Stream Reasoning with Answer Set Programming

— Preliminary Report —

M. Gebser T. Grote R. Kaminski P. Obermeier*
O. Sabuncu T. Schaub

University of Potsdam, Germany

*DERI Galway, Ireland

Outline

- 1 Introduction
- 2 Reactive ASP
- 3 Stream Reasoning
- 4 Outlook

Outline

1 Introduction

- 2 Reactive ASF
- 3 Stream Reasoning
- 4 Outlool

Motivation

- Answer Set Programming (ASP) integrates
 - 1 Concise high-level (first-order) specification
 - 2 Powerful low-level (propositional) reasoning
- ➡ Two-phase computations
 - Instantiation

(gringo)

2 Search

(clasp)

- Stream processing concerned with
 - 1 High-throughput
 - 2 Continuous data
- How use ASP for knowledge-intense stream reasoning?
 - Handle emerging/expiring data upon instantiation
 - 2 Reuse previously learned information upon search
- Knowledge-intense stream reasoning can benefit from
 - High automation
 - Computational power

offered by ASP systems

Motivation

- Answer Set Programming (ASP) integrates
 - 1 Concise high-level (first-order) specification
 - 2 Powerful low-level (propositional) reasoning
- Two-phase computations
 - Instantiation

(gringo)

2 Search

(clasp)

- Stream processing concerned with
 - 1 High-throughput
 - 2 Continuous data
- How use ASP for knowledge-intense stream reasoning?
 - 1 Handle emerging/expiring data upon instantiation
 - 2 Reuse previously learned information upon search
- Knowledge-intense stream reasoning can benefit from
 - 1 High automation
 - 2 Computational power

offered by ASP systems

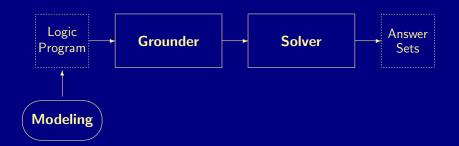
Outline

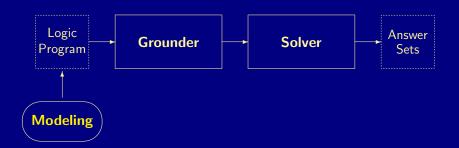
- 1 Introduction
- 2 Reactive ASP
- 3 Stream Reasoning
- 4 Outlook

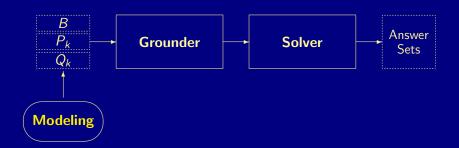
oclingo

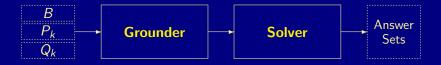
- Reactive grounding and solving
- Online solving in dynamic domains, like Robotics
- Basic architecture of *oclingo*:



