Multi-Context Stream Reasoning

KI 2020

Stefan Ellmauthaler Konstantin Schekotihin

UNIVERSITÄT LEIPZIG

ALPEN-ADRIA UNIVERSITÄT KLAGENFURT | WIEN GRAZ

ellmauthaler@informatik.uni-leipzig.de, konstantin.schekotihin@aau.at

Bamberg, Germany, September 21, 2020

Austrian Research Promotion Agency (FFG), 861263

German Research Foundation (DFG), BR - 1817/7 - 1/2, FOR 1513

Funded by DFG Deutsche Forschungsgemeinschaft German Research Foundation

FFG

Acknowledgement

We like to kindly thank the following colleagues for fruitful discussions and contributions.

- Harald Beck
- Gerhard Brewka
- Minh Dao-Tran
- Carmine Dodaro
- Thomas Eiter
- Ricardo Gonçalves
- Matthias Knorr
- João Leite
- Paul Ogris
- Jörg Pührer
- Patrik Schneider

Agenda I

Agenda II

Outline

Introduction

- Connected digitised world
  - Mobile devices (phones, notebooks, . . .)
  - Electronic devices (fridges, stoves, TVs, doors, . . .)
  - Tools (CCTVs, warehouse parts, . . .)
  - “Things” (wares, items, bits, . . .)
- The World Wide Web Consortium (W3C) works on a standardisation

Introduction

- Connected digitised world
  - Mobile devices (pho tebooks
  - Electronic devices (stoves oors, . . .
  - Tools (CCTVs, ware parts, . . .
  - “Things” (wares, items, bits, . . .)
- The World Wide tium orks rdisation

Introduction

- Connected digitised world
  - Mobile devices (phones, notebooks, . . .)
  - Electronic devices (fridges, stoves, TVs, doors, . . .)
  - Tools (CCTVs, warehouse parts, . . .)
  - “Things” (wares, items, bits, . . .)
- The World Wide Web Consortium (W3C) works on a standardisation

Web of Things (WoT)

Introduction

- Connected digitised world
  - Mobile devices (phones, notebooks, . . .)
  - Electronic devices (fridges, stoves, TVs, doors, . . .)
  - Tools (CCTVs, warehouse parts, . . .)
  - “Things” (wares, items, bits, . . .)
- The World Wide Web Consortium (W3C) works on a standardisation

  Web of Things (WoT)

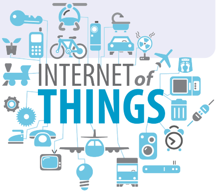
- Industry 4.0 & WoT depends heavily on the idea of
  Internet of Things (IoT)

Introduction

- Connected digitised world
  - Mobile devices (phones, notebooks, . . .)
  - Electronic devices (fridges, stoves, TVs, doors, . . .)
  - Tools (CCTVs, warehouse parts, . . .)
  - “Things” (wares, items, bits, . . .)
- The World Wide Web Consortium (W3C) works on a standardisation

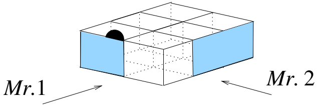
  Web of Things (WoT)

- Industry 4.0 & WoT depends heavily on the idea of **Internet of Things (IoT)**



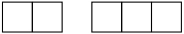
Contextual Reasoning

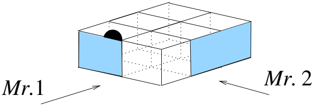
Ghidini & Giunchiglias magic box Ghidini and Giunchiglia (2001)



Contextual Reasoning

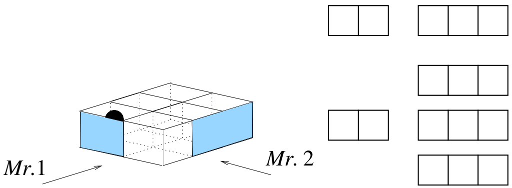
Ghidini & Giunchiglias magic box Ghidini and Giunchiglia (2001)





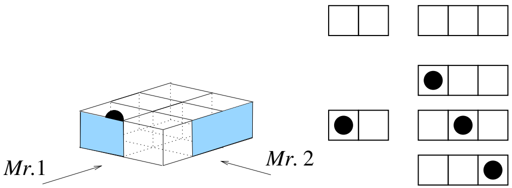
Contextual Reasoning

Ghidini & Giunchiglias magic box Ghidini and Giunchiglia (2001)



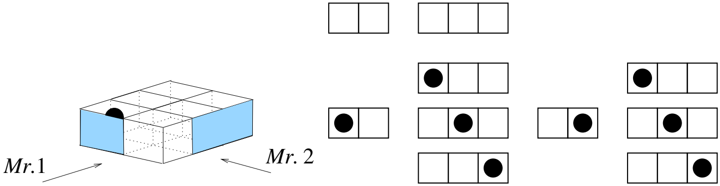
Contextual Reasoning

Ghidini & Giunchiglias magic box Ghidini and Giunchiglia (2001)



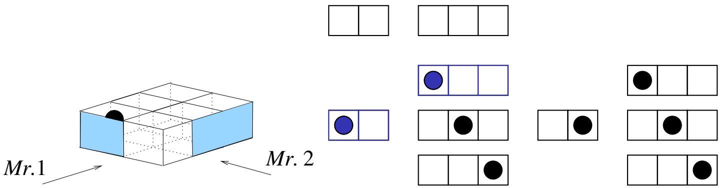
Contextual Reasoning

Ghidini & Giunchiglias magic box Ghidini and Giunchiglia (2001)



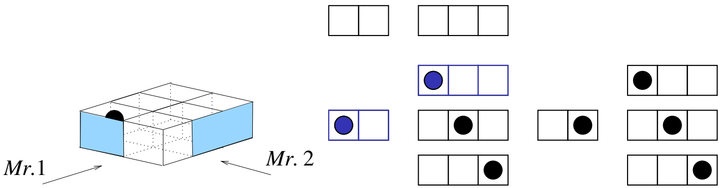
Contextual Reasoning

Ghidini & Giunchiglias magic box Ghidini and Giunchiglia (2001)



Contextual Reasoning

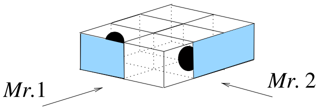
Ghidini & Giunchiglias magic box Ghidini and Giunchiglia (2001)



- **model** information
- **integrate** knowledge bases and context-based information
- **synchronise** knowledge, reasoning, and conclusions

Contextual Reasoning

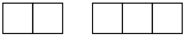
Ghidini & Giunchiglias magic box Ghidini and Giunchiglia (2001)

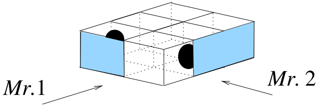


- **model** information
- **integrate** knowledge bases and context-based information
- **synchronise** knowledge, reasoning, and conclusions

Contextual Reasoning

Ghidini & Giunchiglias magic box Ghidini and Giunchiglia (2001)

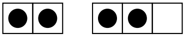


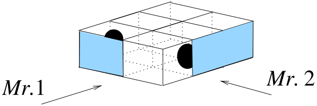


- **model** information
- **integrate** knowledge bases and context-based information
- **synchronise** knowledge, reasoning, and conclusions

Contextual Reasoning

Ghidini & Giunchiglias magic box Ghidini and Giunchiglia (2001)

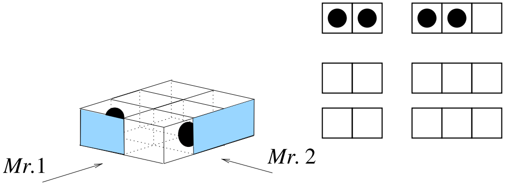




- **model** information
- **integrate** knowledge bases and context-based information
- **synchronise** knowledge, reasoning, and conclusions

Contextual Reasoning

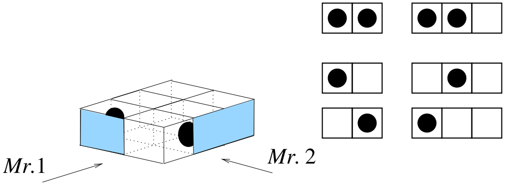
Ghidini & Giunchiglias magic box Ghidini and Giunchiglia (2001)



- **model** information
- **integrate** knowledge bases and context-based information
- **synchronise** knowledge, reasoning, and conclusions

Contextual Reasoning

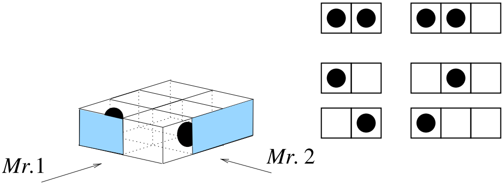
Ghidini & Giunchiglias magic box Ghidini and Giunchiglia (2001)



- **model** information
- **integrate** knowledge bases and context-based information
- **synchronise** knowledge, reasoning, and conclusions

Contextual Reasoning

Ghidini & Giunchiglias magic box Ghidini and Giunchiglia (2001)



- **model** information
- **integrate** knowledge bases and context-based information
- **synchronise** knowledge, reasoning, and conclusions
- **handle** non-determinism and non-mononotonic behaviour

Outline

Knowledge of Contexts

Represented by logics with various properties

- represent formalism of contexts
- monotone and non-monotone logic
- diverse truth-values
  - boolean
  - many valued
  - fuzzy values
- many different semantics

Knowledge of Contexts

Represented by logics with various properties

- represent formalism of contexts
- monotone and non-monotone logic
- diverse truth-values
  - boolean
  - many valued
  - fuzzy values
- many different semantics

}

- one formal structure

Logic
an Abstract Representation

- An abstract way to define a Logic
- Capable of realising monotone and non-monotone logics
- Representing different number of values
(e.g. binary, many valued, fuzzy values, ...)

Definition (Logic Brewka and Eiter (2007))

A logic is a triple $L = \langle KB, BS, \mathbf{acc} \rangle$, where

- KB is a set of knowledge bases,
- BS is a set of belief sets, and
- $\mathbf{acc} : KB \mapsto 2^{BS}$, the *acceptance function* is a function which assigns to each knowledge base a set of belief sets.

Logic
to Represent KRR-Formalisms

Description Logic

Answer Set Programming

Logic

to Represent KRR-Formalisms

Description Logic

- Decidable FO logic fragment
  - Concepts & Roles
  - $\mathcal{T}$Box & $\mathcal{A}$Box
- Monotone
- Many different versions ($\mathcal{AL}$, $\mathcal{ALC}$, $\mathcal{SHIF}$, $\mathcal{SROIC}$,...)
- Complexity around **EXPTime**

Answer Set Programming

Logic

to Represent KRR-Formalisms

Description Logic

- Decidable FO logic fragment
  - Concepts & Roles
  - $\mathcal{T}$Box & $\mathcal{A}$Box
- Monotone
- Many different versions ($\mathcal{AL}$, $\mathcal{ALC}$, $\mathcal{SHIF}$, $\mathcal{SROIC}$, ...)
- Complexity around **EXPTime**

Answer Set Programming

- Rule-Based KR formalism
  - Predicates, Default negation
  - Set of Rules
- Non-monotone
- Normal, disjunctive, negated ASP (w/ optimisation)
- Complexity ranging from **NP** to $\Sigma_3^{\mathbf{P}}$

Logic

to Represent KRR-Formalisms

Simple Storage Mechanism

Stores information as set of strings and replicates them without reasoning

Description Logic

- Decidable FO logic fragment
  - Concepts & Roles
  - $\mathcal{T}$Box & $\mathcal{A}$Box
- Monotone
- Many different versions ($\mathcal{AL}$, $\mathcal{ALC}$, $\mathcal{SHIF}$, $\mathcal{SROIC}$,...)
- Complexity around **EXPTime**

Answer Set Programming

- Rule-Based KR formalism
  - Predicates, Default negation
  - Set of Rules
- Non-monotone
- Normal, disjunctive, negated ASP (w/ optimisation)
- Complexity ranging from **NP** to $\Sigma_3^{\mathbf{P}}$

Logic
to Represent KRR-Formalisms

Simple Storage Mechanism

Stores information as set of strings and replicates them without reasoning

Description Logic

- Decidable FO logic fragment
  - Concepts & Roles
  - $\mathcal{T}$Box & $\mathcal{A}$Box
- Monotone
- Many different versions ($\mathcal{AL}$, $\mathcal{ALC}$, $\mathcal{SHIF}$, $\mathcal{SROIC}$, ...)
- Complexity around **EXPTime**

Answer Set Programming

- Rule-Based KR formalism
  - Predicates, Default negation
  - Set of Rules
- Non-monotone
- Normal, disjunctive, negated ASP (w/ optimisation)
- Complexity ranging from **NP** to $\Sigma_3^{\mathbf{P}}$

SAT-Problem in propositional logic

- Classical propositional Logic
- Modelling of one specific problem

Represent Storage Mechanism

Simple Storage Logic

$L_s = \langle KB_s, BS_s, \mathbf{acc}_s \rangle$

- Given a set E of entries
- $KB_s = BS_s = 2^E$
- $\mathbf{acc}_s$ maps every set $E' \subseteq 2^E$ to $\{E'\}$.

Represent KRR Formalisms

Description Logic $\mathcal{AL}$

$L_d = \langle KB_d, BS_d, \mathbf{acc}_d \rangle$

- KB_d are all ontologies
- BS_d is the set of deductively closed subsets in $\mathcal{AL}$
- $\mathbf{acc}_d$ is a mapping of $kb \in KB_d$ to $M \subseteq 2^{BS_d}$, s.t. $\forall_{m \in M} kb \models m$ holds.

Represent KRR Formalisms

Description Logic $\mathcal{AL}$

$L_d = \langle KB_d, BS_d, \mathbf{acc}_d \rangle$

- KB_d are all ontologies
- BS_d is the set of deductively closed subsets in $\mathcal{AL}$
- $\mathbf{acc}_d$ is a mapping of $kb \in KB_d$ to $M \subseteq 2^{BS_d}$, s.t. $\forall_{m \in M} kb \models m$ holds.

Answer Set Programming

$L_{asp} = \langle KB_{asp}, BS_{asp}, \mathbf{acc}_{asp} \rangle$

- Let A be the set of all possible ground atoms
- KB_{asp} is the set of all answer set programs over A.
- $BS_{asp} = 2^A$
- $\mathbf{acc}_{asp}$ maps each ASP program to its answer sets

Represent one KRR Problem

SAT in propositional logic

$L_p = \langle KB_p, BS_p, \mathbf{acc}_p \rangle$

- KB_p is the set of all well-formed formulae F with respect to the signature Σ in CNF.
- $BS_p = \{\{\top\}, \{\bot\}\}$ is the set of all possible answers “True” and “False”
- $\mathbf{acc}_p$ maps each formula $\sigma \in KB_d$ to $\top$ (resp. $\bot$), if σ is (un-)satisfiable...

Outline

Integrate Knowledge

Bridge Rules

Bridge Rule idea:

- **integrate** knowledge from other contexts
- only transfer **acceptable** information w.r.t. the origin context
- **utilise** integrated knowledge for reasoning

Integrate Knowledge

Bridge Rules

Bridge Rule idea:

- **integrate** knowledge from other contexts
- only transfer **acceptable** information w.r.t. the origin context
- **utilise** integrated knowledge for reasoning

Definition (Bridge Rule)

Let $L = \{L_1, \ldots, L_n\}$ be a set of logics.
An L_k-bridge rule over L, $1 \leq k \leq n$, is of the form

$$s \leftarrow (c_1 : p_1), \ldots, (c_j : p_j), not\ (c_{j+1} : p_{j+1}), \ldots, not\ (c_m : p_m)$$

where

- $\forall_{kb \in KB_k} : s \cup kb \in KB_k$ and
- for each $1 \leq l \leq m$ exists a logic $L_l \in (L_1, \ldots, L_n)$ such that $p_l \in B \in BS_l$ holds.

Multi-Context System (MCS) Brewka and Eiter (2007)

Definition

A Multi-Context System $M = (C_1, \ldots, C_n)$ is a collection of contexts $C_i = (L_i, kb_i, BR_i)$, where

- $L_i = (KB_i, BS_i, \mathbf{acc}_i)$ is an abstract logic,
- $kb_i \in KB_i$ is a knowledge base, and
- $BR_i = \{br_1, \ldots, br_m\}$ is a set of bridge rules over $\{L_1, \ldots, L_m\}$[1].

[1] note that L_i implies that it is defined in C_i

Example

Planner

Logic: L_{asp}

Consistency-Checker

Logic: L_p

Logger

Logic: L_s

Example

Planner

Logic: L_{asp}
Knowledge base:

$a \leftarrow not\ b.$

$b \leftarrow not\ a.$

$a \leftarrow exception.$

Consistency-Checker

Logic: L_p
Knowledge base:

$a \land \neg b$

Logger

Logic: L_s
Knowledge base:

$\emptyset$

Example

Planner

Logic: L_{asp}
Knowledge base:
$a \leftarrow not\ b.$
$b \leftarrow not\ a.$
$a \leftarrow exception.$
Bridge rules:
$exception \leftarrow c_3$: issue

Consistency-Checker

Logic: L_p
Knowledge base:
$a \wedge \neg b$

Bridge rules:
$a \leftarrow c_1 : a$
$b \leftarrow c_1 : b$

Logger

Logic: L_s
Knowledge base:
$\emptyset$

Bridge rules:
issue $\leftarrow c_2 : \bot$

Example

Planner

Logic: L_{asp}

Knowledge base:

$a \leftarrow not\ b.$

$b \leftarrow not\ a.$

$a \leftarrow exception.$

Bridge rules:

$exception \leftarrow c_3$: issue

Consistency-Checker

Logic: L_p

Knowledge base:

$a \wedge \neg b$

Bridge rules:

$a \leftarrow c_1 : a$

$b \leftarrow c_1 : b$

Logger

Logic: L_s

Knowledge base:

$\emptyset$

Bridge rules:

issue $\leftarrow c_2 : \bot$

How to compute the result of such an MCS?

Semantics

How to compute the result of an MCS

- Uniform representation of results

Semantics

How to compute the result of an MCS

- Uniform representation of results

Definition (Belief State)

Let $M = \langle C_1, \ldots, C_n \rangle$ be an MCS. A belief state B $= \langle B_1, \ldots, B_n \rangle$ is a sequence such that for each $1 \leq i \leq n$, $B_i \in BS_i$ holds.

Semantics

How to compute the result of an MCS

- Uniform representation of results

Definition (Belief State)

Let $M = \langle C_1, \ldots, C_n \rangle$ be an MCS. A belief state B $= \langle B_1, \ldots, B_n \rangle$ is a sequence such that for each $1 \leq i \leq n$, $B_i \in BS_i$ holds.

- Identify acceptable information due to the belief state

Semantics

How to compute the result of an MCS

- Uniform representation of results

Definition (Belief State)

Let $M = \langle C_1, \ldots, C_n \rangle$ be an MCS. A belief state B $= \langle B_1, \ldots, B_n \rangle$ is a sequence such that for each $1 \leq i \leq n$, $B_i \in BS_i$ holds.

- Identify acceptable information due to the belief state

Definition (Acceptable Rule)

Let $M = (C_1, \ldots, C_n)$ be an MCS and B $= \langle B_1, \ldots, B_n \rangle$ be a belief state for M. A bridge rule is acceptable w.r.t. B if

- for all $(c_i : p) \in$ body$^+(br) : p \in B_i$ and
- for all $(c_i : p) \in$ body$^-(br) : p \notin B_i$ holds.

We will use $app(R, \text{B})$ to represent all consequences of acceptable rules of R w.r.t. B.

Semantics

How to compute the result of an MCS

- Uniform representation of results

Definition (Belief State)

Let $M = \langle C_1, \ldots, C_n \rangle$ be an MCS. A belief state B $= \langle B_1, \ldots, B_n \rangle$ is a sequence such that for each $1 \leq i \leq n$, $B_i \in BS_i$ holds.

