R	I
✔	*	*	*	:	S	T	R

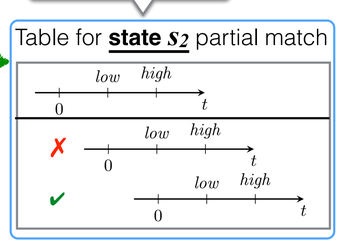
We construct for each length

Timed Pattern Matching

Infinitely Many Tables!!

Construct for each **state**

Table for **state s_2** partial match



- Infinitely many timestamps

String Matching

Length 3 partial match in string matching

S T R

Timed Pattern Matchin

Length 2 partial match in timed pattern matching

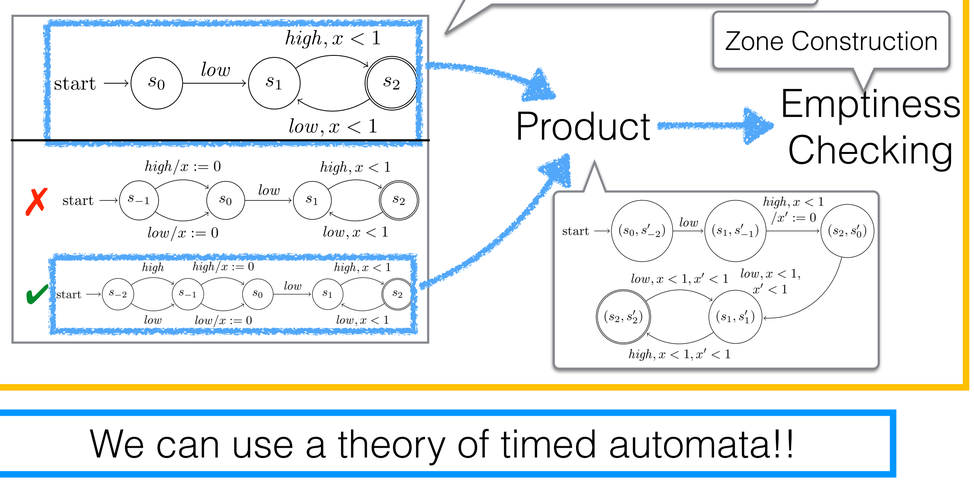
Infinitely many timestamps

Represent by a timed automaton

State s_2 partial match in timed pattern matching

Table for **state s_2** partial match

We want to compare them.



We can use a theory of timed automata!!

Implementation — MONAA —

- We implemented our algorithm as a tool "**MONAA**".
  - https://github.com/MasWag/monaa
- Input format: Timed Automata (TA) and Timed Regular Expressions (TRE)
- We also provide C++ API "**libmonaa**"

Evaluation

Table 3. Execution time (sec.)

Length of timed word	MONAA (TRE)	MONAA (TA)	libmonaa (TA is hard coded)	Montre (online)	Montre (offline)
708	7.03	0.80	0.20	0.13	0.03
218,247	7.55	1.27	0.31	37.45	1.56
436,611	8.05	1.73	0.42	75.93	3.13
655,237	8.54	2.21	0.53	115.88	4.69
870,967	9.16	2.69	0.64	153.71	6.21
1,087,411	9.53	3.14	0.75	189.55	7.75
1,304,404	10.05	3.60	0.85	216.92	9.33
1,527,632	10.53	4.06	0.97	260.77	10.88
1,739,525	11.05	4.56	1.07	289.63	12.39

Efficient and online

Blow up!!

Efficient but offline only

Table 4. Memory usage (kbytes)

Length of timed word	MONAA (TRE)	MONAA (TA)	libmonaa (TA is hard coded)	Montre (offline)
708	16,468	10,808	7,308	27,456
218,247	16,312	10,808	7,464	45,700
436,611	16,312	10,752	7,308	65,764
655,237	16,344	10,692	7,308	87,928
870,967	16,468	10,840	7,288	99,540
1,087,411	16,280	10,900	7,452	109,076
1,304,404	16,340	10,768	7,292	147,048
1,527,632	16,468	10,696	7,440	153,992
1,739,525	16,312	10,808	7,288	166,660

constant!!

Blow up!!

連絡先: 和賀 正樹 国立情報学研究所 アーキテクチャ科学研究系
TEL：03-6273-4886 FAX：03-6273-4886 Email：mwaga@nii.ac.jp