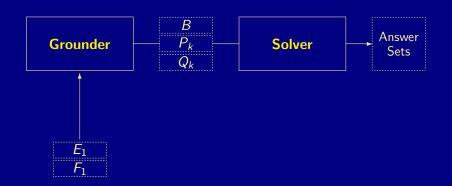


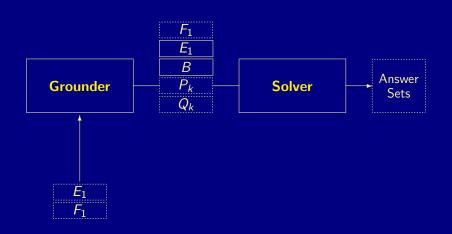


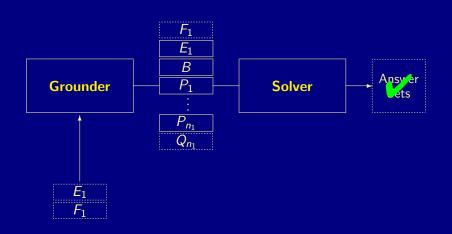


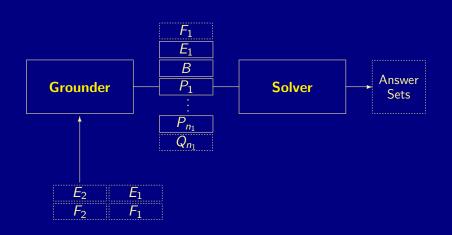


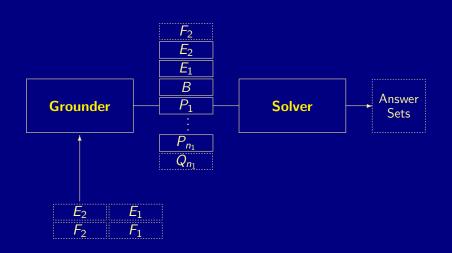


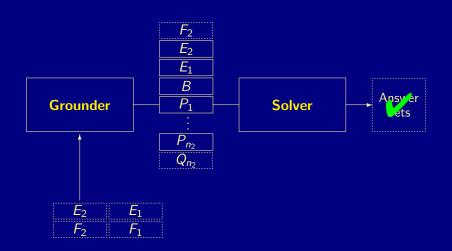


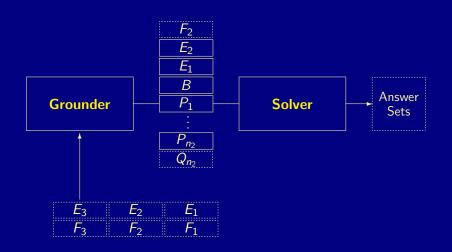


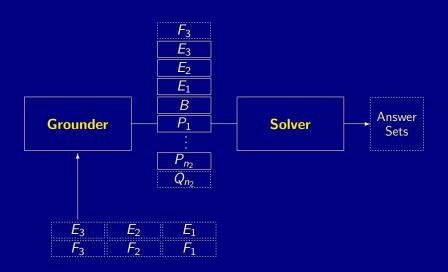


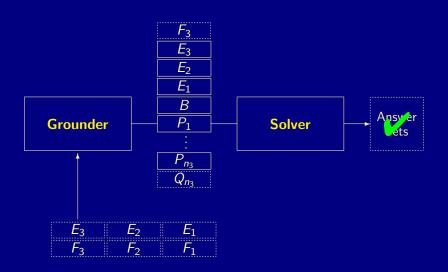


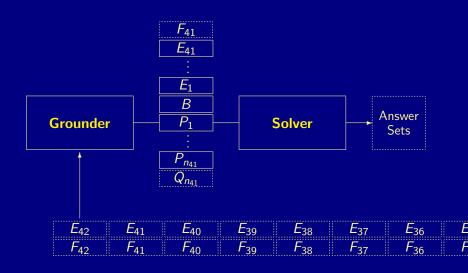


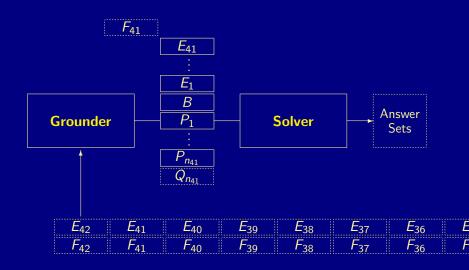


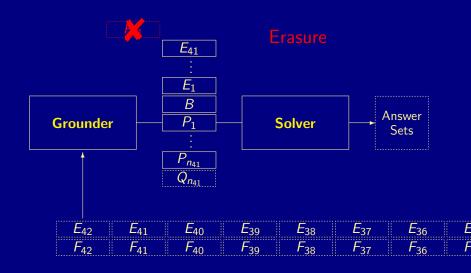


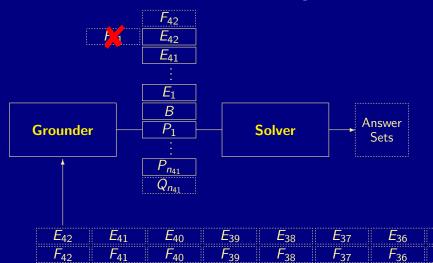


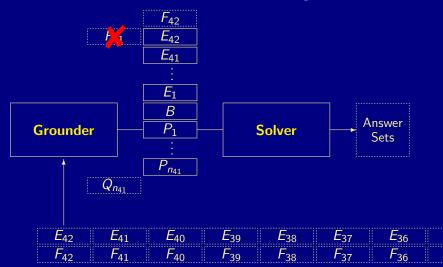


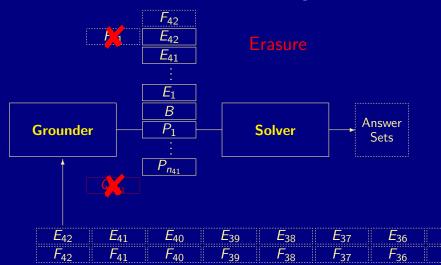


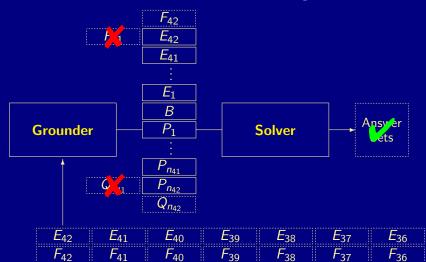












Outline

1 Introduction

- 2 Reactive ASP
- 3 Stream Reasoning
- 4 Outlool

Running Example

- Continuous data flow
- Reasoning over recent data
- Focus on time window

Running Example

- Continuous data flow
- Reasoning over recent data
- Focus on time window

Running Example

Position	1	2	3	
Reading	а		b	
	×			

- Continuous data flow
- Reasoning over recent data
- Focus on time window

Running Example

Position	1	2	3	
Reading	a	а	b	
		V		

- Continuous data flow
- Reasoning over recent data
- Focus on time window

Running Example

- Continuous data flow
- Reasoning over recent data
- Focus on time window

Running Example

- Continuous data flow
- Reasoning over recent data
- Focus on time window

Traditional ASP

Running Example

Position	2	3	
Reading		b	

```
Encoding
accept :-
    read(a,1).
    read(a,2).
    read(a,7-1).
    read(b,3).
    accept :- read(a,1), read(a,0).
    read(a,2).
    accept :- read(a,2), read(a,1).
    read(b,3).
    accept :- read(a,3), read(a,2).
```

Running Example

■ Continuously recognize strings matching: (a|b)*aa

Position	1	2	3	
Reading	a		b	
	×			

Encoding

```
accept :-
```

Data

Instantiation

read(a,1). accept :- read(a,1), read(a,0). read(a,T), read(a,2). accept := read(a,2), read(a,1). read(a,T-1). read(b,3). accept :- read(a,3), read(a,2).

Running Example

■ Continuously recognize strings matching: $(a|b)^*aa$

Position	1	2	3	
Reading	a	а	b	
		V		

Encoding

accept :-

Data

Instantiation

read(a,1). accept :- read(a,1), read(a,0). read(a,T), read(a,2). accept := read(a,2), read(a,1). read(a,T-1). read(b,3). accept :- read(a,3), read(a,2).

Running Example