- Identify acceptable information due to the belief state

Definition (Acceptable Rule)

Let $M = (C_1, \ldots, C_n)$ be an MCS and B $= \langle B_1, \ldots, B_n \rangle$ be a belief state for M. A bridge rule is acceptable w.r.t. B if

- for all $(c_i : p) \in \text{body}^+(br) : p \in B_i$ and
- for all $(c_i : p) \in \text{body}^-(br) : p \notin B_i$ holds.

We will use $app(R, \text{B})$ to represent all consequences of acceptable rules of R w.r.t. B.

- Add acceptable rule consequences to associated context

Semantics

How to compute the result of an MCS

- Uniform representation of results

Definition (Belief State)

Let $M = \langle C_1, \ldots, C_n \rangle$ be an MCS. A belief state $\mathrm{B} = \langle B_1, \ldots, B_n \rangle$ is a sequence such that for each $1 \leq i \leq n$, $B_i \in BS_i$ holds.

- Identify acceptable information due to the belief state

Definition (Acceptable Rule)

Let $M = (C_1, \ldots, C_n)$ be an MCS and $\mathrm{B} = \langle B_1, \ldots, B_n \rangle$ be a belief state for M. A bridge rule is acceptable w.r.t. B if

- for all $(c_i : p) \in \mathrm{body}^+(br) : p \in B_i$ and
- for all $(c_i : p) \in \mathrm{body}^-(br) : p \notin B_i$ holds.

We will use $app(R, \mathrm{B})$ to represent all consequences of acceptable rules of R w.r.t. B.

- Add acceptable rule consequences to associated context , and

$\Rightarrow$ cope with cycle (KB update, belief state update, acceptable rule update)

Example

Planner - c_1

Logic: L_{asp}
Knowledge base:
$a \leftarrow not\ b.$
$b \leftarrow not\ a.$
$a \leftarrow exception.$
Bride rules:
$exception \leftarrow c_3 :$ issue

Consistency - c_2

Logic: L_p
Knowledge base:
$a \wedge \neg b$

Bridge rules:
$a \leftarrow c_1 : a$
$b \leftarrow c_1 : b$

Logger - c_3

Logic: L_s
Knowledge base:
$\emptyset$

Bridge rules:
issue $\leftarrow c_2 : \bot$

Example

Planner - c_1

Logic: L_{asp}
Knowledge base:
$a \leftarrow not\ b.$
$b \leftarrow not\ a.$
$a \leftarrow exception.$
Bride rules:
$exception \leftarrow c_3 :$ issue

Consistency - c_2

Logic: L_p
Knowledge base:
$a \wedge \neg b$

Bridge rules:
$a \leftarrow c_1 : a$
$b \leftarrow c_1 : b$

Logger - c_3

Logic: L_s
Knowledge base:
$\emptyset$

Bridge rules:
issue $\leftarrow c_2 : \bot$

Result 1

$\{a\}$

Result 2

$\{b\}$

Example

Planner - c_1

Logic: L_{asp}

Knowledge base:

$a \leftarrow not\ b.$

$b \leftarrow not\ a.$

$a \leftarrow exception.$

Bride rules:

$exception \leftarrow c_3 : \text{issue}$

Consistency - c_2

Logic: L_p

Knowledge base:

$a \wedge \neg b$

Bridge rules:

$a \leftarrow c_1 : a$

$b \leftarrow c_1 : b$

Logger - c_3

Logic: L_s

Knowledge base:

$\emptyset$

Bridge rules:

$\text{issue} \leftarrow c_2 : \bot$

Result 1

$\{a\}$

Result 2

$\{b\}$

Result 1

$\top$

Example

Planner - c_1

Logic: L_{asp}
Knowledge base:
$a \leftarrow not\ b.$
$b \leftarrow not\ a.$
$a \leftarrow exception.$
Bride rules:
$exception \leftarrow c_3 : \text{issue}$

Consistency - c_2

Logic: L_p
Knowledge base:
$a \wedge \neg b$

Bridge rules:
$a \leftarrow c_1 : a$
$b \leftarrow c_1 : b$

Logger - c_3

Logic: L_s
Knowledge base:
$\emptyset$

Bridge rules:
$\text{issue} \leftarrow c_2 : \bot$

Result 1

$\{a\}$

Result 2

$\{b\}$

Result 1

$\top$

Result 1

$\emptyset$

Example

Planner - c_1

Logic: L_{asp}

Knowledge base:

$a \leftarrow not\ b.$

$b \leftarrow not\ a.$

$a \leftarrow exception.$

Bride rules:

$exception \leftarrow c_3$: issue

Consistency - c_2

Logic: L_p

Knowledge base:

$a \wedge \neg b$

Bridge rules:

$a \leftarrow c_1 : a$

$b \leftarrow c_1 : b$

Logger - c_3

Logic: L_s

Knowledge base:

$\emptyset$

Bridge rules:

issue $\leftarrow c_2 : \bot$

Result 1

$\{a\}$

Result 2

$\{b\}$

Result 1

$\top$

Result 1

$\emptyset$

Example

Planner - c_1

Logic: L_{asp}
Knowledge base:
$a \leftarrow not\ b.$
$b \leftarrow not\ a.$
$a \leftarrow exception.$
Bride rules:
$exception \leftarrow c_3 :$ issue

Consistency - c_2

Logic: L_p
Knowledge base:
$a \wedge \neg b$

Bridge rules:
$a \leftarrow c_1 : a$
$b \leftarrow c_1 : b$

Logger - c_3

Logic: L_s
Knowledge base:
$\emptyset$

Bridge rules:
issue $\leftarrow c_2 : \bot$

Result 1

$\{a\}$

Result 2

$\{b\}$

Result 1

$\top$

Result 1

$\emptyset$

Example

Planner - c_1

Logic: L_{asp}

Knowledge base:

$a \leftarrow not\ b.$

$b \leftarrow not\ a.$

$a \leftarrow exception.$

Bride rules:

$exception \leftarrow c_3 : \text{issue}$

Consistency - c_2

Logic: L_p

Knowledge base:

$a \land \neg b$

Bridge rules:

$a \leftarrow c_1 : a$

$b \leftarrow c_1 : b$

Logger - c_3

Logic: L_s

Knowledge base:

$\emptyset$

Bridge rules:

$\text{issue} \leftarrow c_2 : \bot$

Result 1

$\{a\}$

Result 1

$\top$

Result 1

$\emptyset$

Result 2

$\{b\}$

Example

Planner - c_1

Logic: L_{asp}
Knowledge base:
$a \leftarrow not\ b.$
$b \leftarrow not\ a.$
$a \leftarrow exception.$
Bride rules:
$exception \leftarrow c_3 :$ issue

Result 1

$\{a\}$

Result 2

$\{b\}$

Consistency - c_2

Logic: L_p
Knowledge base:
$a \wedge \neg b$

Bridge rules:
$a \leftarrow c_1 : a$
$b \leftarrow c_1 : b$

Result 1

$\top$

Logger - c_3

Logic: L_s
Knowledge base:
$\emptyset$

Bridge rules:
issue $\leftarrow c_2 : \bot$

Result 1

$\emptyset$

Example

Planner - c_1

Logic: L_{asp}
Knowledge base:
$a \leftarrow not\ b.$
$b \leftarrow not\ a.$
$a \leftarrow exception.$
Bride rules:
$exception \leftarrow c_3$: issue

Consistency - c_2

Logic: L_p
Knowledge base:
$a \wedge \neg b$

Bridge rules:
$a \leftarrow c_1 : a$
$b \leftarrow c_1 : b$

Logger - c_3

Logic: L_s
Knowledge base:
$\emptyset$

Bridge rules:
issue $\leftarrow c_2 : \bot$

Result 1

$\{a\}$

Result 2

$\{b\}$

Revised result

$\bot$

Result 1

$\emptyset$

Example

Planner - c_1

Logic: L_{asp}
Knowledge base:
$a \leftarrow not\ b.$
$b \leftarrow not\ a.$
$a \leftarrow exception.$
Bride rules:
$exception \leftarrow c_3$: issue

Consistency - c_2

Logic: L_p
Knowledge base:
$a \wedge \neg b$

Bridge rules:
$a \leftarrow c_1 : a$
$b \leftarrow c_1 : b$

Logger - c_3

Logic: L_s
Knowledge base:
$\emptyset$

Bridge rules:
issue $\leftarrow c_2 : \bot$

Result 1

$\{a\}$

Revised result

$\bot$

Revised result

$\{'issue'\}$

Result 2

$\{b\}$

Example

Planner - c_1

Logic: L_{asp}
Knowledge base:
$a \leftarrow not\ b.$
$b \leftarrow not\ a.$
$a \leftarrow exception.$
Bride rules:
$exception \leftarrow c_3 :$ issue

Consistency - c_2

Logic: L_p
Knowledge base:
$a \wedge \neg b$

Bridge rules:
$a \leftarrow c_1 : a$
$b \leftarrow c_1 : b$

Logger - c_3

Logic: L_s
Knowledge base:
$\emptyset$

Bridge rules:
issue $\leftarrow c_2 : \bot$

Result 1

$\{a\}$

Result 2

$\{b\}$

Revised result

$\bot$

Revised result

$\{'issue'\}$

Example

Planner - c_1

Logic: L_{asp}
Knowledge base:
$a \leftarrow not\ b.$
$b \leftarrow not\ a.$
$a \leftarrow exception.$
Bride rules:
$exception \leftarrow c_3$: issue

Consistency - c_2

Logic: L_p
Knowledge base:
$a \wedge \neg b$

Bridge rules:
$a \leftarrow c_1 : a$
$b \leftarrow c_1 : b$

Logger - c_3

Logic: L_s
Knowledge base:
$\emptyset$

Bridge rules:
issue $\leftarrow c_2 : \bot$

Result 1

$\{a\}$

Revised result

$\bot$

Revised result

$\{'issue'\}$

Result 2

$\{b\}$

$\Rightarrow$ Fixpoint semantics

Example

Planner - c_1

Logic: L_{asp}
Knowledge base:
$a \leftarrow not\ b.$
$b \leftarrow not\ a.$
$a \leftarrow exception.$
Bride rules:
$exception \leftarrow c_3$: issue

Consistency - c_2

Logic: L_p
Knowledge base:
$a \wedge \neg b$

Bridge rules:
$a \leftarrow c_1 : a$
$b \leftarrow c_1 : b$

Logger - c_3

Logic: L_s
Knowledge base:
$\emptyset$

Bridge rules:
issue $\leftarrow c_2 : \bot$

Result 1

$\{a\}$

Revised result

$\bot$

Revised result

$\{'issue'\}$

Result 2

$\{b\}$

$\Rightarrow$ Fixpoint semantics

$\Rightarrow$ “Equilibria” semantics

Equilibria

Definition (Equilibrium)

Let $M = (C_1, \ldots, C_n)$ be an MCS and B $= \langle B_1, \ldots, B_n \rangle$ be a belief state for M. A belief state B for M is an equilibrium, if for each B_i in $1 \leq i \leq n$

$$B_i \in \mathbf{acc}_i(kb_i \cup app(BR_i, B))$$

holds.

Equilibria

Definition (Equilibrium)

Let $M = (C_1, \ldots, C_n)$ be an MCS and B $= \langle B_1, \ldots, B_n \rangle$ be a belief state for M. A belief state B for M is an equilibrium, if for each B_i in $1 \leq i \leq n$

$$B_i \in \mathbf{acc}_i(kb_i \cup app(BR_i, B))$$

holds.

Example (Solution)

Equilibrium: $\langle \{a\}, \{\top\}, \emptyset \rangle$

MCS Overview

- **Advantages**
  - abstract logic to **represent** various formalisms
  - simple structure of bridge rules to **integrate knowledge**
  - **strongly coupled semantics** due to the concept of equilibria
- **Shortcomings**
  - knowledge base has **monotone growth**
  - knowledge **cannot** be **revised**

Outline

Extending MCS

Conceptional Idea of managed Multi-Context Systems (mMCS) Brewka *et al.* (2011b)[2]

To tackle **monotone growth** and the **inability** to **revise** knowledge ...

- allow each Context to **manage** itself, to
- **add** knowledge (as before),
- **revise** knowledge, and
- allow to use **different semantics** per context

[2] further detailed in Weinzierl (2014)

Extending MCS

Conceptional Idea of managed Multi-Context Systems (mMCS) Brewka *et al.* (2011b)[2]

To tackle **monotone growth** and the **inability** to **revise** knowledge ...

- allow each Context to **manage** itself, to — **management function**
- **add** knowledge (as before),
- **revise** knowledge, and — **"management base"**
- allow to use **different semantics** per context — **logic suite**

[2] further detailed in Weinzierl (2014)

managed Multi-Context System

Basics

Definition (Logic Suite)

A logic suite $LS = (KB_{LS}, BS_{LS}, ACC_{LS})$ consists of the set BS_{LS} of possible belief sets, the set KB_{LS} of well-formed knowledge-bases, and a nonempty set ACC_{LS} of possible semantics of LS, i.e. $\mathbf{acc}_{LS} \in ACC_{LS}$ implies $\mathbf{acc}_{LS} : KB_{LS} \rightarrow 2^{BS_{LS}}$.

Definition (Management Base)

A set of names for operators is called a management base.

Definition (Management Function)

A management function over a logic suite LS and a management base OP is a function $mng : 2^{F^{OP}_{LS}} \times KB_{LS} \rightarrow 2^{KB_{LS} \times ACC_{LS}} \setminus \{\emptyset\}$.

managed Multi-Context System

Definition (Managed Multi-Context System)

A managed Multi-Context System M is a collection $(C_1, \dots, C_n)$ of managed contexts where, for $1 \leq i \leq n$, each managed context C_i is a quintuple $C_i = (LS_i, kb_i, br_i, OP_i, mng_i)$ such that

- $LS_i = (BS_{LS_i}, KB_{LS_i}, ACC_{LS_i})$ is a logic suite,
- $kb_i \in KB_{LS_i}$ is a knowledge base,
- OP_i is a management base,
- br_i is a set of bridge rules for C_i, with the form
  $$op_i \leftarrow (c_1 : p_1), \dots, (c_j : p_j), not(c_{j+1} : p_{j+1}), \dots, not(c_m : p_m).$$
  such that $op_i \in F^{OP_i}_{LS_i}$ and for all $1 \leq k \leq m$ there exists a context $c_k \in (C_1, \dots, C_n)$ such that $p_k \in B \in BS_{LS_{c_k}}$, and
- mng_i is a management function over LS_i and OP_i.

Definition (Equilibria for managed Multi-Context Systems)

Let $M = (C_1, \dots, C_n)$ be a managed multi-context system. A belief state $\mathrm{B} = (B_1, \dots, B_n)$ is an equilibrium of M iff for every $1 \leq i \leq n$ there exists some $(kb'_i, \mathbf{acc}_{LS_i}) \in mng_i(app_i(br_i, \mathrm{B}), kb_i)$ such that $B_i \in \mathbf{acc}_{LS_i}(kb'_i)$.

Example

Planner - c_1

Logic: L_{asp}
Knowledge base:

$a \leftarrow not\ b.$
$b \leftarrow not\ a.$

Bridge rules:

$add(tok) \leftarrow not\ c_2{:}tok$
$add(a) \leftarrow c_3{:}\bot, not\ c_1{:}tok$

Implications - c_2

Logic: L_{asp}
Knowledge base:

$c \leftarrow \top.$

Bridge rules:

$add(tok) \leftarrow not\ c_1{:}tok$
$add(b) \leftarrow c_1{:}b$
$del(c \leftarrow \top) \leftarrow c_1{:}b$
$add(c) \leftarrow c_1{:}tok$

Consistency - c_3

Logic: L_p
Knowledge base:

$a \wedge \neg b$

Bridge rules:

$add(a) \leftarrow c_1{:}a$
$add(b) \leftarrow c_1{:}b$

Example

Planner - c_1

Logic: L_{asp}
Knowledge base:
$a \leftarrow not\ b.$
$b \leftarrow not\ a.$

Bridge rules:
$add(tok) \leftarrow not\ c_2{:}tok$
$add(a) \leftarrow c_3{:}\bot, not\ c_1{:}tok$

Implications - c_2

Logic: L_{asp}
Knowledge base:
$c \leftarrow \top.$

Bridge rules:
$add(tok) \leftarrow not\ c_1{:}tok$
$add(b) \leftarrow c_1{:}b$
$del(c \leftarrow \top) \leftarrow c_1{:}b$
$add(c) \leftarrow c_1{:}tok$

Consistency - c_3

Logic: L_p
Knowledge base:
$a \wedge \neg b$

Bridge rules:
$add(a) \leftarrow c_1{:}a$
$add(b) \leftarrow c_1{:}b$

Results

{a}	{tok, c}	$\{\top\}$
{a, tok}	{c}	$\{\top\}$
{b, tok}	{b}	$\{\bot\}$
{b, tok}	{b,c}	$\{\bot\}$

Outline

Inconsistency

An mMCS is inconsistent if there exists no equilibrium

Reasons for inconsistencies may be:

- Local inconsistency - acceptance function of the logic returns an **empty set**
- Operator inconsistency - operators are contradicting each other (e.g.**revise** knowledgebase s.t. $revise(a)$ and $revise(\neg a)$ is requested)
- Global inconsitency - bridge rules are causing inconsistencies

Inconsistencies can be countered by

- offering properties, which ensure that inconsistencies cannot occur,
- explain what is causing an inconsistency, and
- provide a diagnosis on how to achieve a consistent version of the mMCS

Local Consistency

Local consistency by utilising adequate **management functions**

Definition (local consistency preserving)

A management function mng is local consistency (lc-) preserving if, for each set of operational statements O and each knowledgebase kb, in every pair $(kb', \mathbf{acc}) \in \mathbf{mng}(O, kb)$ the knowledgebase kb' is consistent (i.e. $\mathbf{acc}(kb') \neq \emptyset$).

Definition (locally consistent mMCS)

An mMCS is locally consistent if in each equilibrium B $= (b_1, \ldots, b_n)$, all b_i are consistent belief sets.

Proposition

Let M be an mMCS s.t. all management functions are lc-preserving. Then M is locally consistent.

Local Consistency

Local consistency by utilising adequate **management functions**

Definition (local consistency preserving)

A management function mng is local consistency (lc-) preserving if, for each set of operational statements O and each knowledgebase kb, in every pair $(kb', \mathbf{acc}) \in \mathbf{mng}(O, kb)$ the knowledgebase kb' is consistent (i.e. $\mathbf{acc}(kb') \neq \emptyset$).

Definition (locally consistent mMCS)

An mMCS is locally consistent if in each equilibrium $\mathrm{B} = (b_1, \ldots, b_n)$, all b_i are consistent belief sets.

Proposition

Let M be an mMCS s.t. all management functions are lc-preserving. Then M is locally consistent.

Note

This ensures Operator consistency too

Global Consistency

Diagnoses and Explanations

- A **Diagnosis** of an mMCS M is a pair (D_1, D_2) of sets of bridge rules of M, s.t.
  - removing all bridge rules D_1 from M and
  - adding the heads of all bridge rules of D_2 as fact-rules of M

  ensures that the M has an equilibrium.
  We will denote the set of all Diagnoses for M with $D^{\pm}(M)$.

- An **Explanation** of an mMCS M is a pair (E_1, E_2) of sets of bridge rules, s.t. each set of bridge rules which
  - contain E_1 and
  - has no rules to apply the heads of E_2

  will lead M to be inconsistent.

- minimal Diagnoses and Explanations are a dual problem Schüller (2012)

Global Consistency

When does a diagnosis exist?

Conservative approach

If every context of an inconsistent mMCS is lc-preserving, then there exists a diagnosis.

Can we use semantics with inconsistent conclusions, but avoid this situation?

Global Consistency

When does a diagnosis exist?

Conservative approach

If every context of an inconsistent mMCS is lc-preserving, then there exists a diagnosis.

Can we use semantics with inconsistent conclusions, but avoid this situation?

Definition

A context C_i with knowledge base kb_i in an mMCS M is totally coherent, if for every belief state B of M some $(kb_i', \mathbf{acc}_{LS_i}) \in mng_i(app_i(\mathrm{B}), kb_i)$ exists, such that $\mathbf{acc}_{LS_i}(kb_i') \neq \emptyset$. It is **totally incoherent** if no belief state B fulfils that condition.

Proposition

Let M be an inconsistent mMCS. Then there exists a diagnosis if no context of M is totally incoherent.

The absence of total incoherence is sufficient, but not necessary.
Is there a Notion which characterises the existence of a diagnosis?

Global Consistency

Diagnosis existence

Definition

A context C_i with knowledge base kb_i in an mMCS M is omni-coherent, if for every set H of operational statements, occurring in the heads of the bridge rules br_i some $(kb'_i, \mathbf{acc}_{LS_i}) \in mng_i(app_i(H), kb_i)$ exists, such that $\mathbf{acc}_{LS_i}(kb'_i) \neq \emptyset$. It is **omni-incoherent** if no such H fulfils that condition.

Proposition

Let M be an inconsistent mMCS. Then there exists a diagnosis if and only if no context of M is omni-incoherent.

Global Consistency

Ensure inconsistent mMCS cannot occur

- Any **acyclic mMCS** with omni-coherent contexts has an equilibrium.

- Any **acyclic mMCS** with totally coherent contexts has an equilibrium.

- Any **acyclic mMCS** with lc-preserving contexts has an equilibrium.

Outline

Translating MCS to mMCS

An MCS $M = (C_1, \ldots, C_n)$ can be translated into a mMCS $M' = (C'_1 \ldots, C'_n)$ by translating every context:

- Logic $L = \langle KB, BS, \mathbf{acc} \rangle$ into logic suite $LS = \langle KB, BS, \{\mathbf{acc}\} \rangle$
- Introduction of $OP_i = \{add_i\}$
- Managementfunction $\mathbf{mng} : add_i(O, kb_i) = \{(kb_i \cup \{a \mid add(s) \in O\}, \mathbf{acc}_i)\}$

Translating mMCS to MCS

A mMCS $M = (C_1, \ldots, C_n)$ can be translated into an MCS $M' = (C'_1 \ldots, C'_n)$ by translating every context:

- Knowledgebase translation
  $KB' = \{kb \cup \{op_{newsym}(o) \mid o \in O\} \mid kb \in KB, O \subseteq F_{LS}^{OP}\}$
- Belief sets stay the same
- Managementfunction gets implemented in acceptance-function
  $\mathbf{acc}(kb) = \{B \mid B \in \mathbf{acc}'(kb'),$
  $(kb', \mathbf{acc}') \in mng(\{o \mid op_{newsym}(o) \in kb\}, kb \setminus op_{newsym}(F_{LS}^{OP}))\}$
- Bridgerule heads are substituted by op_{newsym}

Translating mMCS to MCS

A mMCS $M = (C_1, \ldots, C_n)$ can be translated into an MCS $M' = (C'_1 \ldots, C'_n)$ by translating every context:

- Knowledgebase translation
  $KB' = \{kb \cup \{op_{newsym}(o) \mid o \in O\} \mid kb \in KB, O \subseteq F_{LS}^{OP}\}$
- Belief sets stay the same
- Managementfunction gets implemented in acceptance-function
  $\mathbf{acc}(kb) = \{B \mid B \in \mathbf{acc}'(kb'),$
  $(kb', \mathbf{acc}') \in mng(\{o \mid op_{newsym}(o) \in kb\}, kb \setminus op_{newsym}(F_{LS}^{OP}))\}$
- Bridgerule heads are substituted by op_{newsym}

Therefore MCS and mMCS are equally expressive

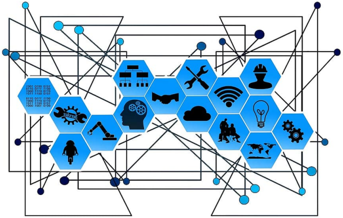
Computation Complexity

Existence of an equilibrium

$\mathcal{CC}(M)$	**P**	$\Sigma_i^{\mathbf{P}}$	$\Delta_{i+1}^{\mathbf{P}}$	**PSPACE**	**EXPTime**
$\mathcal{CONS}(M)$	**NP**	$\Sigma_i^{\mathbf{P}}$	$\Sigma_{i+1}^{\mathbf{P}}$	**PSPACE**	**EXPTime**

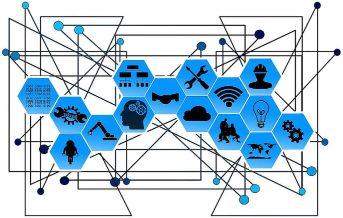
Outline

From Static Instances to Dynamic Streams



- Modern networks are growing fast as various new devices go online from tiny sensors to fridges or self-driving cars
- Dynamic streams of data, potentially infinite, with different frequency of changes:
  - low: smart buildings, railroad monitoring, business processes
  - high: stock trading, self-driving cars, network monitoring
- In IoT or Industry 4.0 scenarios data is pushed rather than pulled

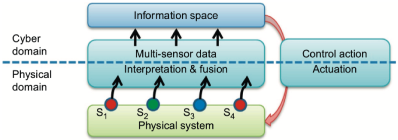
From Static Instances to Dynamic Streams



- Modern networks are growing fast as various new devices go online from tiny sensors to fridges or self-driving cars
- Dynamic streams of data, potentially infinite, with different frequency of changes:
  - low: smart buildings, railroad monitoring, business processes
  - high: stock trading, self-driving cars, network monitoring
- In IoT or Industry 4.0 scenarios data is pushed rather than pulled
- Stream reasoning is applicable in multiple *use cases* (e.g., Della Valle *et al.* [2008,2009]):
  - Montoring & Control: Applications for drones/robots (Amazon) or smart infrastructures (Dell EMC, Siemens)
  - Prediction: Maintenance of production lines/analysis labs (Infineon)
  - Diagnosis/Configuration: Detection, identification of causes, and reaction on various disruptions

Use Case: Dynamic (Re)configuration of Cyber-Physical Systems

Observation: Stream Reasoning is suited to model dynamic (re)configuration scenarios

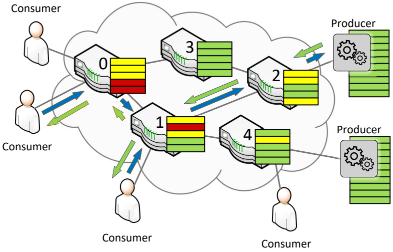


CPS: combing physical space with information space via computing, communication and control.

- Structuring into interlinked reasoning components, evolving over time
- Logical *separation of concerns (SoC) / tasks*
  - **Producers**: components/systems that provide data to the Stream Reasoning system. E.g. sensors can be viewed as such
  - **Monitors**: stream reasoners that observe and aggregate data streams from producers, and report (feed information) to configurators
  - **Configurators**: reasoners calculating the setup thru re-configuring the CPS; may involve complex decision component, richer high level stream reasoning
  - **Actuators**: components/systems that change the setup in the CPS environment according to the output of the configurators
- SoC may be weakened (integrate actuators into producers)

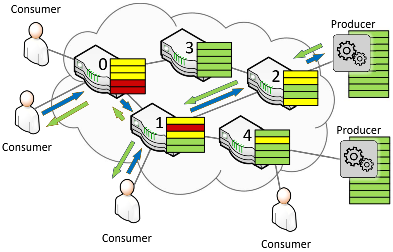
Scenario: Network Administration

- Scalability problems:
  - Popular content producers get overloaded
  - Network connections become congested
  - Content distribution over the Internet needs workarounds
- New Internet architectures are proposed to solve these problems
  - Scalable content distribution as a main feature



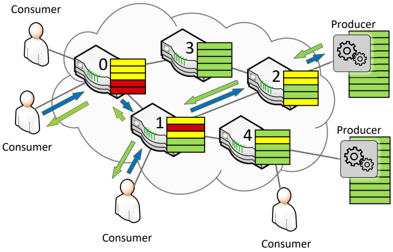
Scenario: Network Administration

- Scalability problems:
  - Popular content producers get overloaded
  - Network connections become congested
  - Content distribution over the Internet needs workarounds
- New Internet architectures are proposed to solve these problems
  - Scalable content distribution as a main feature
- ***Content-Centric Networking***
  - Address content in the network by "name" – physical location irrelevant
  - ***Content-Centric Routers (CCRs)*** can route interest packages, cache and adapt media chunks in highly dynamic conditions
  - **Cache sizes are limited** – need efficient caching strategies that can react on changes of users' interests over time



Scenario: Network Administration

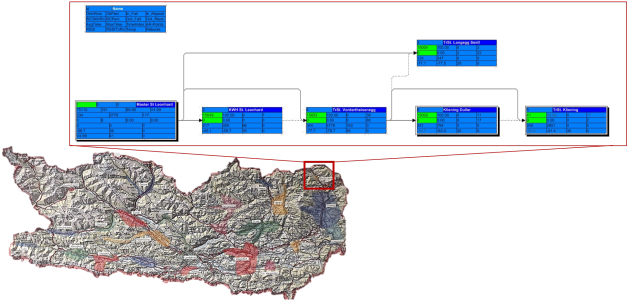
- Scalability problems:
  - Popular content producers get overloaded
  - Network connections become congested
  - Content distribution over the Internet needs workarounds
- New Internet architectures are proposed to solve these problems
  - Scalable content distribution as a main feature
- ***Content-Centric Networking***
  - Address content in the network by “name” – physical location irrelevant
  - ***Content-Centric Routers (CCRs)*** can route interest packages, cache and adapt media chunks in highly dynamic conditions
  - **Cache sizes are limited** – need efficient caching strategies that can react on changes of users’ interests over time



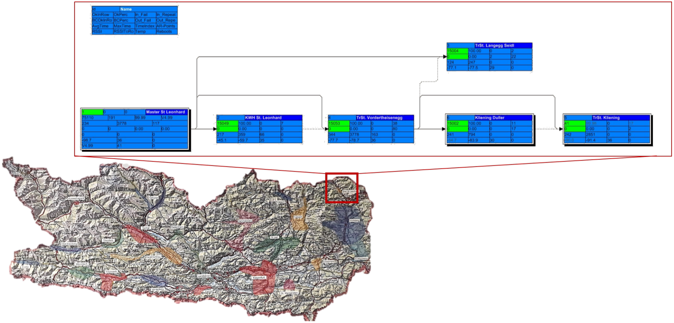
- **Producers**: Content requests/chunks sent over the network and statistics aggregated by routers
- **Monitors**: stream reasoners detect changes in interests and activity of users by analyzing network statistics
- **Configurators**: selection of a caching strategy allowing for the best possible load reduction on the network
- **Actuators**: network interfaces and controllers of routers

Scenario: Stream Reasoning for Energy Grids

- Monitoring and control of energy grids is essential for reliable energy supply
- Grid operators use specific networks to transmit real-time data from distant energy distribution nodes
- **Example**: Kelag AG networks in Austrian Alps (shown in color) use radio modems for data transmission
  - Due to weather conditions, human activity, etc., radio links between modems might become unstable
  - Stream reasoning system should reconfigure modems and antennas to enable alternative possibly stable routes



Scenario: Stream Reasoning for Energy Grids, cont'd



- **Producers**: Modems and the Master Node that communicate and collect information about radio signal quality and transmitted packages
- **Monitors**: stream reasoners detect instability of radio signals and deviations in connection speed
- **Configurators**: select network topology allowing for the best possible data transfer rates between nodes of the network
- **Actuators**: radio modems and antenna orientation controllers

Scenario: Cooperative Intelligent Transporation Systems (C-ITS)

- **Cooperative-ITS (Vision):**
  - Health & Safety by monitoring
  - Efficient urban mobility by optimizations
  - Enabling technology for autonomous cars!

V2X Technology Overview

Scenario: Cooperative Intelligent Transporation Systems (C-ITS)

- **Cooperative-ITS (Vision):**
  - Health & Safety by monitoring
  - Efficient urban mobility by optimizations
  - Enabling technology for autonomous cars!
- **Vehicle-to-X communication (V2X)**
  - Traffic participants exchange information as V2X messages (ETSI, 2013)
  - Real time, simultaneously, and location based

V2X Technology Overview

Scenario: Cooperative Intelligent Transportation Systems (C-ITS)

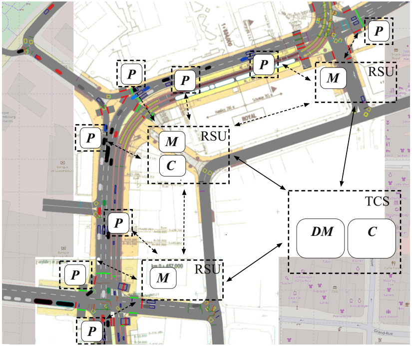
- **Cooperative-ITS (Vision):**
  - Health & Safety by monitoring
  - Efficient urban mobility by optimizations
  - Enabling technology for autonomous cars!
- **Vehicle-to-X communication (V2X)**
  - Traffic participants exchange information as V2X messages (ETSI, 2013)
  - Real time, simultaneously, and location based
- **Wide variety of reasoning tasks available:**
  - Quick monitoring of problems
    → Event detection
  - Finding more complex, long-term problems
    → Model-based Diagnosis
  - Adapt traffic lights to current traffic
    → Configuration

V2X Technology Overview

Scenario: C-ITS, cont'd

C-ITS infrastructure as a CPS:

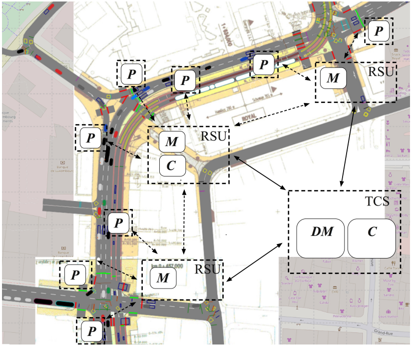
- Dotted boxes are (cyber-)physical units
- Each intersection has one roadside unit (RSU) that communicats via V2X (dotted)
- Central traffic control center (TCS) is connected to all RSUs



Model of intersection in Luxemburg

Scenario: C-ITS, cont'd

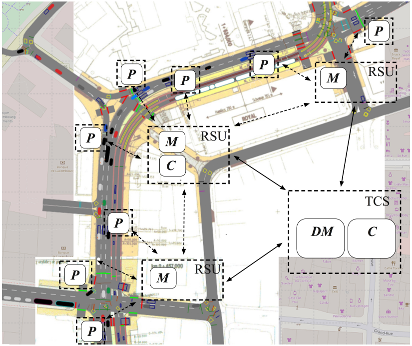
- **C-ITS infrastructure as a CPS:**
  - Dotted boxes are (cyber-)physical units
  - Each intersection has one roadside unit (RSU) that communicats via V2X (dotted)
  - Central traffic control center (TCS) is connected to all RSUs
- **Producers**:
  - Traffic participants like vehicles are mobile sensors
  - Vehicles send status and traffic lights signal phases via V2X



Model of intersection in Luxemburg

Scenario: C-ITS, cont'd

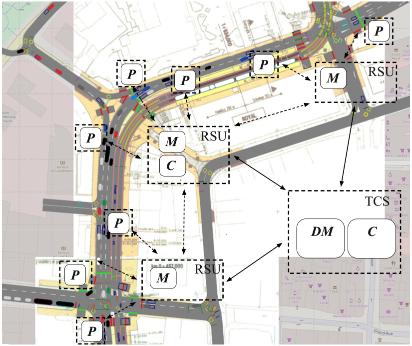
- **C-ITS infrastructure as a CPS:**
  - Dotted boxes are (cyber-)physical units
  - Each intersection has one roadside unit (RSU) that communicats via V2X (dotted)
  - Central traffic control center (TCS) is connected to all RSUs
- **Producers**:
  - Traffic participants like vehicles are mobile sensors
  - Vehicles send status and traffic lights signal phases via V2X
- **Monitors**:
  - Stream aggregation/event detection on RSUs
  - High volume, velocity streams from sensors
  - Local view of traffic



Model of intersection in Luxemburg

Scenario: C-ITS, cont'd

- **C-ITS infrastructure as a CPS:**
  - Dotted boxes are (cyber-)physical units
  - Each intersection has one roadside unit (RSU) that communicats via V2X (dotted)
  - Central traffic control center (TCS) is connected to all RSUs
- **Producers**:
  - Traffic participants like vehicles are mobile sensors
  - Vehicles send status and traffic lights signal phases via V2X
- **Monitors**:
  - Stream aggregation/event detection on RSUs
  - High volume, velocity streams from sensors
  - Local view of traffic
- **Configurators:**
  - Main configurator is in the TCS
  - Optimize traffic flow via dynamic configurating of the traffic lights
  - Global view of traffic



Model of intersection in Luxemburg

Scenario: Quality Assurance Lab

- **Quality Assurance Lab as a CPS:**
  - Lab engineers have to accomplish series of QA tests for each job
  - Resources of a lab are limited and must be utilized in the best possible way to avoid delays
  - Classic scheduling approaches are not applicable since durations and sequences of actions are not deterministic

Semiconductor QA Lab (c) Wikipedia

Scenario: Quality Assurance Lab

- **Quality Assurance Lab as a CPS:**
  - Lab engineers have to accomplish series of QA tests for each job
  - Resources of a lab are limited and must be utilized in the best possible way to avoid delays
  - Classic scheduling approaches are not applicable since durations and sequences of actions are not deterministic
- **Producers**:
  - Engineers performing QA tasks
  - Tools that can be occupied, malfunctioning, maintained, etc.
  - Production environment delivering products for QA

Semiconductor QA Lab (c) Wikipedia

Scenario: Quality Assurance Lab

- **Quality Assurance Lab as a CPS:**
  - Lab engineers have to accomplish series of QA tests for each job
  - Resources of a lab are limited and must be utilized in the best possible way to avoid delays
  - Classic scheduling approaches are not applicable since durations and sequences of actions are not deterministic
- **Producers**:
  - Engineers performing QA tasks
  - Tools that can be occupied, malfunctioning, maintained, etc.
  - Production environment delivering products for QA
- **Monitors**:
  - Data stream describing events in the Lab (personnel, tools, incoming queues, etc.)
  - Low volume/velocity streams from sensors
  - Automatic detection of disruptions (rules or ML models)

Semiconductor QA Lab (c) Wikipedia

Scenario: Quality Assurance Lab

- **Quality Assurance Lab as a CPS:**
  - Lab engineers have to accomplish series of QA tests for each job
  - Resources of a lab are limited and must be utilized in the best possible way to avoid delays
  - Classic scheduling approaches are not applicable since durations and sequences of actions are not deterministic
- **Producers**:
  - Engineers performing QA tasks
  - Tools that can be occupied, malfunctioning, maintained, etc.
  - Production environment delivering products for QA
- **Monitors**:
  - Data stream describing events in the Lab (personnel, tools, incoming queues, etc.)
  - Low volume/velocity streams from sensors
  - Automatic detection of disruptions (rules or ML models)

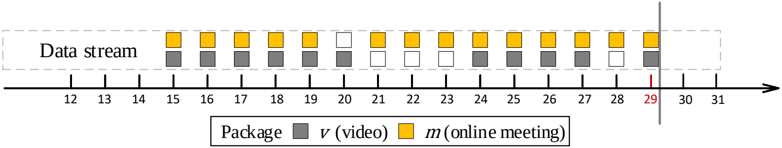
Semiconductor QA Lab (c) Wikipedia

- **Configurators:**
  - Simulation of Lab future events using scheduling/planing techniques
  - Optimize the thoughput of the Lab by recommending tasks to engineers
  - Predict/detect possible delays for team leads

Outline

Example: Network Administration

Network administrators would like the routers to select the best possible caching strategy depending on the statistics of user requests