■ Continuously recognize strings matching: (a|b)*aa

Position	1	2	3	
Reading	а	а	Ь	
		1		

Encoding

```
accept :-
```

Data

Instantiation

read(a,1). accept :- read(a,1), read(a,0). read(a,T), read(a,2). accept :- read(a,2), read(a,1). read(a,T-1). read(b,3). accept :- read(a,3), read(a,2).

Running Example

■ Continuously recognize strings matching: $(a|b)^*aa$

Position	1	2	3	
Reading	а	а	b	

```
Encoding Data Instantiation

accept :- read(a,1). accept :- read(a,2), read(a,1).
    read(a,T-1), read(b,3). accept :- read(a,3), read(a,2).
not read(a;b,T+1).
```

Running Example

not read(a;b,T+1).

■ Continuously recognize strings matching: $(a|b)^*aa$

Position	1	2	3	
Reading	а	а	Ь	
			X	

```
Encoding
                 Data
accept :-
                 read(a,1).
   read(a,T), read(a,2).
```

```
accept :- read(a,2), read(a,1),
                                     not read(a;b,3).
read(a,T-1), read(b,3), accept :- read(a,3), read(a,2),
```

not read(a;b,4).

Instantiation

Running Example

■ Continuously recognize strings matching: $(a|b)^*aa$

Position	1	2	3	
Reading	а	а	Ь	

```
Encoding

accept :- read(a,1).
    read(a,T), read(a,2).
    read(a,T-1), read(b,3).

not read(a;b,T+1). ...
```

```
Instantiation
```

Running Example

■ Continuously recognize strings matching: $(a|b)^*aa$

Position	1	2	3	
Reading	а	а	Ь	
	×	V	X	

```
Instantiation
```

Running Example

■ Continuously recognize strings matching: $(a|b)^*aa$

Position	1	2	3	
Reading	а	а	Ь	

Encoding[t]

```
accept :- read(a,1).
    read(a,t), read(a,2).
    read(a,t-1), read(b,3).
not read(a;b,t+1).
```

E-Data

```
Instantiation
```

Running Example

■ Continuously recognize strings matching: $(a|b)^*aa$

Position	2	3	
Reading		b	

Encoding[t]

```
accept :-
   read(a,t),
   read(a,t-1).
```

F-Data

```
read(a,1)
read(a,2)
read(b,3)
```

Instantiation

```
accept :- read(a,1), read(a,0).
accept :- read(a,2), read(a,1).
accept :- read(a,3), read(a,2).
```

Running Example

■ Continuously recognize strings matching: $(a|b)^*aa$

Position	1	3	
Reading	а	b	
	X		

Running Example

■ Continuously recognize strings matching: $(a|b)^*aa$

Position	1	2	3	
Reading	а	а	b	
		×		

```
Encoding[t] F-Data Instantiation
accept :- read(a,1). accept :- read(a,0).
read(a,t), read(a,2). accept :- read(a,2), read(a,1).
read(a,t-1). read(b,3). accept :- read(a,3), read(a,2).
```