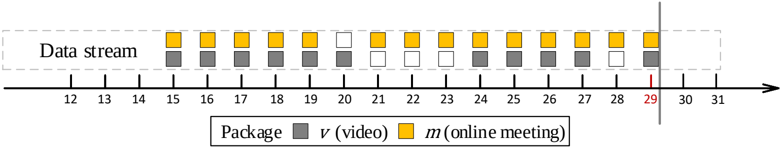


$$\{\ldots, 19 : \{package(v, r_1), package(m, r_1)\}, 20 : \{package(v, r_1)\}, 21 : \{package(m, r_1)\}, \ldots\}$$

- Factor: *Current daytime* – has high correlation with number of active users and their behavioral patterns

Example: Network Administration

Network administrators would like the routers to select the best possible caching strategy depending on the statistics of user requests

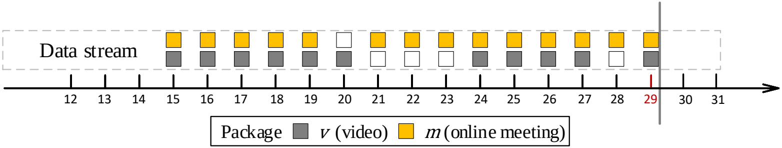


$$\{\ldots, 19 : \{package(v, r_1), package(m, r_1)\}, 20 : \{package(v, r_1)\}, 21 : \{package(m, r_1)\}, \ldots\}$$

- Factor: *Current daytime* – has high correlation with number of active users and their behavioral patterns
- Some sample scenarios are:
  - Morning: the number of active users is low and they are interested in different media
  - Evening: in the evening a lot of users are watching a small amount of popular series

Example: Network Administration

Network administrators would like the routers to select the best possible caching strategy depending on the statistics of user requests



$$\{\ldots, 19 : \{package(v, r_1), package(m, r_1)\}, 20 : \{package(v, r_1)\}, 21 : \{package(m, r_1)\}, \ldots\}$$

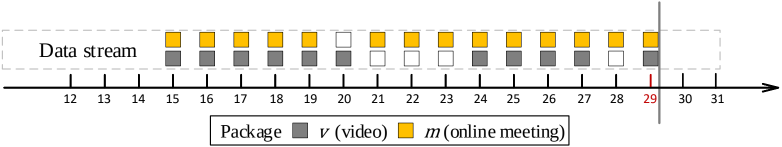
- Factor: *Current daytime* – has high correlation with number of active users and their behavioral patterns
- Some sample scenarios are:
  - Morning: the number of active users is low and they are interested in different media
  - Evening: in the evening a lot of users are watching a small amount of popular series
- Possible caching strategies for the scenarios above:
  - Random: replaces a *random* chunk in the cache with the current chunk received by the networking unit
  - LFU: the received chunk replaces the *Least Frequently Used* chunk in the cache

Stream Reasoning: Basic Notions

- Knowledge base (ontology, logic program, etc.) KB

Stream Reasoning: Basic Notions

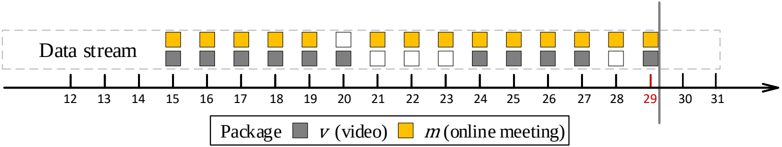
- Knowledge base (ontology, logic program, etc.) KB
- Streams:
  - A stream is a pair $S = (T, \upsilon)$ of a timeline T and a mapping $\upsilon : T \to 2^{\mathcal{A}}$, where
    - $\mathcal{A}$ is a set of facts (atoms)
    - $T = [l, u] = \{l, l{+}1, \ldots, u\} \subseteq 2^{\mathbb{N}_0}$ is an interval of integers



Example: $T = [19, 29]$
$\upsilon = \{19 : \{package(v, r_1), package(m, r_1)\}, 20 : \{package(v, r_1), \ldots\}\}$

Stream Reasoning: Basic Notions

- Knowledge base (ontology, logic program, etc.) KB
- Streams:
  - A **stream** is a pair $S = (T, \upsilon)$ of a timeline T and a mapping $\upsilon : T \to 2^{\mathcal{A}}$, where
    - $\mathcal{A}$ is a set of facts (atoms)
    - $T = [l, u] = \{l, l+1, \ldots, u\} \subseteq 2^{\mathbb{N}_0}$ is an interval of integers



Example: $T = [19, 29]$
$\upsilon = \{19 : \{package(v, r_1), package(m, r_1)\}, 20 : \{package(v, r_1), \ldots\}\}$

- Query Q: formula, to be evaluated against KB over data stream S

Example: $package(v, r_1)$; $package(X, r_1)$

Stream Reasoning: Basic Notions (cont'd)

Question: *How to evaluate Q against KB over S at a query time point t?*

- Q may refer to the query time, involve prediction / postdiction etc

Example: $\mathrm{X}\,package(v, r_1)$ "A video package v must be observed on the router r_1 in the ne**X**t step."

- **Temporal Reasoning**
  - temporal logic over (data) streams
  - use linear time logic (LTL) (Pnueli, 1977), branching time logic (CTL, CTL*) (Clarke and Emerson, 1981)
  - Complex Event Processing: patterns in data (event) streams (e.g., ETALIS (Anicic *et al.*, 2012), RTEC (Artikis *et al.*, 2014))

Stream Reasoning vs. Stream Processing

- **Streaming Data**
  - high data volume, volatility of data (speed of data change)
  - deliberate *information loss* (drop data): use *data snapshots*
  - evaluate *pull-based* (at given time-points) or push-based (when data appears)
  - *incremental evaluation* desired
  - *time management* is important
    - system vs. application time
    - point-wise vs. interval representation

Stream Reasoning vs. Stream Processing

- **Streaming Data**
  - high data volume, volatility of data (speed of data change)
  - deliberate *information loss* (drop data): use *data snapshots*
  - evaluate *pull-based* (at given time-points) or push-based (when data appears)
  - *incremental evaluation* desired
  - *time management* is important
    - system vs. application time
    - point-wise vs. interval representation

- **Stream Processing vs. Stream Reasoning:**
  - “lower level” processing (selections, joins) vs. “higher level” (reasoning steps)
  - “declarative” Stream-X
  - controversial views in the Stream Reasoning community

Data Snapshots: Window Functions

- Important aspect of stream reasoning: use only *window* view of data, i.e., limited observability at each point in time

Data Snapshots: Window Functions

- Important aspect of stream reasoning: use only *window* view of data, i.e., limited observability at each point in time
- Different types of windows in practice:
  - *time-based windows* (within time bounds)
  - *tuple-based windows* (number of tuples, count)
  - *partition-based windows* (split input data, process separately)
  - in addition, *sliding* or *tumbling* (consider atom repeatedly / once)

Data Snapshots: Window Functions

- Important aspect of stream reasoning: use only *window* view of data, i.e., limited observability at each point in time
- Different types of windows in practice:
  - *time-based windows* (within time bounds)
  - *tuple-based windows* (number of tuples, count)
  - *partition-based windows* (split input data, process separately)
  - in addition, *sliding* or *tumbling* (consider atom repeatedly / once)
- Model data snapshots (windows) as *substreams* of a stream

Data Snapshots: Window Functions

- Important aspect of stream reasoning: use only *window* view of data, i.e., limited observability at each point in time
- Different types of windows in practice:
  - *time-based windows* (within time bounds)
  - *tuple-based windows* (number of tuples, count)
  - *partition-based windows* (split input data, process separately)
  - in addition, *sliding* or *tumbling* (consider atom repeatedly / once)
- Model data snapshots (windows) as *substreams* of a stream
- Formally, windows are functions

$$w : (S, t) \mapsto S'$$

assigning each stream $S = (T, \upsilon)$ and $t \in T$ a substream $S' \subseteq S$, which means $S' = (T', \upsilon')$ such that $T' \subseteq T$ and $\upsilon'(t) \subseteq \upsilon(t)$, for all $t \in T'$

Example: Window Functions

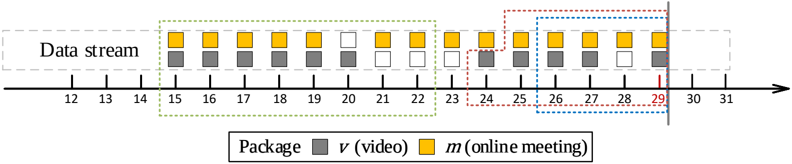
- Window operators $\boxplus$ (substream generation)

$$\boxplus^w \Longleftrightarrow w(S, t)$$

Example: Window Functions

- Window operators $\boxplus$ (substream generation)

$$\boxplus^w \iff w(S,t)$$

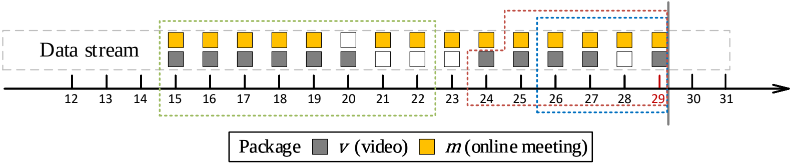


- Examples:
  - $\boxplus^4 := \boxplus^{w_\tau^{(4,0,1)}}$ last 4 units from the current time point (29) with step 1 – sliding time-based window, shown in blue

Example: Window Functions

- Window operators $\boxplus$ (substream generation)

$$\boxplus^w \iff w(S,t)$$

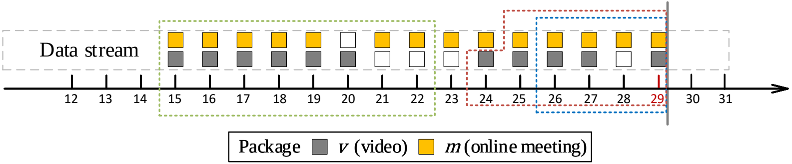


- Examples:
  - $\boxplus^4 := \boxplus^{w_\tau^{(4,0,1)}}$ last 4 units from the current time point (29) with step 1 – sliding time-based window, shown in blue
  - $\boxplus^{8(8)} := \boxplus^{w_\tau^{(8,0,8)}}$ last 8 units with step 8 – hoping time-based window, shown in green

Example: Window Functions

- Window operators $\boxplus$ (substream generation)

$$\boxplus^w \Longleftrightarrow w(S,t)$$



- Examples:
  - $\boxplus^4 := \boxplus^{w_\tau^{(4,0,1)}}$ last 4 units from the current time point (29) with step 1 – sliding time-based window, shown in blue
  - $\boxplus^{8(8)} := \boxplus^{w_\tau^{(8,0,8)}}$ last 8 units with step 8 – hoping time-based window, shown in green
  - $\boxplus^{\#10} = \boxplus^{w_\#^{10,0}}$ last 10 tuples – sliding tuple-based window, shown in red – non-deterministic!

Temporal Operators and Formulas by Example

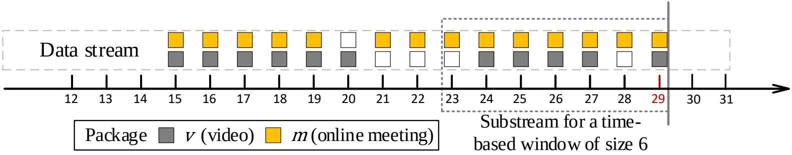
Languages for stream reasoning often extend logic languages with stream access / processing features

- Atoms from $\mathcal{A}$ (atomic formulas a), e.g., $\mathcal{A} = \{package(v, r_1), package(m, r_1)\}$
- Boolean connectives $\land$, $\lor$, $\rightarrow$, $\neg$

Temporal Operators and Formulas by Example

Languages for stream reasoning often extend logic languages with stream access / processing features

- Atoms from $\mathcal{A}$ (atomic formulas a), e.g., $\mathcal{A} = \{package(v, r_1), package(m, r_1)\}$
- Boolean connectives $\wedge$, $\vee$, $\rightarrow$, $\neg$
- Window operators $\boxplus$
- Temporal operators $\Diamond$, $\Box$, $@_t$

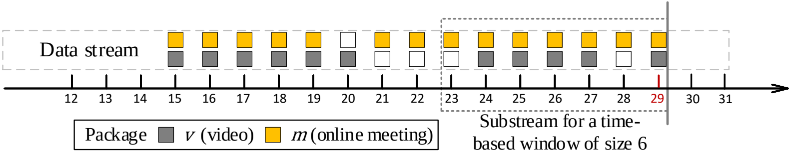


- Examples:

Temporal Operators and Formulas by Example

Languages for stream reasoning often extend logic languages with stream access / processing features

- Atoms from $\mathcal{A}$ (atomic formulas a), e.g., $\mathcal{A} = \{package(v, r_1), package(m, r_1)\}$
- Boolean connectives $\wedge$, $\vee$, $\rightarrow$, $\neg$
- Window operators $\boxplus$
- Temporal operators $\Diamond$, $\Box$, $@_t$

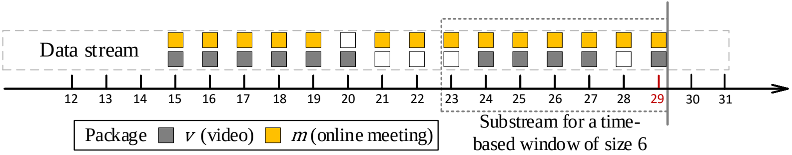


- Examples:
  - $\boxplus^6 \Box\, package(m, r_1)$, $\boxplus^6 \Diamond\, package(v, r_1)$, and $@_{t-3}\, package(v, r_1)$ hold

Temporal Operators and Formulas by Example

Languages for stream reasoning often extend logic languages with stream access / processing features

- Atoms from $\mathcal{A}$ (atomic formulas a), e.g., $\mathcal{A} = \{package(v, r_1), package(m, r_1)\}$
- Boolean connectives $\wedge$, $\vee$, $\rightarrow$, $\neg$
- Window operators $\boxplus$
- Temporal operators $\Diamond$, $\Box$, $@_t$

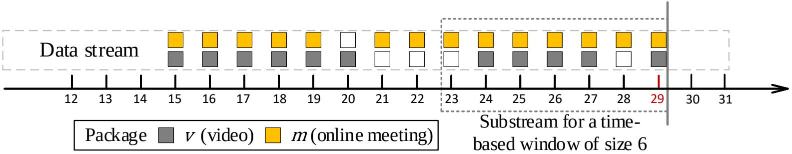


- Examples:
  - $\boxplus^6 \Box\, package(m, r_1)$, $\boxplus^6 \Diamond\, package(v, r_1)$, and $@_{t-3}\, package(v, r_1)$ hold
  - $\boxplus^6 @_{28}\, package(v, r_1)$ and $\Box\, package(m, r_1)$ do not hold

Temporal Operators and Formulas by Example

Languages for stream reasoning often extend logic languages with stream access / processing features

- Atoms from $\mathcal{A}$ (atomic formulas a), e.g., $\mathcal{A} = \{package(v, r_1), package(m, r_1)\}$
- Boolean connectives $\wedge$, $\vee$, $\rightarrow$, $\neg$
- Window operators $\boxplus$
- Temporal operators $\Diamond$, $\Box$, $@_t$

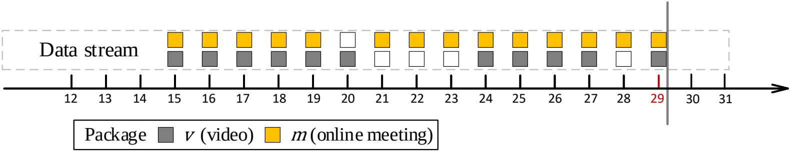


- Examples:
  - $\boxplus^6 \Box\, package(m, r_1)$, $\boxplus^6 \Diamond\, package(v, r_1)$, and $@_{t-3}\, package(v, r_1)$ hold
  - $\boxplus^6 @_{28}\, package(v, r_1)$ and $\Box\, package(m, r_1)$ do not hold
- Note: nesting of windows is possible!

$$\boxplus^{60} \Box \boxplus^5 \Diamond\, package(v, r_1) \qquad \boxplus^{\#20} \boxplus^5 \Diamond\, package(m, r_1)$$

Entailment by Example

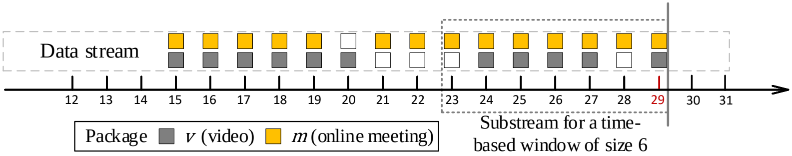
Entailment $M, S^{\star}, t \Vdash \alpha$ where $M = \langle S^{\star}, W, B \rangle$ consists of the initial stream $S^{\star}$, window functions W and static background B



$$M, S^{\star}, 29 \Vdash \boxplus^{6} \Diamond package(v, r_1)\ ?$$

Entailment by Example

Entailment $M, S^\star, t \Vdash \alpha$ where $M = \langle S^\star, W, B \rangle$ consists of the initial stream $S^\star$, window functions W and static background B



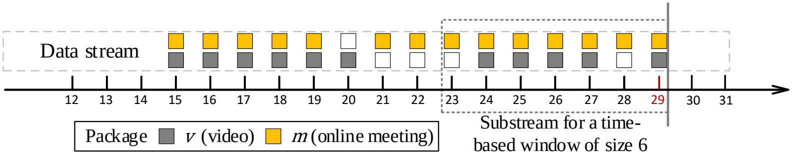
$M, S^\star, 29 \Vdash \boxplus^6 \Diamond package(v, r_1)$? evaluate window $\Rightarrow S$

$\Uparrow$

$M, S, 29 \Vdash \Diamond package(v, r_1)$?

Entailment by Example

Entailment $M, S^{\star}, t \Vdash \alpha$ where $M = \langle S^{\star}, W, B \rangle$ consists of the initial stream $S^{\star}$, window functions W and static background B



$M, S^{\star}, 29 \Vdash \boxplus^{6} \Diamond package(v, r_1)$? evaluate window $\Rightarrow S$

$\Uparrow$

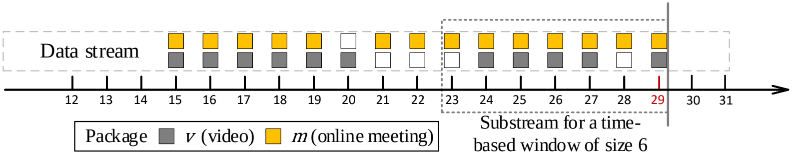
$M, S, 29 \Vdash \Diamond package(v, r_1)$?

$\Uparrow$

$M, S, 24 \Vdash package(v, r_1)$

Entailment by Example

Entailment $M, S^\star, t \Vdash \alpha$ where $M = \langle S^\star, W, B \rangle$ consists of the initial stream $S^\star$, window functions W and static background B



$M, S^\star, 29 \Vdash \boxplus^6 \Diamond package(v, r_1)$? evaluate window $\Rightarrow S$

$\Uparrow$

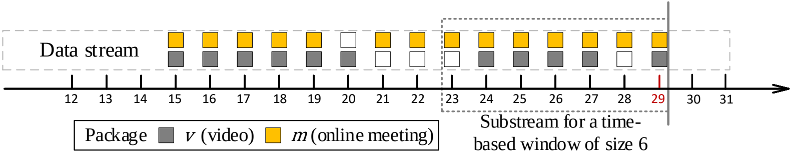
$M, S, 29 \Vdash \Diamond package(v, r_1)$?

$\Uparrow$

$M, S, 24 \Vdash package(v, r_1)$ ✓

Entailment by Example

Entailment $M, S^\star, t \Vdash \alpha$ where $M = \langle S^\star, W, B \rangle$ consists of the initial stream $S^\star$, window functions W and static background B



$M, S^\star, 29 \Vdash \boxplus^6 \Diamond package(v, r_1)$? evaluate window $\Rightarrow S$

$\Uparrow$

$M, S, 29 \Vdash \Diamond package(v, r_1)$?

$\Uparrow$

$M, S, 24 \Vdash package(v, r_1)$ ✓

defines query evaluation if the whole stream $S^\star$ is data

Historical Developments

Different communities looked at different aspects

- **Data Management:**
  - stream processing approach
  - continuous queries
  - low-level, high rate input data (cross-joins, pattern matching, etc.)
  - windows for partial data snapshots

Historical Developments

Different communities looked at different aspects

- **Data Management:**
  - stream processing approach
  - continuous queries
  - low-level, high rate input data (cross-joins, pattern matching, etc.)
  - windows for partial data snapshots
- **Knowledge Representation and Reasoning:**
  - stream reasoning
  - higher-level, lower rate (scalability!)
  - changing knowledge bases (ontologies, rule bases)

Historical Developments

Different communities looked at different aspects

- **Data Management:**
  - stream processing approach
  - continuous queries
  - low-level, high rate input data (cross-joins, pattern matching, etc.)
  - windows for partial data snapshots
- **Knowledge Representation and Reasoning:**
  - stream reasoning
  - higher-level, lower rate (scalability!)
  - changing knowledge bases (ontologies, rule bases)
- **Semantic Web:**
  - lifting stream data to a semantic level
  - linked stream data (coupling tuples with timestamps)
  - several extensions of SPARQL (e.g. CSPARQL or CQELS)

Observations

- **Lack of (unified) formal foundations**
  stream processing:
  - often operational semantics; unpredictable
  - systems may give for same query different results

Observations

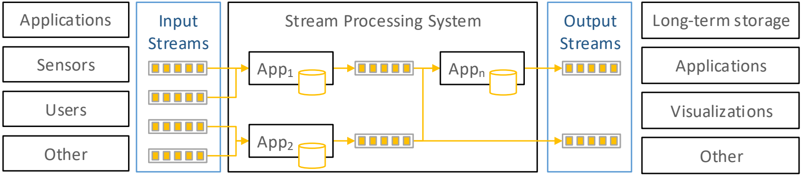
- **Lack of (unified) formal foundations**
  stream processing:
  - often operational semantics; unpredictable
  - systems may give for same query different results
- **Comparisons / benchmarks unsatisfactory**
  - semantics outcome not / weakly addressed (tuple counting)
  - benchmarks geared towards high-frequency / limits
  - no general methods, no reference semantics

Observations

- **Lack of (unified) formal foundations**
  stream processing:
  - often operational semantics; unpredictable
  - systems may give for same query different results
- **Comparisons / benchmarks unsatisfactory**
  - semantics outcome not / weakly addressed (tuple counting)
  - benchmarks geared towards high-frequency / limits
  - no general methods, no reference semantics
- **Advanced features missing**
  - nondeterminism
  - incomplete information
  - negation
  - model generation

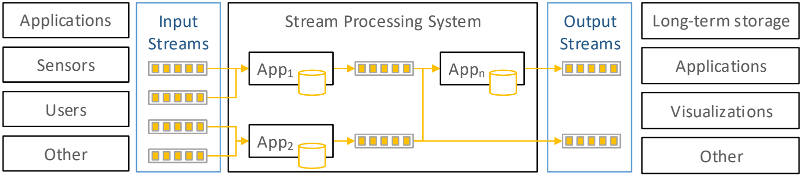
Outline

Stream Processing Systems



- **Stream processing systems:** designed for low-latency, distributed computations on high volumes of continuously incoming data, e.g.
  - Apache Kafka (Bejeck and Narkhede, 2018)
  - Apache Flink (Hueske and Kalavri, 2019)

Stream Processing Systems



- **Stream processing systems:** designed for low-latency, distributed computations on high volumes of continuously incoming data, e.g.
  - Apache Kafka (Bejeck and Narkhede, 2018)
  - Apache Flink (Hueske and Kalavri, 2019)
- Different systems can support
  - automatic deployment of processing infrastructures into clouds and clusters
  - load balancing and management data communication
  - storage/caching of computation results (stateless vs. stateful computations)
  - time management for computation and communication
  - fail recovery mechanisms
  - various programming paradigms, e.g., procedural or functional programming
  - limited query execution, e.g., SQL-like language in Apache Flink

Outline

Databases and Data Streams

- **Before 2000**
  - Novel applications of computers in the 80s, e.g., Supervisory Control and Data Acquisition (SCADA) systems, led to increased utilization of data streams
  - Existing databases using relational, hierarchical, or network models could not handle continuously incoming data efficiently
  - **Active databases with event-condition rules** (Dayal *et al.*, 1995; Widom and Ceri, 1996) ⇒ triggers in modern databases
  - Continuous queries over append-only databases (Terry *et al.*, 1992)

[3] `https://www.w3.org/community/rsp`

Databases and Data Streams

- **Before 2000**
  - Novel applications of computers in the 80s, e.g., Supervisory Control and Data Acquisition (SCADA) systems, led to increased utilization of data streams
  - Existing databases using relational, hierarchical, or network models could not handle continuously incoming data efficiently
  - **Active databases with event-condition rules** (Dayal *et al.*, 1995; Widom and Ceri, 1996) $\Rightarrow$ triggers in modern databases
  - Continuous queries over append-only databases (Terry *et al.*, 1992)
- **2000 –**
  - Sliding, tumbling (non-overlapping) windows as well as latch windows that can maintain states of multiple windows in the Aurora system (Abadi *et al.*, 2003)
  - Stanford Stream Data Management (STREAM) (Arasu *et al.*, 2003) which employed Continuous Query Language (CQL) (Arasu *et al.*, 2006)
    - Provides an explicit operational semantics
    - Three kinds of window operators: partitioned as well as time- and tuple-based windows
  - Streaming data on the Web resulted in development of continuous query languages, e.g., C-SPARQL (Barbieri *et al.*, 2010) or CQELS (Phuoc *et al.*, 2011), for RDF Stream Processing[3]

[3] https://www.w3.org/community/rsp

Databases and Data Streams (cont'd)

- **Early databases**:
  - Stanford Stream Data Management (STREAM) (Arasu *et al.*, 2003) wit CQL
  - TelegraphCQ (Madden *et al.*, 2002) with the SQL-based language (StreaQuel)
  - Aurora (Carney *et al.*, 2002) as a workflow-oriented system
  - COUGAR (Fung *et al.*, 2002) provides object-oriented extension

- **Current databases**:
  - Hancock (Cortes *et al.*, 2016) introduces transactional data streams
  - PipelineDB [4] is an extension of PostgreSQL
  - Esper [5] and Odysseus [6] provide event processing languages

- **Semantic Web-based systems**:
  - C-SPARQL (Barbieri *et al.*, 2010)
  - CQELS (Phuoc *et al.*, 2011)
  - SPARQLstream (Calbimonte *et al.*, 2016)

[4] https://www.pipelinedb.com/

[5] http://www.espertech.com/esper/

[6] http://odysseus.informatik.uni-oldenburg.de/

Outline

Complex Event Processing (CEP) Systems (Luckham, 2005)

- Consider streams of event notifications
- **CEP systems** usually employ multiple event consumers that derive composite (high-level) events out of sequences (patterns) of input (low-level) events
- Support declarative languages with complex expressions over temporal intervals and sequences of events
  - Rapide (Luckham, 1996): language to simulate concurrent and distributed systems
  - CEDR (Barga *et al.*, 2007): system considering both system and application time to provide different consistency guarantees for derived events
  - Cayuga (Brenna *et al.*, 2007): SQL-like query language without window operators, uses a specific Cayuga algebra
  - Sase (Wu *et al.*, 2006): system designed for large-scale event processing, provides a CQL-like language that allows for usage of sliding time-based windows
  - Tesla (Cugola and Margara, 2010): rule-based language used in the T-Rex system (Cugola and Margara, 2012)
    - formal semantics defined in a Metric Temporal Logic (MTL)
    - supports temporal operators, timers, aggregate functions, negation-as-failure

ETALIS (Anicic *et al.*, 2012)

- Complex event processing language, with rules $a \leftarrow pt$, where
  - a is an atom
  - pt is a *complex event pattern* on *intervals*
- Relations akin to Allen's [1983] interval logic (Sequence, During,...)

ETALIS (Anicic *et al.*, 2012)

- Complex event processing language, with rules $a \leftarrow pt$, where
  - a is an atom
  - pt is a *complex event pattern* on *intervals*
- Relations akin to Allen's [1983] interval logic (Sequence, During,...)
- Interpretation: $a \mapsto \mathcal{I}(a) \subseteq \{\langle t, t' \rangle \mid t \leq t' \in \mathbb{R}_0^+\}$
- Event stream: $a \mapsto \epsilon(a) \subseteq \mathbb{R}_0^+$

ETALIS (Anicic *et al.*, 2012)

- Complex event processing language, with rules $a \leftarrow pt$, where
  - a is an atom
  - pt is a *complex event pattern* on *intervals*
- Relations akin to Allen's [1983] interval logic (Sequence, During,...)
- Interpretation: $a \mapsto \mathcal{I}(a) \subseteq \{\langle t, t' \rangle \mid t \leq t' \in \mathbb{R}_0^+\}$
- Event stream: $a \mapsto \epsilon(a) \subseteq \mathbb{R}_0^+$
- $\mathcal{I}$ is a *model* of rule base $\mathcal{R}$ for event stream ϵ, if
  (i) $\langle t, t \rangle \in \mathcal{I}(a)$ for each atom a appearing at t in ϵ, and
  (ii) $\langle t, t' \rangle \in \mathcal{I}(a)$ for each $\langle t, t' \rangle$ matching pt in a rule $a \leftarrow pt$

ETALIS (Anicic *et al.*, 2012)

- Complex event processing language, with rules $a \leftarrow pt$, where
  - a is an atom
  - pt is a *complex event pattern* on *intervals*
- Relations akin to Allen's [1983] interval logic (Sequence, During,...)
- Interpretation: $a \mapsto \mathcal{I}(a) \subseteq \{\langle t, t' \rangle \mid t \leq t' \in \mathbb{R}_0^+\}$
- Event stream: $a \mapsto \epsilon(a) \subseteq \mathbb{R}_0^+$
- $\mathcal{I}$ is a *model* of rule base $\mathcal{R}$ for event stream ϵ, if
  (i) $\langle t, t \rangle \in \mathcal{I}(a)$ for each atom a appearing at t in ϵ, and
  (ii) $\langle t, t' \rangle \in \mathcal{I}(a)$ for each $\langle t, t' \rangle$ matching pt in a rule $a \leftarrow pt$

Example:

- Event stream ϵ: $\epsilon(x) = \{1\}, \epsilon(y) = \{3, 5\}$
- Rule base $\mathcal{R} = \{a \leftarrow x \text{ SEQ } y\}$
- Tuples $\langle 1, 3 \rangle$ and $\langle 1, 5 \rangle$ match $x \text{ SEQ } y$ and must be assigned to a

Outline

Temporal Reasoning

- Starting with Linear Time Logic (LTL) (Pnueli, 1977), various temporal logics have been extensively studied and applied in formal verification of hardware and software
- Prototypical problem in *Model Checking:* given
  - a temporal logic formula φ, describing some property
  - a Kripke structure $M = \langle S, R, L \rangle$, describing a system, where
    - S is a (finite) set of states,
    - $R \subseteq S \times S$ is a set of transitions, and
    - $L : S \to 2^{\mathcal{A}}$ assigns every state s a set $L(s) \subseteq \mathcal{A}$ of propositional atoms

  decide whether φ holds for any path $\pi = s_0, s_1, \ldots$ in M, i.e., $M \models \varphi$

Temporal Reasoning

- Starting with Linear Time Logic (LTL) (Pnueli, 1977), various temporal logics have been extensively studied and applied in formal verification of hardware and software
- Prototypical problem in *Model Checking:* given
  - a temporal logic formula φ, describing some property
  - a Kripke structure $M = \langle S, R, L \rangle$, describing a system, where
    - S is a (finite) set of states,
    - $R \subseteq S \times S$ is a set of transitions, and
    - $L : S \to 2^{\mathcal{A}}$ assigns every state s a set $L(s) \subseteq \mathcal{A}$ of propositional atoms

  decide whether φ holds for any path $\pi = s_0, s_1, \ldots$ in M, i.e., $M \models \varphi$
- Extensions of LTL, such as Metric Temporal Logic (MTL) (Koymans, 1990), allow for expressions with time bounds

Temporal Reasoning

- Starting with Linear Time Logic (LTL) (Pnueli, 1977), various temporal logics have been extensively studied and applied in formal verification of hardware and software
- Prototypical problem in *Model Checking:* given
  - a temporal logic formula φ, describing some property
  - a Kripke structure $M = \langle S, R, L \rangle$, describing a system, where
    - S is a (finite) set of states,
    - $R \subseteq S \times S$ is a set of transitions, and
    - $L : S \rightarrow 2^{\mathcal{A}}$ assigns every state s a set $L(s) \subseteq \mathcal{A}$ of propositional atoms

  decide whether φ holds for any path $\pi = s_0, s_1, \ldots$ in M, i.e., $M \models \varphi$
- Extensions of LTL, such as Metric Temporal Logic (MTL) (Koymans, 1990), allow for expressions with time bounds
- Relation to stream reasoning: sequences of sets of propositional atoms $L(s_0), L(s_1), \ldots$ – corresponding to paths – can be used to represent streams
- Temporal Action Logic (Doherty *et al.*, 2009): builds on MTL, to control drones
- DyKnow (Heintz *et al.*, 2010): one of the first systems that implements continuous reasoning for temporal logics over data streams
  - applications in robotics, chronicle recognition, automatic configuration, etc.

Outline

Prolog

- Procedural semantics of Prolog enables natural handling of streams by adding a timestamp as an additional term to every atom

$$previous(T1, X) \leftarrow msg(T, X), msg(T1, X), T1 < T$$

  - Note: programming becomes more complicated: e.g., infinite evaluations possible
  - Nevertheless, Prolog was successfully used to implement real-time reasoning in ETALIS

Prolog

- Procedural semantics of Prolog enables natural handling of streams by adding a timestamp as an additional term to every atom

$$previous(T1,X) \leftarrow msg(T,X), msg(T1,X), T1 < T$$

  - Note: programming becomes more complicated: e.g., infinite evaluations possible
  - Nevertheless, Prolog was successfully used to implement real-time reasoning in ETALIS

- Temporal Prolog (Hrycej, 1993): implements reified temporal logic (Reichgelt, 1987), provides language constructs for temporal dependencies between atoms

Prolog

- Procedural semantics of Prolog enables natural handling of streams by adding a timestamp as an additional term to every atom

$$previous(T1, X) \leftarrow msg(T, X), msg(T1, X), T1 < T$$

  - Note: programming becomes more complicated: e.g., infinite evaluations possible
  - Nevertheless, Prolog was successfully used to implement real-time reasoning in ETALIS

- Temporal Prolog (Hrycej, 1993): implements reified temporal logic (Reichgelt, 1987), provides language constructs for temporal dependencies between atoms

- Novel proposal: Lazy Stream Programming (Tarau *et al.*, 2019)

  lazy stream = encapsulation of a stream into a mechanism that provides its elements on demand

Prolog

- Procedural semantics of Prolog enables natural handling of streams by adding a timestamp as an additional term to every atom

$$previous(T1, X) \leftarrow msg(T, X), msg(T1, X), T1 < T$$

  - Note: programming becomes more complicated: e.g., infinite evaluations possible
  - Nevertheless, Prolog was successfully used to implement real-time reasoning in ETALIS

- Temporal Prolog (Hrycej, 1993): implements reified temporal logic (Reichgelt, 1987), provides language constructs for temporal dependencies between atoms

- Novel proposal: Lazy Stream Programming (Tarau *et al.*, 2019)

  lazy stream = encapsulation of a stream into a mechanism that provides its elements on demand

  - generic lazy stream generators for stateful computations on finite + infinite sequences
  - answer stream generators: special class supporting AND-streams (forward, by recursion) and OR-streams (backward, by disjunction)
    - can wrap e.g. a socket reader as stream
    - details of operations (open, read, close) remain hidden
  - lazy lists are materialized *on demand*, using attributed variables ⇒ can handle (potentially) infinite streams
  - beneficial for memory consumption (use garbage collection and tabling)

Outline

Datalog for Stream Reasoning

- Datalog is a well-known database query language with rules of the form:

$$a \leftarrow b_1, \ldots, b_n, \quad 0 \leq n$$

  where a is an atom and b_i are literals (atoms or negated atoms)
- Historically, multiple extensions of Datalog considered continuously changing data
  - Datalog LITE (Gottlob *et al.*, 2002): deductive query language that views Kripke structures as relational databases on which programs can be evaluated
  - DEDALUS (Alvaro *et al.*, 2010): extends Datalog with an explicit notion of time by augmenting all atoms with timestamps and providing corresponding reasoning algorithms
  - Temporal Datalog (Orgun and Wadge, 1992): based on temporal relational algebra which can be used to model temporal relations among data without explicit reference to time

Datalog for Stream Reasoning

- Datalog is a well-known database query language with rules of the form:

$$a \leftarrow b_1, \ldots, b_n, \quad 0 \leq n$$

  where a is an atom and b_i are literals (atoms or negated atoms)
- Historically, multiple extensions of Datalog considered continuously changing data
    - Datalog LITE (Gottlob *et al.*, 2002): deductive query language that views Kripke structures as relational databases on which programs can be evaluated
    - DEDALUS (Alvaro *et al.*, 2010): extends Datalog with an explicit notion of time by augmenting all atoms with timestamps and providing corresponding reasoning algorithms
    - Temporal Datalog (Orgun and Wadge, 1992): based on temporal relational algebra which can be used to model temporal relations among data without explicit reference to time
- Early uses of Datalog are in temporal databases (Baudinet *et al.*, 1993), e.g., pattern mining (Padmanabhan and Tuzhilin, 1996), or model checking (Datalog LITE)
- Argument for time (Ronca *et al.*, 2018) or state (Lausen *et al.*, 1998) is suggestive

Datalog for Stream Reasoning (cont'd)

- Modern Datalog extensions such as Metric Time Datalog (Brandt *et al.*, 2017) or Streamlog (Zaniolo, 2012), have additional features, e.g. aggregate functions and algorithms for high-performance stream reasoning
- **StreamLog** solves continuous cumulative evaluation of blocking queries (BLQ) using a Progressive Closing World Assumption (PCWA) on timestamped-ordered stream and database facts

Datalog for Stream Reasoning (cont'd)

- Modern Datalog extensions such as Metric Time Datalog (Brandt *et al.*, 2017) or Streamlog (Zaniolo, 2012), have additional features, e.g. aggregate functions and algorithms for high-performance stream reasoning

- **StreamLog** solves continuous cumulative evaluation of blocking queries (BLQ) using a Progressive Closing World Assumption (PCWA) on timestamped-ordered stream and database facts

Example Q1 can be answered at time T (refer only to past) whereas Q2 is **blocking** ⇒ unable to produce output tuples until the entire input is seen

$$Q1: \quad repeated(T, X) \leftarrow msg(T, X), msg(T0, X), T > T0$$

$$Q2: \quad last(T, Z) \leftarrow msg(T, Z), \neg next(T, Z).$$
$$next(T, Z) \leftarrow msg(T1, Z), T1 > T.$$

Datalog for Stream Reasoning (cont'd)

- Modern Datalog extensions such as Metric Time Datalog (Brandt *et al.*, 2017) or Streamlog (Zaniolo, 2012), have additional features, e.g. aggregate functions and algorithms for high-performance stream reasoning

- **StreamLog** solves continuous cumulative evaluation of blocking queries (BLQ) using a Progressive Closing World Assumption (PCWA) on timestamped-ordered stream and database facts

Example Q1 can be answered at time T (refer only to past) whereas Q2 is **blocking** $\Rightarrow$ unable to produce output tuples until the entire input is seen

$$Q1: \quad repeated(T,X) \leftarrow msg(T,X), msg(T0,X), T > T0$$

$$Q2: \quad last(T,Z) \leftarrow msg(T,Z), \neg next(T,Z).$$
$$next(T,Z) \leftarrow msg(T1,Z), T1 > T.$$

- PCWA can be enforced using local stratification on time $\Rightarrow$ Streamlog
- Possible to express e.g. shortest path queries on streaming arcs
- For Sequential Programs, the unique stable model can be computed by fixpoint iteration over bi-states (old + new predicate versions)

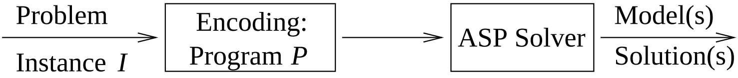
Outline

Recall: Answer Set Programming

- Answer Set Programming (ASP) is a widely used approach to declarative solving of hard combinatorial (optimization) problems

General idea: answer sets are solutions!