Running Example

■ Continuously recognize strings matching: $(a|b)^*aa$

Position	1	3	
Reading	а	b	
	X		

Running Example

■ Continuously recognize strings matching: $(a|b)^*aa$

Position	1	2	3	
Reading	а	а	b	

Running Example

■ Continuously recognize strings matching: $(a|b)^*aa$

Position	1	2	3	
Reading	а	а	b	
		V		

Running Example

■ Continuously recognize strings matching: $(a|b)^*aa$

Position	1	2	3	
Reading	а	а	b	

```
Encoding[t] F-Data Instantiation

accept :- read(a,1). accept :- read(a,0).
   read(a,t), read(a,2). accept :- read(a,2), read(a,1).
   read(a,t-1). read(b,3). accept :- read(a,3), read(a,2).
```

Running Example

■ Continuously recognize strings matching: $(a|b)^*aa$

Position	1	2	3	
Reading	а	а	Ь	
			X	

Running Example

■ Continuously recognize strings matching: $(a|b)^*aa$

Position	1	2	3	
Reading	а	а	b	

```
Encoding[t] F-Data Instantiation

accept :- read(a,1). accept :- read(a,0).
    read(a,t), read(a,2). accept :- read(a,2), read(a,1).
    read(b,3). accept :- read(a,3), read(a,2).
    ...
```

Data must be replayed!

Running Example

■ Continuously recognize strings matching: $(a|b)^*aa$

Position	1	2	3	
Reading	а	а	Ь	

```
Encoding[t] F-Data Instantiation

accept :- read(a,1). accept :- read(a,0).
   read(a,t), read(a,2). accept :- read(a,2), read(a,1).
   read(a,t-1). read(b,3). accept :- read(a,3), read(a,2).
```

X Data must be replayed!

Running Example

■ Continuously recognize strings matching: $(a|b)^*aa$

Position	1	2	3	
Reading	а	а	b	
	X	1	X	

```
Encoding[t]
```

```
accept :-
   read(a,t), read(a,2).
   read(a,t-1).
```

F-Data

```
read(a,1).
read(b,3).
```

Instantiation

```
accept :- read(a,1), read(a,0).
 accept :- read(a,2), read(a,1).
accept :- read(a,3), read(a,2).
```

Data must be replayed !

Running Example

■ Continuously recognize strings matching: $(a|b)^*aa$

Position	1	2	3	
Reading	а	а	Ь	

Encoding[t]

```
accept :-
   read(a,t),
   read(a,t-1).
```

F:L-Data

```
read(a,1);
read(a,2);
read(b,3);
```

Instantiation

```
accept :- read(a,1), read(a,0).
accept :- read(a,2), read(a,1).
accept :- read(a,3), read(a,2).
```

Data must be replayed

Running Example

■ Continuously recognize strings matching: $(a|b)^*aa$

Position	1	2	3	
Reading	а		b	
	X			

```
Encoding[t] F:2-Data Instantiation accept :- read(a,1), read(a,0). read(a,t), read(a,2). accept :- read(a,2), read(a,1). read(b,3). ...
```

Data must be replayed !

Running Example

■ Continuously recognize strings matching: $(a|b)^*aa$

Position	1	2	3	
Reading	а	а	b	
		V		

```
Encoding[t] F:2	ext{-Data} Instantiation accept :- read(a,1). read(a,0). read(a,t), read(a,2). accept :- read(a,2), read(a,1). read(b,3). accept :- read(a,3), read(a,2).
```

Data must be replayed

Running Example

■ Continuously recognize strings matching: $(a|b)^*aa$

Position	1	2	3	
Reading	а	а	Ь	
			X	

Data must be replayed!

Running Example

■ Continuously recognize strings matching: $(a|b)^*aa$

Position	1	3	
Reading	а	b	

■ Seamless integration of online data with offline encoding!

Outline

1 Introduction

- 2 Reactive ASP
- 3 Stream Reasoning
- 4 Outlook

Discussion

- We provide
 - 1 Language
 - 2 Semantics
 - 3 Modeling techniques
 - 4 (Prototype) implementation

for transparent grounding and solving of (time-decaying) logic programs w.r.t. emerging/expiring stream data

- Future work addresses
 - 1 Data cleansing
 - 2 Query answering
 - 3 Optimization support
- → ASP technology for knowledge-intense stream reasoning
 - Application areas include
 - 1 Ambient Assisted Living
 - 2 Robotics Planning and Control
 - 3 Policy Monitoring and Enforcement
 - 4 etc.