Solving a problem instance I by computing answer sets



- **Method:**
  1. *encode* I as a (non-monotonic) logic program P, such that solutions of I are represented by models of P
  2. *compute* some model M of P, using an ASP solver
  3. *extract* a solution for I from M.

  variant: compute multiple/all models (for multiple/all solutions)
- **Common:** decompose I into *problem specification* and *data*
- **Approach:** *guess & check* (aka *generate & test*) plus *auxiliary defs*

- Versatile knowledge representation language supporting default negation, aggregates, external functions, etc., combined with high-performance solvers
- Applications in various domains: configuration, call routing, workspace management, etc. [AI Magazine, 2016; KI 2018, special issues on ASP]

Multi-shot Solving

- ASP solvers supporting various extensions over rich APIs, e.g., clingo[7] or wasp[8], that allow programmers to solve problems iteratively[9]
- **Multi-shot solving**: execute "ground & solve" for different parts of a program/instance while preserving internal state of the solver
  - *Continuously solve changing logic programs*!
- Iterative grounding and solving is based on composing modules (P_i, I_i, O_i), where
  - P_i is a (ground) program to be solved in the ith iteration, and
  - I_i, O_i are sets of ground atoms representing input resp. output atoms
- Adding new ground modules to the solver is easy, but removal is hard

[7] `https://potassco.org/`

[8] `https://github.com/alviano/wasp`

[9] Historically, multi-shot solving was first implemented in *oclingo*, which is a version of *clingo*.

Multi-shot Solving

- ASP solvers supporting various extensions over rich APIs, e.g., clingo[7] or wasp[8], that allow programmers to solve problems iteratively[9]
- **Multi-shot solving**: execute "ground & solve" for different parts of a program/instance while preserving internal state of the solver
  - *Continuously solve changing logic programs*!
- Iterative grounding and solving is based on composing modules (P_i, I_i, O_i), where
  - P_i is a (ground) program to be solved in the ith iteration, and
  - I_i, O_i are sets of ground atoms representing input resp. output atoms
- Adding new ground modules to the solver is easy, but removal is hard
- **Solution:** external atoms in rule bodies whose truth values are set via an API
  - If a rule must be removed, set the corresponding external atom to false
  - High memory consumption due to impossibility of deleting any rules from memory
  - $\Rightarrow$ Restart the solver when a predefined amount of memory is allocated
  - $\Rightarrow$ Write programs communicating all changes via external atoms (avoid grounding)
  - $\Rightarrow$ use systems supporting external predicates like in dvlhex (Eiter *et al.*, 2018) to avoid grounding problems

---

[7] `https://potassco.org/`

[8] `https://github.com/alviano/wasp`

[9] Historically, multi-shot solving was first implemented in *oclingo*, which is a version of *clingo*.

Stream Reasoning with ASP

- **Multi-shot solving**: natural to implement stream reasoning using ASP (Beck *et al.*, 2017; Obermeier *et al.*, 2019)
  - Careful programs design is essential as large modules need lots of memory
  - Using external predicates and external functions is essential for efficient handling of temporal dependencies between atoms
- **Single-shot solving**: use if search for an answer set can be done in parallel

Stream Reasoning with ASP

- **Multi-shot solving**: natural to implement stream reasoning using ASP (Beck *et al.*, 2017; Obermeier *et al.*, 2019)
  - Careful programs design is essential as large modules need lots of memory
  - Using external predicates and external functions is essential for efficient handling of temporal dependencies between atoms
- **Single-shot solving**: use if search for an answer set can be done in parallel
- StreamRule (Mileo *et al.*, 2013) uses ASP for continuous reevaluation of a program encoding a query to an RDF stream
  - Pham *et al.* (2019b) parallelize evaluation of programs by analyzing their extended dependency graphs (EDG) indicating dependencies between predicates
  - A stratified ASP program can be instantiated for different sets of input predicates, by respecting connected components in EDG

$$q(X) \leftarrow p(X), r(X,Y) \quad q(X) \leftarrow p(X), s(X)$$

  - The sets are obtained by analyzing dependencies between sinks (indegree=0)

component $\{p, r, s\}$, may be split into $\{p, r\}$ and $\{p, s\}$ (heuristics)

Stream Reasoning with ASP

- **Multi-shot solving**: natural to implement stream reasoning using ASP (Beck *et al.*, 2017; Obermeier *et al.*, 2019)
  - Careful programs design is essential as large modules need lots of memory
  - Using external predicates and external functions is essential for efficient handling of temporal dependencies between atoms
- **Single-shot solving**: use if search for an answer set can be done in parallel
- StreamRule (Mileo *et al.*, 2013) uses ASP for continuous reevaluation of a program encoding a query to an RDF stream
  - Pham *et al.* (2019b) parallelize evaluation of programs by analyzing their extended dependency graphs (EDG) indicating dependencies between predicates
  - A stratified ASP program can be instantiated for different sets of input predicates, by respecting connected components in EDG

$$q(X) \leftarrow p(X), r(X, Y) \quad q(X) \leftarrow p(X), s(X)$$

  - The sets are obtained by analyzing dependencies between sinks (indegree=0)

    component $\{p, r, s\}$, may be split into $\{p, r\}$ and $\{p, s\}$ (heuristics)

- C-ASP (Pham *et al.*, 2019a): reasoning over RDF streams, a query language extending standard ASP-core with
  - time- and tuple-based windows,
  - stream management directives, and
  - triggering functions

Outline

reactive MCS

reactive Multi-Context Systems Brewka *et al.* (2018)

- based on managed Multi-Context Systems Brewka *et al.* (2011b)
- preliminary version got presented at ECAI 2014 Brewka *et al.* (2014)
- evolving Multi-Context Systems at ECAI 2014 Gonçalves *et al.* (2014)

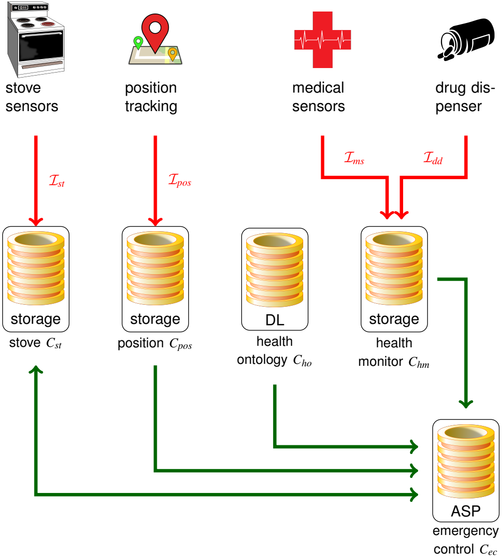
$\Rightarrow$ complete redefinition of rMCS

Current reactive Multi-Context Systems

- streamlined definitions
- a generalisation of managed Multi-Context Systems
- declarative and operative bridge rules
- results on inconsistency management
- results on complexity
- results on simulating other approaches

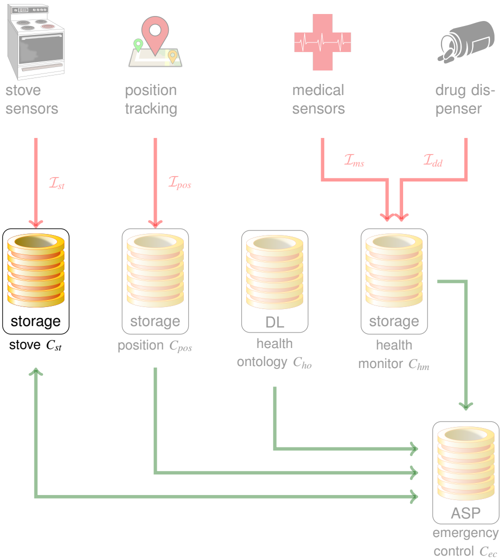
Syntax

Building Blocks



Syntax

Building Blocks



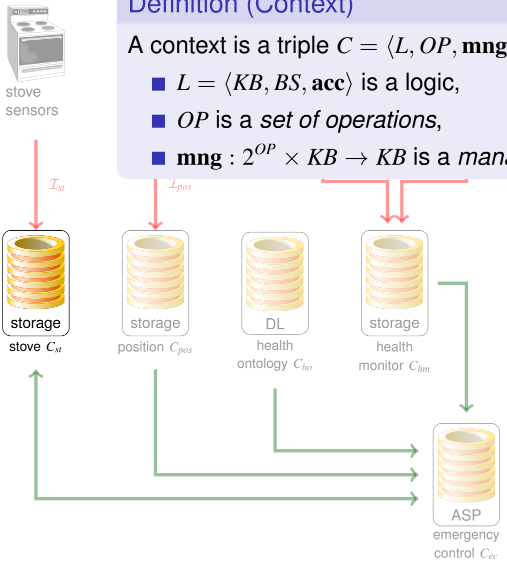
Syntax

Building Blocks

Definition (Context)

A context is a triple $C = \langle L, OP, \mathbf{mng} \rangle$ where

- $L = \langle KB, BS, \mathbf{acc} \rangle$ is a logic,
- OP is a *set of operations*,
- $\mathbf{mng} : 2^{OP} \times KB \to KB$ is a *management function*.



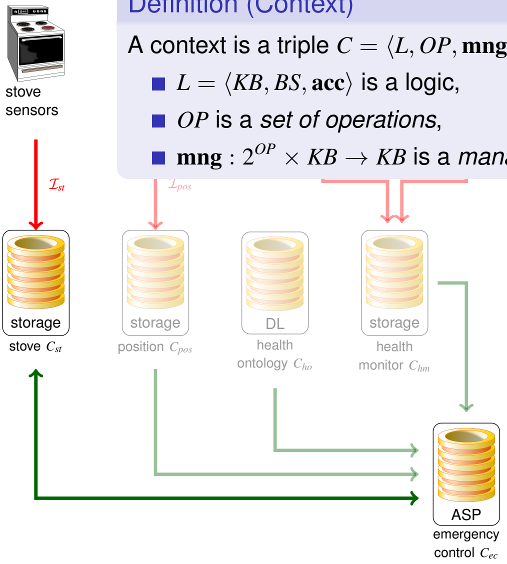
Syntax

Building Blocks

Definition (Context)

A context is a triple $C = \langle L, OP, \mathbf{mng} \rangle$ where

- $L = \langle KB, BS, \mathbf{acc} \rangle$ is a logic,
- OP is a *set of operations*,
- $\mathbf{mng} : 2^{OP} \times KB \to KB$ is a *management function*.



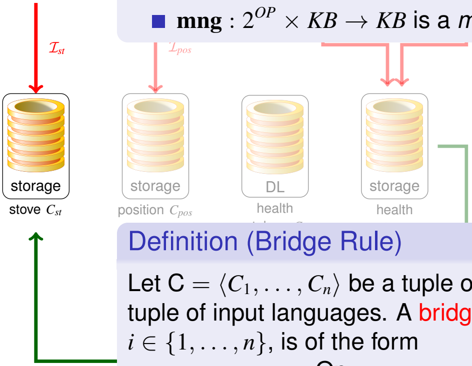
Syntax

Building Blocks

Definition (Context)

A context is a triple $C = \langle L, OP, \mathbf{mng} \rangle$ where

- $L = \langle KB, BS, \mathbf{acc} \rangle$ is a logic,
- OP is a *set of operations*,
- $\mathbf{mng} : 2^{OP} \times KB \rightarrow KB$ is a *management function*.



Definition (Bridge Rule)

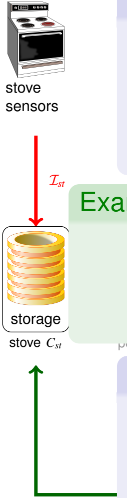
Let C $= \langle C_1, \ldots, C_n \rangle$ be a tuple of contexts and IL $= \langle IL_1, \ldots, IL_k \rangle$ a tuple of input languages. A bridge rule for C_i over C and IL, $i \in \{1, \ldots, n\}$, is of the form

$$\text{Op} \leftarrow a_1, \ldots, a_j, \mathit{not}\ a_{j+1}, \ldots, \mathit{not}\ a_m \text{ or}$$

$$\mathbf{next}(\text{Op}) \leftarrow a_1, \ldots, a_j, \mathit{not}\ a_{j+1}, \ldots, \mathit{not}\ a_m$$

Syntax

Building Blocks



Definition (Context)

A context is a triple $C = \langle L, OP, \mathbf{mng} \rangle$ where

- $L = \langle KB, BS, \mathbf{acc} \rangle$ is a logic,
- OP is a *set of operations*,
- $\mathbf{mng} : 2^{OP} \times KB \rightarrow KB$ is a *management function*.

Example

$$\text{setTemp}(hot) \leftarrow st{::}\text{tmp}(T), 42 < T$$
$$\mathbf{next}(\text{setPower}(off)) \leftarrow ec{:}\text{turnOff}(stove)$$
$$\mathbf{next}(\text{setPower}(off)) \leftarrow st{::}\text{switch}, st{:}\text{pw}$$

Definition (Bridge Rule)

Let C $= \langle C_1, \ldots, C_n \rangle$ be a tuple of contexts and IL $= \langle IL_1, \ldots, IL_k \rangle$ a tuple of input languages. A bridge rule for C_i over C and IL, $i \in \{1, \ldots, n\}$, is of the form

$$\text{Op} \leftarrow a_1, \ldots, a_j, not\ a_{j+1}, \ldots, not\ a_m \text{ or}$$
$$\mathbf{next}(\text{Op}) \leftarrow a_1, \ldots, a_j, not\ a_{j+1}, \ldots, not\ a_m$$

Syntax

Definition (Reactive Multi-Context System)

A *reactive Multi-Context System* is a tuple $M = \langle \mathsf{C}, \mathsf{IL}, \mathsf{BR} \rangle$, where

- $\mathsf{C} = \langle C_1, \ldots, C_n \rangle$ is a tuple of contexts;
- $\mathsf{IL} = \langle IL_1, \ldots, IL_k \rangle$ is a tuple of input languages;
- $\mathsf{BR} = \langle BR_1, \ldots, BR_n \rangle$ is a tuple such that each BR_i, $i \in \{1, \ldots, n\}$, is a set of bridge rules for C_i over C and IL.

Semantics

Current Snapshot

Definition (Configuration of Knowledge Bases)

Let $M = \langle \mathsf{C}, \mathsf{IL}, \mathsf{BR} \rangle$ be an rMCS, such that $\mathsf{C} = \langle C_1, \ldots, C_n \rangle$. A configuration of knowledge bases for M is a tuple $\mathsf{KB} = \langle kb_1, \ldots, kb_n \rangle$, such that $kb_i \in KB_i$, for each $i \in \{1, \ldots, n\}$. We use Con_M to denote the set of all configurations of knowledge bases for M.

Definition (Belief State)

Let $M = \langle \langle C_1, \ldots, C_n \rangle, \mathsf{IL}, \mathsf{BR} \rangle$ be an rMCS. Then, a belief state for M is a tuple $\mathsf{B} = \langle B_1, \ldots, B_n \rangle$ such that $B_i \in BS_i$, for each $i \in \{1, \ldots, n\}$. We use Bel_M to denote the set of all belief states for M.

Definition (Input)

Let $M = \langle \mathsf{C}, \langle IL_1, \ldots, IL_k \rangle, \mathsf{BR} \rangle$ be an rMCS. Then an input for M is a tuple $\mathsf{I} = \langle I_1, \ldots, I_k \rangle$ such that $I_i \subseteq IL_i$, $i \in \{1, \ldots, k\}$. The *set of all inputs for* M is denoted by Inp_M.

Semantics
One-Shot Reasoning

- Only utilise Declarative Bridge Rules
- A belief state is an Equilibrium if
  - the updated knowledge base
    (i.e. the management function result on the belief state, the input, and the current configuration)
  - has as the belief state one of the accepted belief states
    (i.e. it is part of the deductive closure of the semantics)

Semantics

One-Shot Reasoning

- Only utilise Declarative Bridge Rules
- A belief state is an Equilibrium if
  - the updated knowledge base
    (i.e. the management function result on the belief state, the input, and the current configuration)
  - has as the belief state one of the accepted belief states
    (i.e. it is part of the deductive closure of the semantics)

Definition (Equilibrium)

Let $M = \langle\langle C_1, \ldots, C_n\rangle, \mathsf{IL}, \mathsf{BR}\rangle$ be an rMCS, $\mathsf{KB} = \langle kb_1, \ldots, kb_n\rangle$ a configuration of knowledge bases for M, and I an input for M. Then, a belief state $\mathsf{B} = \langle B_1, \ldots, B_n\rangle$ for M is an equilibrium of M given KB and I if, for each $i \in \{1, \ldots, n\}$, we have that

$$B_i \in \mathbf{acc}_i(kb'), \text{ where } kb' = \mathbf{mng}_i(app_i^{now}(\mathsf{I}, \mathsf{B}), kb_i).$$

Semantics

Reactive Reasoning

- Extend the concept of the Input, to be an Input Stream
- Operative Bridge Rules allow configuration changes
- Updates are based on the previously computed Equilibrium
- Results represented as Equilibria Stream and its dual Configuration Stream

Semantics

Reactive Reasoning

Definition (Update Function)

Let $M = \langle \mathsf{C}, \mathsf{IL}, \mathsf{BR} \rangle$ be an rMCS such that $\mathsf{C} = \langle C_1, \ldots, C_n \rangle$, $\mathsf{KB} = \langle kb_1, \ldots, kb_n \rangle$ a configuration of knowledge bases for M, I an input for M, and B a belief state for M. Then, $\mathbf{upd}_M(\mathsf{KB}, \mathsf{I}, \mathsf{B}) = \langle kb'_1, \ldots, kb'_n \rangle$ is the update function for M, such that for each $i \in \{1 \ldots, n\}$, $kb'_i = \mathbf{mng}_i(app_i^{next}(\mathsf{I}, \mathsf{B}), kb_i)$ holds.

Definition (Input Stream)

Let $M = \langle \mathsf{C}, \mathsf{IL}, \mathsf{BR} \rangle$ be an rMCS such that $\mathsf{IL} = \langle IL_1, \ldots, IL_k \rangle$. An input stream for M (until τ) is a function $\mathcal{I} : [1..\tau] \to \mathsf{Inp}_M$ where $\tau \in \mathbb{N} \cup \{\infty\}$.

Semantics

Equilibria Stream

Definition (Equilibria Stream)

Let $M = \langle \mathsf{C}, \mathsf{IL}, \mathsf{BR} \rangle$ be an rMCS, KB a configuration of knowledge bases for M, and $\mathcal{I}$ an input stream for M until τ where $\tau \in \mathbb{N} \cup \{\infty\}$. Then, an equilibria stream of M given KB and $\mathcal{I}$ is a function $\mathcal{B} : [1..\tau] \rightarrow \mathsf{Bel}_M$ such that

- $\mathcal{B}^t$ is an equilibrium of M given $\mathcal{KB}^t$ and $\mathcal{I}^t$, where $\mathcal{KB}^t$ is inductively defined as
  - $\mathcal{KB}^1 = \mathsf{KB}$
  - $\mathcal{KB}^{t+1} = \mathbf{upd}_M(\mathcal{KB}^t, \mathcal{I}^t, \mathcal{B}^t)$.

In a dual manner, we will refer to the function $\mathcal{KB} : [1..\tau] \rightarrow \mathsf{Con}_M$ as the *configurations stream of M given* KB*, $\mathcal{I}$, and $\mathcal{B}$*.

Semantics

Partial Equilibria Stream

Definition (Partial Equilibria Stream)

Let $M = \langle \mathsf{C}, \mathsf{IL}, \mathsf{BR} \rangle$ be an rMCS, $\mathsf{KB} = \langle kb_1, \ldots, kb_n \rangle$ a configuration of knowledge bases for M, and $\mathcal{I}$ an input stream for M until τ where $\tau \in \mathbb{N} \cup \{\infty\}$. Then, a partial equilibria stream of M given KB and $\mathcal{I}$ is a partial function $\mathcal{B} : [1..\tau] \nrightarrow \mathsf{Bel}_M$ such that

- $\mathcal{B}^t$ is an equilibrium of M given $\mathcal{KB}^t$ and $\mathcal{I}^t$,
- or $\mathcal{B}^t$ is undefined.

$\mathcal{KB}^t$ is inductively defined as

- $\mathcal{KB}^1 = \mathsf{KB}$
- $\mathcal{KB}^{t+1} = \begin{cases} \mathbf{upd}_M(\mathcal{KB}^t, \mathcal{I}^t, \mathcal{B}^t), & \text{if } \mathcal{B}^t \text{ is not undefined.} \\ \mathcal{KB}^t, & \text{otherwise.} \end{cases}$

Modelling Aspects

Simple Tasks

- Flipping data (self-dependent)
- Handling time
- Windows
- Forgetting

Declarative and Operational Bridge Rules

Example

Flip the power for the stove if a switch is pressed.

Declarative and Operational Bridge Rules

Example

Flip the power for the stove if a switch is pressed.

Declarative approach

- setPower$(\mathit{off}) \leftarrow \mathit{st}::\mathrm{switch}, \mathit{st}{:}\mathrm{pw}$
- setPower$(\mathit{on}) \leftarrow \mathit{st}::\mathrm{switch}, \mathit{not}\ \mathit{st}{:}\mathrm{pw}$

Declarative and Operational Bridge Rules

Example

Flip the power for the stove if a switch is pressed.

Declarative approach

- setPower$(\mathit{off}) \leftarrow \mathit{st}::\mathrm{switch}, \mathit{st}{:}\mathrm{pw}$
- setPower$(\mathit{on}) \leftarrow \mathit{st}::\mathrm{switch}, \mathit{not}\ \mathit{st}{:}\mathrm{pw}$
- No Equilibrium can be found

Declarative and Operational Bridge Rules

Example

Flip the power for the stove if a switch is pressed.

Declarative approach

- $\text{setPower}(\mathit{off}) \leftarrow \mathit{st}{::}\text{switch}, \mathit{st}{:}\text{pw}$
- $\text{setPower}(\mathit{on}) \leftarrow \mathit{st}{::}\text{switch}, \mathit{not}\ \mathit{st}{:}\text{pw}$
- No Equilibrium can be found

Operational approach

- $\mathbf{next}(\text{setPower}(\mathit{off})) \leftarrow \mathit{st}{::}\text{switch}, \mathit{st}{:}\text{pw}$
- $\mathbf{next}(\text{setPower}(\mathit{on})) \leftarrow \mathit{st}{::}\text{switch}, \mathit{not}\ \mathit{st}{:}\text{pw}$

Declarative and Operational Bridge Rules

Example

Flip the power for the stove if a switch is pressed.

Declarative approach

- $\text{setPower}(\mathit{off}) \leftarrow \mathit{st}::\text{switch}, \mathit{st}:\text{pw}$
- $\text{setPower}(\mathit{on}) \leftarrow \mathit{st}::\text{switch}, \mathit{not}\ \mathit{st}:\text{pw}$
- No Equilibrium can be found

Operational approach - without sensor data

- $\text{add}(\mathit{switchpower}) \leftarrow \mathit{st}::\text{switch}$
- $\mathbf{next}(\text{setPower}(\mathit{off})) \leftarrow \mathit{st}:\text{switchpower}, \mathit{st}:\text{pw}$
- $\mathbf{next}(\text{setPower}(\mathit{on})) \leftarrow \mathit{st}:\text{switchpower}, \mathit{not}\ \mathit{st}:\text{pw}$

Handling Time

Possible ways

- Sensor
- Time-Context

Time Context

$$\begin{aligned} \mathsf{setTime}(\mathrm{now}(\mathit{0})) &\leftarrow \mathit{not\ clock}{:}\mathit{timeAvailable} \\ \mathbf{next}(\mathit{add}(\mathrm{timeAvailable})) &\leftarrow \mathit{clock}{:}\mathrm{now}(\mathit{0}) \\ \mathbf{next}(\mathit{setTime}(\mathrm{now}(T+\mathit{1}))) &\leftarrow \mathit{clock}{:}\mathrm{now}(T) \end{aligned}$$

Forgetting and Windowing

Volatile Information and Reasoning with a Window

$$
\begin{aligned}
\mathbf{next}(\mathsf{add}(\text{alert}(\textit{stove}, T))) &\leftarrow c{::}\text{now}(T), \textit{ec}{:}\text{alert}(\textit{stove}). \\
\mathbf{next}(\mathsf{del}(\text{alert}(\textit{stove}, T))) &\leftarrow \textit{stE}{:}\text{alert}(\textit{stove}, T), \textit{not } \textit{ec}{:}\text{alert}(\textit{stove}). \\
\mathsf{add}(\text{emergency}(\textit{stove})) &\leftarrow c{::}\text{now}(T), \textit{ec}{:}\text{alert}(\textit{stove}), \\
&\quad\ \textit{stE}{:}\text{alert}(\textit{stove}, T'), \\
&\quad\ \textit{stE}{:}\text{winE}(Y), |T - T'| \geq Y.
\end{aligned}
$$

Dynamic Window

$$
\begin{aligned}
\mathbf{next}(\mathsf{set}(\text{win}(P, X))) &\leftarrow \textit{ed}{:}\text{defWin}(P, X), \textit{not } \textit{ed}{:}\text{susp}(E). \\
\mathbf{next}(\mathsf{set}(\text{win}(P, Y))) &\leftarrow \textit{ed}{:}\text{rel}(P, E, Y), \textit{ed}{:}\text{susp}(E). \\
\mathsf{alarm}(E) &\leftarrow \textit{ed}{:}\text{conf}(E).
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{next}(\text{add}(\text{P}(T))) &\leftarrow c{::}\text{now}(T), s{::}P. \\
\mathbf{next}(\text{del}(\text{P}(T'))) &\leftarrow \textit{ed}{:}\text{P}(T'), c{::}\text{now}(T), \textit{ed}{:}\text{win}(P, Z), T' < T - Z.
\end{aligned}
$$

Outline

reactive Multi-Context Systems so far ...

Motivation

- **integration** of heterogeneous KR-formalisms
- **awareness** of continuous flow of knowledge

Realisation

- Contexts with different KR & Reasoning formalisms
- Bridge-Rules for exchange of beliefs
- Notion of Equilibrium as Semantics (**“synchron”**)
- Run represents the change of knowledge and belief over time

... and a slight look to online-applications

- Many different services and sources of knowledge
- Continuous flow of information
- Data collection till sufficient knowledge for their tasks is available
- Communication is often query-based
- Asynchronous communication protocols

Computer Aided Emergency Team Management

Example Environment - Emergency Team Management

- Emergency Call
- Classification and Prioritisation of each case
- Overview of available rescue units
- Overview on ETAs for each unit and case
- Suggesting optimal assignments
- Communicate Tasks to rescue units

Requirements

- Fast response to events
- Consider different sources of data
- Modularity for additional components
- Human as last instance for decisions

Consequences of Asynchronicity

- Contexts compute their belief sets independently
- No agreement on a common Equilibrium
- No defined basis for Bridge-Rules to be applicable
- Need for Output-Rules (OR)
- Keep track of information provided by OR
- Input stream for each context
- Interaction with environment:
  - aMCS wide input streams
  - aMCS wide output streams

Other Design-Choices

- Each context decide when to compute
  - realised by computation controller
- Dynamic adjustments of context-management
  - computation controller (cc)
  - output rules (OR)
  - context-semantics ($\mathbf{acc}$)
  - context update function ($\mathbf{cu}$)
- Logic suite

asynchronous Multi-Context Systems

Ellmauthaler and Pührer (2015); Ellmauthaler (2018)

Definition

A data package is a pair D $= \langle s, I \rangle$, where $s \in N$ is either a context name or a sensor name, stating the source of D, and $I \subseteq IL$ is a set of pieces of information. An information buffer is a sequence of data packages.

asynchronous Multi-Context Systems

Ellmauthaler and Pührer (2015); Ellmauthaler (2018)

Definition

A data package is a pair D $= \langle s, I \rangle$, where $s \in N$ is either a context name or a sensor name, stating the source of D, and $I \subseteq IL$ is a set of pieces of information. An information buffer is a sequence of data packages.

Definition

Let $C = \langle \mathrm{n}, LS \rangle$ be a context. An output rule r for C is an expression of the form

$$\langle \mathrm{n}, \mathrm{i} \rangle \leftarrow b_1, \ldots, b_j, \neg\, b_{j+1}, \ldots, \neg\, b_m, \quad (1)$$

such that n $\in N$ is the name of a context or an output stream, i $\in IL$ is a piece of information, and every b_ℓ $(1 \leq \ell \leq m)$ is a belief for C, i.e. $b_\ell \in B$ for some $B \in BS_{LS}$.

asynchronous Multi-Context Systems

Definition

Let $C = \langle \mathsf{n}, LS \rangle$ be a context, OR a set of output rules for C, $B \in B_{LS}$ a belief set, and $\mathsf{n}' \in N$ a name. Then, the data package

$$d_C(B, OR, \mathsf{n}') = \langle \mathsf{n}, \{i \mid r \in OR, head(r) = \langle \mathsf{n}', \mathsf{i} \rangle, B \models body(r)\} \rangle$$

is the output of C with respect to OR under relevant for n.

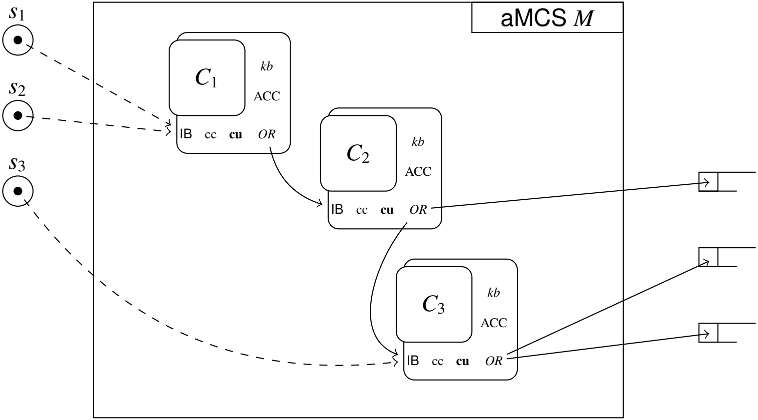
asynchronous Multi-Context Systems

Definition

Let $C = \langle \mathsf{n}, LS \rangle$ be a context. A configuration of C is a tuple $\mathsf{CF} = \langle kb, \mathrm{ACC}, \mathsf{IB}, \mathsf{CM} \rangle$, where $kb \in KBLS$, $\mathrm{ACC} \in ACC_{LS}$, IB is a finite information buffer, and CM is a context management for C which is a triple $\mathsf{CM} = \langle \mathrm{cc}, \mathbf{cu}, OR \rangle$, where

- cc is a computation controller for C,
- OR is a set of output rules for C, and
- $\mathbf{cu}$ is a context update function for C which is a function that maps an information buffer $\mathsf{IB} = \mathsf{D}_1, \ldots, \mathsf{D}_m$ and an admissible knowledge base of LS to a configuration $\mathsf{CF}' = \langle kb', \mathrm{ACC}', \mathsf{IB}', \mathsf{CM}' \rangle$ of C with $\mathsf{IB}' = \mathsf{D}_k, \ldots, \mathsf{D}_m$ for some $k \geq 1$.

asynchronous Multi-Context Systems



Run of an aMCS

Configuration of an aMCS

- Configuration for each Context
- Content of each output stream (output buffer)

Run of an aMCS

Configuration of an aMCS

- Configuration for each Context
- Content of each output stream (output buffer)

Definition (Run structure)

Let $M = \langle\langle C_1, \ldots, C_n\rangle, \langle \mathsf{o}_1, \ldots, \mathsf{o}_m\rangle\rangle$ be an aMCS. A run structure for M is a sequence

$$R = \ldots, \mathsf{CF}^t, \mathsf{CF}^{t+1}, \mathsf{CF}^{t+2}, \ldots \quad ,$$

where $t \in \mathbb{Z}$ is a point in time, and every $\mathsf{CF}^{t'}$ in R ($t' \in \mathbb{Z}$) is a configuration of M.

Run of an aMCS

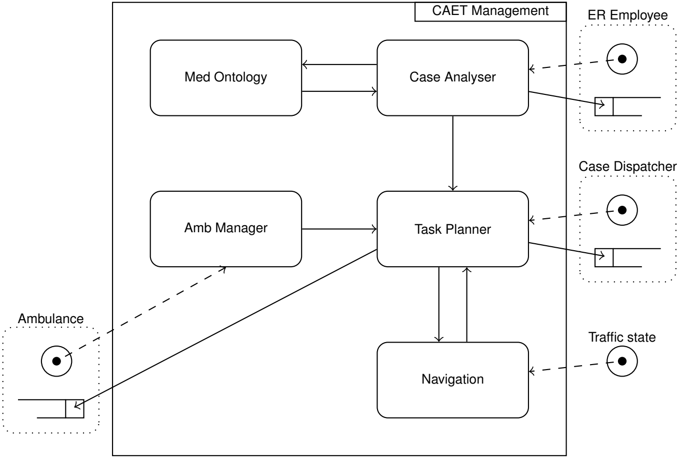
Time-awareness

- Computation of belief sets takes time
- Enumeration of belief sets takes time
- Verification of non-existence of (further) belief sets takes time

Run execution

- If a Context finds a belief set, OR are applied
- Information is distributed to input-buffers of contexts or output streams
- If a Context has finished its computation, $\mathtt{EOC}$ is sent to all stakeholders

Example of an aMCS



Differences to rMCS

- rMCSs use equilibria
  - strong semantics
  - tight integration approach where context semantics are interdependent
  - every context need to agree $\rightarrow$ synchronous approach
- rMCSs have equilibria as source of non-determinism
- aMCSs have computation time as source of non-determinism

Simulation of rMCS

- For each Context C_i of the rMCS, introduce three aMCS Contexts:
  - C_i^{kb} stores its current knowledge base
  - $C_i^{kb'}$ stores update of the knowledge base and compute its semantics
  - C_i^{m} implements the bridge rules and the management function

Simulation of rMCS

- For each Context C_i of the rMCS, introduce three aMCS Contexts:
  - C_i^{kb} stores its current knowledge base
  - $C_i^{kb'}$ stores update of the knowledge base and compute its semantics
  - C_i^{m} implements the bridge rules and the management function
- Three contexts for the rMCS, where
  - C^{obs} receives sensor data and distributes the information,
  - C^{guess} guesses equilibrium candidates and propagates them to C_i^m, and
  - C^{check} compares all results of the contexts and informs other contexts if an equilibrium has been found

Outline

Interval Streams

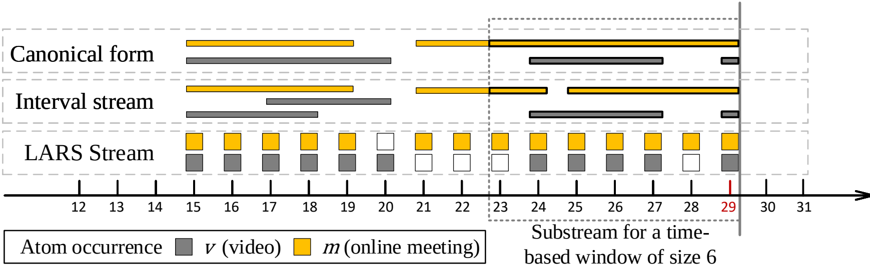
- **Interval stream** is pair $S_I = (T, \eta)$, where
  - T is a timeline and
  - evaluation function $\eta : \mathcal{A} \to 2^{\mathcal{I}(T)}$ is a mapping that assigns to every atom $a \in \mathcal{A}$ a subset of the set of nonempty closed intervals over T, i.e., $\mathcal{I}(T) = \{I = [i,j] \mid I \subseteq T\}$

Interval Streams

- **Interval stream** is pair $S_I = (T, \eta)$, where
  - T is a timeline and
  - evaluation function $\eta : \mathcal{A} \to 2^{\mathcal{I}(T)}$ is a mapping that assigns to every atom $a \in \mathcal{A}$ a subset of the set of nonempty closed intervals over T, i.e., $\mathcal{I}(T) = \{I = [i,j] \mid I \subseteq T\}$
- **Equivalence**: Two interval streams $S_I = (T, \eta)$ and $S'_I = (T', \eta')$ are *equivalent*, if $T = T'$ and for every $a \in \mathcal{A}$, $\bigcup \eta(a) = \bigcup \eta'(a)$

Interval Streams

- **Interval stream** is pair $S_I = (T, \eta)$, where
  - T is a timeline and
  - evaluation function $\eta : \mathcal{A} \to 2^{\mathcal{I}(T)}$ is a mapping that assigns to every atom $a \in \mathcal{A}$ a subset of the set of nonempty closed intervals over T, i.e., $\mathcal{I}(T) = \{I = [i,j] \mid I \subseteq T\}$
- **Equivalence**: Two interval streams $S_I = (T, \eta)$ and $S'_I = (T', \eta')$ are *equivalent*, if $T = T'$ and for every $a \in \mathcal{A}$, $\bigcup \eta(a) = \bigcup \eta'(a)$
- **Mapping** between interval and LARS streams using *canonical form*



Interval Streams, cont'd

- **Substreams**: $S'_I = (T', \eta')$ is a *substream* of $S_I = (T, \eta)$, denoted $S'_I \subseteq S_I$, if
  - $T' \subseteq T$ and
  - for every $I' \in \eta'(a)$, where $a \in \mathcal{A}$, some $I \in \eta(a)$ exists such that $I' \subseteq I$

Interval Streams, cont'd

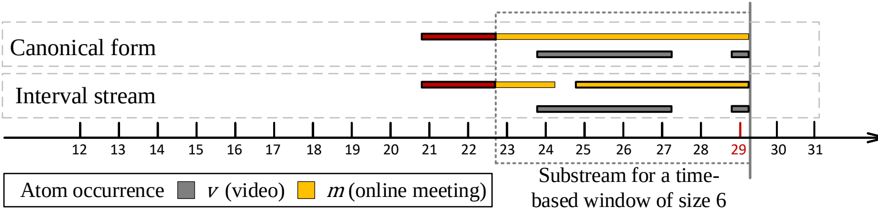
- **Substreams**: $S'_I = (T', \eta')$ is a *substream* of $S_I = (T, \eta)$, denoted $S'_I \subseteq S_I$, if
  - $T' \subseteq T$ and
  - for every $I' \in \eta'(a)$, where $a \in \mathcal{A}$, some $I \in \eta(a)$ exists such that $I' \subseteq I$
- **Window functions** in the interval semantics "crop" intervals

A *window function* w is any (computable) function that given an interval stream $S_I = (T, \eta)$ and a time point t, returns a substream $S'_I = w(S_I, t)$ of S_I

Interval Streams, cont'd

- **Substreams**: $S'_I = (T', \eta')$ is a *substream* of $S_I = (T, \eta)$, denoted $S'_I \subseteq S_I$, if
  - $T' \subseteq T$ and
  - for every $I' \in \eta'(a)$, where $a \in \mathcal{A}$, some $I \in \eta(a)$ exists such that $I' \subseteq I$
- **Window functions** in the interval semantics "crop" intervals

A *window function* w is any (computable) function that given an interval stream $S_I = (T, \eta)$ and a time point t, returns a substream $S'_I = w(S_I, t)$ of S_I



Streaming Atoms

- **Streaming atoms** ($\mathcal{A}^+$) are defined by the grammar:

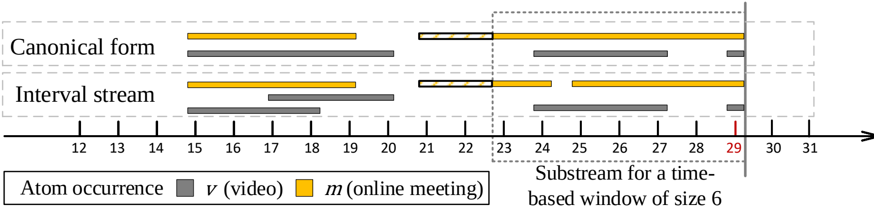
$$a \mid @_t a \mid \boxplus^w @_t a \mid \boxplus^w \Diamond a \mid \boxplus^w \Box a$$

Streaming Atoms

- **Streaming atoms** ($\mathcal{A}^+$) are defined by the grammar:

$$a \mid @_t a \mid \boxplus^w @_t a \mid \boxplus^w \Diamond a \mid \boxplus^w \Box a$$

- **Satisfaction**: Given a structure $M = (S_I, W)$ and the evaluation time point $t \in T$

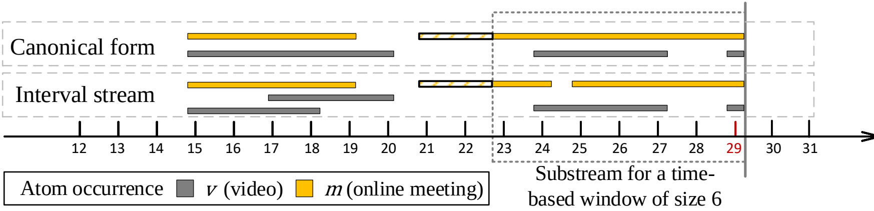


Streaming Atoms

- **Streaming atoms** ($\mathcal{A}^+$) are defined by the grammar:

$$a \mid @_t a \mid \boxplus^w @_t a \mid \boxplus^w \Diamond a \mid \boxplus^w \Box a$$

- **Satisfaction**: Given a structure $M = (S_I, W)$ and the evaluation time point $t \in T$
  - atom $a \in \mathcal{A}$ holds, if $t \in \bigcup \eta(a)$, e.g., m holds at $t = 29$

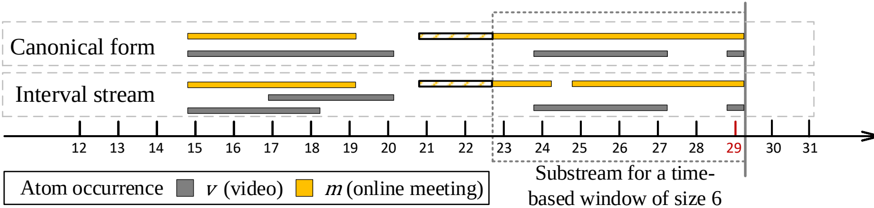


Streaming Atoms

- **Streaming atoms** ($\mathcal{A}^+$) are defined by the grammar:

$$a \mid @_t a \mid \boxplus^w @_t a \mid \boxplus^w \Diamond a \mid \boxplus^w \Box a$$

- **Satisfaction**: Given a structure $M = (S_I, W)$ and the evaluation time point $t \in T$
  - atom $a \in \mathcal{A}$ holds, if $t \in \bigcup \eta(a)$, e.g., m holds at $t = 29$
  - $\Box a$ holds, if $\bigcup \eta(a) = T$, e.g., $\Box m$ does not hold, but $\boxplus^6 \Box m$ does;

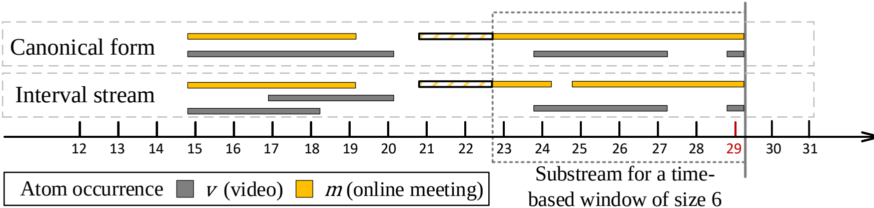


Streaming Atoms

- **Streaming atoms** ($\mathcal{A}^+$) are defined by the grammar:

$$a \mid @_t a \mid \boxplus^w @_t a \mid \boxplus^w \Diamond a \mid \boxplus^w \Box a$$

- **Satisfaction**: Given a structure $M = (S_I, W)$ and the evaluation time point $t \in T$
  - atom $a \in \mathcal{A}$ holds, if $t \in \bigcup \eta(a)$, e.g., m holds at $t = 29$
  - $\Box a$ holds, if $\bigcup \eta(a) = T$, e.g., $\Box m$ does not hold, but $\boxplus^6 \Box m$ does;
  - $\Diamond a$ holds, if $\bigcup \eta(a) \neq \emptyset$, e.g., $\boxplus^6 \Diamond v$ holds;

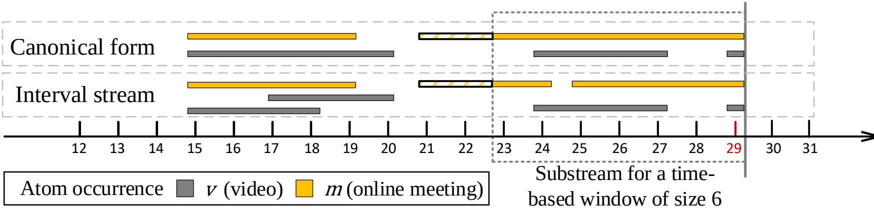


Streaming Atoms

- **Streaming atoms** ($\mathcal{A}^+$) are defined by the grammar:

$$a \mid @_t a \mid \boxplus^w @_t a \mid \boxplus^w \Diamond a \mid \boxplus^w \Box a$$

- **Satisfaction**: Given a structure $M = (S_I, W)$ and the evaluation time point $t \in T$
  - atom $a \in \mathcal{A}$ holds, if $t \in \bigcup \eta(a)$, e.g., m holds at $t = 29$
  - $\Box a$ holds, if $\bigcup \eta(a) = T$, e.g., $\Box m$ does not hold, but $\boxplus^6 \Box m$ does;
  - $\Diamond a$ holds, if $\bigcup \eta(a) \neq \emptyset$, e.g., $\boxplus^6 \Diamond v$ holds;
  - $@_{t'} a$ holds, if $t' \in \bigcup \eta(a)$, e.g., $\boxplus^6 @_{28} v$ does not hold; and

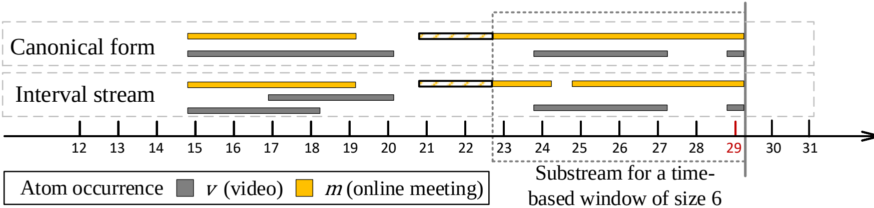


Streaming Atoms

- **Streaming atoms** ($\mathcal{A}^+$) are defined by the grammar:

$$a \mid @_t a \mid \boxplus^w @_t a \mid \boxplus^w \Diamond a \mid \boxplus^w \Box a$$

- **Satisfaction**: Given a structure $M = (S_I, W)$ and the evaluation time point $t \in T$
  - atom $a \in \mathcal{A}$ holds, if $t \in \bigcup \eta(a)$, e.g., m holds at $t = 29$
  - $\Box a$ holds, if $\bigcup \eta(a) = T$, e.g., $\Box m$ does not hold, but $\boxplus^6 \Box m$ does;
  - $\Diamond a$ holds, if $\bigcup \eta(a) \neq \emptyset$, e.g., $\boxplus^6 \Diamond v$ holds;
  - $@_{t'} a$ holds, if $t' \in \bigcup \eta(a)$, e.g., $\boxplus^6 @_{28} v$ does not hold; and
  - $\boxplus^w \alpha$ holds, if α holds for $w(S_I, t)$ at t



Logic-based framework for Analytical Reasoning over Streams (LARS)

LARS is a language for stream reasoning that combines advantages of ASP with *temporal* $\Diamond$, $\Box$, $@_t$ and *window* operators $\boxplus$

Logic-based framework for Analytical Reasoning over Streams (LARS)

LARS is a language for stream reasoning that combines advantages of ASP with *temporal* $\Diamond$, $\Box$, $@_t$ and *window* operators $\boxplus$

- **Plain LARS rules** are of the form
  $r: \ \alpha \leftarrow \beta_1, \ldots, \beta_m, \text{not } \beta_{m+1}, \ldots, \text{not } \beta_n$ where

Logic-based framework for Analytical Reasoning over Streams (LARS)

LARS is a language for stream reasoning that combines advantages of ASP with *temporal* $\diamond$, $\Box$, $@_t$ and *window* operators $\boxplus$

- **Plain LARS rules** are of the form
  $r: \ \alpha \leftarrow \beta_1, \ldots, \beta_m, \text{not } \beta_{m+1}, \ldots, \text{not } \beta_n$ where
  - the head α is an atom a or $@_t a$, and

Logic-based framework for Analytical Reasoning over Streams (LARS)

LARS is a language for stream reasoning that combines advantages of ASP with *temporal* $\Diamond$, $\Box$, $@_t$ and *window* operators $\boxplus$

- **Plain LARS rules** are of the form
  $r: \ \alpha \leftarrow \beta_1, \ldots, \beta_m, \text{not } \beta_{m+1}, \ldots, \text{not } \beta_n$ where
  - the head α is an atom a or $@_t a$, and
  - β_i are streaming atoms that occur in positive $B^+(r) = \{\beta_1, \ldots, \beta_m\}$ or negative $B^-(r) = \{\beta_{m+1}, \ldots, \beta_n\}$ body

Logic-based framework for Analytical Reasoning over Streams (LARS)

LARS is a language for stream reasoning that combines advantages of ASP with *temporal* $\Diamond$, $\Box$, $@_t$ and *window* operators $\boxplus$

- **Plain LARS rules** are of the form
  $r: \ \alpha \leftarrow \beta_1, \ldots, \beta_m, \text{not } \beta_{m+1}, \ldots, \text{not } \beta_n$ where
  - the head α is an atom a or $@_t a$, and
  - β_i are streaming atoms that occur in positive $B^+(r) = \{\beta_1, \ldots, \beta_m\}$ or negative $B^-(r) = \{\beta_{m+1}, \ldots, \beta_n\}$ body
- **Satisfaction**: Given a structure $M = (S_I, W)$ and the evaluation time point $t \in T$, we define that:

Logic-based framework for Analytical Reasoning over Streams (LARS)

LARS is a language for stream reasoning that combines advantages of ASP with *temporal* $\Diamond$, $\Box$, $@_t$ and *window* operators $\boxplus$

- **Plain LARS rules** are of the form
  $r: \ \alpha \leftarrow \beta_1, \ldots, \beta_m, \text{not } \beta_{m+1}, \ldots, \text{not } \beta_n$ where
  - the head α is an atom a or $@_t a$, and
  - β_i are streaming atoms that occur in positive $B^+(r) = \{\beta_1, \ldots, \beta_m\}$ or negative $B^-(r) = \{\beta_{m+1}, \ldots, \beta_n\}$ body
- **Satisfaction**: Given a structure $M = (S_I, W)$ and the evaluation time point $t \in T$, we define that:
  - $M, t \models \alpha$, where $\alpha \in \mathcal{A}^+$, if α holds in S_I at t;

Logic-based framework for Analytical Reasoning over Streams (LARS)

LARS is a language for stream reasoning that combines advantages of ASP with *temporal* $\Diamond$, $\Box$, $@_t$ and *window* operators $\boxplus$

- **Plain LARS rules** are of the form
  $r: \ \alpha \leftarrow \beta_1, \ldots, \beta_m, \text{not } \beta_{m+1}, \ldots, \text{not } \beta_n$ where
  - the head α is an atom a or $@_t a$, and
  - β_i are streaming atoms that occur in positive $B^+(r) = \{\beta_1, \ldots, \beta_m\}$ or negative $B^-(r) = \{\beta_{m+1}, \ldots, \beta_n\}$ body
- **Satisfaction**: Given a structure $M = (S_I, W)$ and the evaluation time point $t \in T$, we define that:
  - $M, t \models \alpha$, where $\alpha \in \mathcal{A}^+$, if α holds in S_I at t;
  - $M, t \models B(r)$ if $M, t \models \beta_i$ and $M, t \not\models \beta_j$, where $\beta_i \in B^+(r)$ and $\beta_j \in B^-(r)$;

Logic-based framework for Analytical Reasoning over Streams (LARS)

LARS is a language for stream reasoning that combines advantages of ASP with *temporal* $\Diamond$, $\Box$, $@_t$ and *window* operators $\boxplus$

- **Plain LARS rules** are of the form
  $r: \ \alpha \leftarrow \beta_1, \ldots, \beta_m, \text{not } \beta_{m+1}, \ldots, \text{not } \beta_n$ where
  - the head α is an atom a or $@_t a$, and
  - β_i are streaming atoms that occur in positive $B^+(r) = \{\beta_1, \ldots, \beta_m\}$ or negative $B^-(r) = \{\beta_{m+1}, \ldots, \beta_n\}$ body
- **Satisfaction**: Given a structure $M = (S_I, W)$ and the evaluation time point $t \in T$, we define that:
  - $M, t \models \alpha$, where $\alpha \in \mathcal{A}^+$, if α holds in S_I at t;
  - $M, t \models B(r)$ if $M, t \models \beta_i$ and $M, t \not\models \beta_j$, where $\beta_i \in B^+(r)$ and $\beta_j \in B^-(r)$;
  - $M, t \models r$ if $M, t \models B(r)$ implies $M, t \models \alpha$;

Logic-based framework for Analytical Reasoning over Streams (LARS)

LARS is a language for stream reasoning that combines advantages of ASP with *temporal* $\Diamond$, $\Box$, $@_t$ and *window* operators $\boxplus$

- **Plain LARS rules** are of the form
  $r: \ \alpha \leftarrow \beta_1, \ldots, \beta_m, \text{not } \beta_{m+1}, \ldots, \text{not } \beta_n$ where
  - the head α is an atom a or $@_t a$, and
  - β_i are streaming atoms that occur in positive $B^+(r) = \{\beta_1, \ldots, \beta_m\}$ or negative $B^-(r) = \{\beta_{m+1}, \ldots, \beta_n\}$ body
- **Satisfaction**: Given a structure $M = (S_I, W)$ and the evaluation time point $t \in T$, we define that:
  - $M, t \models \alpha$, where $\alpha \in \mathcal{A}^+$, if α holds in S_I at t;
  - $M, t \models B(r)$ if $M, t \models \beta_i$ and $M, t \not\models \beta_j$, where $\beta_i \in B^+(r)$ and $\beta_j \in B^-(r)$;
  - $M, t \models r$ if $M, t \models B(r)$ implies $M, t \models \alpha$;
  - $M, t \models \Pi$ if for every rule r in Π it holds that $M, t \models r$

Logic-based framework for Analytical Reasoning over Streams (LARS)

LARS is a language for stream reasoning that combines advantages of ASP with *temporal* $\Diamond$, $\Box$, $@_t$ and *window* operators $\boxplus$

- **Plain LARS rules** are of the form
  $r:\ \alpha \leftarrow \beta_1, \ldots, \beta_m, \text{not } \beta_{m+1}, \ldots, \text{not } \beta_n$ where
  - the head α is an atom a or $@_t a$, and
  - β_i are streaming atoms that occur in positive $B^+(r) = \{\beta_1, \ldots, \beta_m\}$ or negative $B^-(r) = \{\beta_{m+1}, \ldots, \beta_n\}$ body
- **Satisfaction**: Given a structure $M = (S_I, W)$ and the evaluation time point $t \in T$, we define that:
  - $M, t \models \alpha$, where $\alpha \in \mathcal{A}^+$, if α holds in S_I at t;
  - $M, t \models B(r)$ if $M, t \models \beta_i$ and $M, t \not\models \beta_j$, where $\beta_i \in B^+(r)$ and $\beta_j \in B^-(r)$;
  - $M, t \models r$ if $M, t \models B(r)$ implies $M, t \models \alpha$;
  - $M, t \models \Pi$ if for every rule r in Π it holds that $M, t \models r$
- Differentiate between a data stream $D_I = (T, \eta_D)$ of extensional atoms $\mathcal{A}^E$ and an interpretation stream $S_{Int} = (T, \eta)$ for D_I where $\eta(a) = \eta_D(a)$, for all $a \in \mathcal{A}^E$.

Logic-based framework for Analytical Reasoning over Streams (LARS)

LARS is a language for stream reasoning that combines advantages of ASP with *temporal* $\Diamond$, $\Box$, $@_t$ and *window* operators $\boxplus$

- **Plain LARS rules** are of the form
  $r: \ \alpha \leftarrow \beta_1, \ldots, \beta_m, \text{not } \beta_{m+1}, \ldots, \text{not } \beta_n$ where
  - the head α is an atom a or $@_t a$, and
  - β_i are streaming atoms that occur in positive $B^+(r) = \{\beta_1, \ldots, \beta_m\}$ or negative $B^-(r) = \{\beta_{m+1}, \ldots, \beta_n\}$ body
- **Satisfaction**: Given a structure $M = (S_I, W)$ and the evaluation time point $t \in T$, we define that:
  - $M, t \models \alpha$, where $\alpha \in \mathcal{A}^+$, if α holds in S_I at t;
  - $M, t \models B(r)$ if $M, t \models \beta_i$ and $M, t \not\models \beta_j$, where $\beta_i \in B^+(r)$ and $\beta_j \in B^-(r)$;
  - $M, t \models r$ if $M, t \models B(r)$ implies $M, t \models \alpha$;
  - $M, t \models \Pi$ if for every rule r in Π it holds that $M, t \models r$
- Differentiate between a data stream $D_I = (T, \eta_D)$ of extensional atoms $\mathcal{A}^E$ and an interpretation stream $S_{Int} = (T, \eta)$ for D_I where $\eta(a) = \eta_D(a)$, for all $a \in \mathcal{A}^E$.
- **Answer stream** for a data stream $D_I = (T, \eta_D)$ and a plain LARS program Π is an interpretation stream $S_{Int} = (T, \eta)$ such that

Logic-based framework for Analytical Reasoning over Streams (LARS)

LARS is a language for stream reasoning that combines advantages of ASP with *temporal* $\Diamond$, $\Box$, $@_t$ and *window* operators $\boxplus$

- **Plain LARS rules** are of the form
  $r: \ \alpha \leftarrow \beta_1, \ldots, \beta_m, \text{not } \beta_{m+1}, \ldots, \text{not } \beta_n$ where
  - the head α is an atom a or $@_t a$, and
  - β_i are streaming atoms that occur in positive $B^+(r) = \{\beta_1, \ldots, \beta_m\}$ or negative $B^-(r) = \{\beta_{m+1}, \ldots, \beta_n\}$ body
- **Satisfaction**: Given a structure $M = (S_I, W)$ and the evaluation time point $t \in T$, we define that:
  - $M, t \models \alpha$, where $\alpha \in \mathcal{A}^+$, if α holds in S_I at t;
  - $M, t \models B(r)$ if $M, t \models \beta_i$ and $M, t \not\models \beta_j$, where $\beta_i \in B^+(r)$ and $\beta_j \in B^-(r)$;
  - $M, t \models r$ if $M, t \models B(r)$ implies $M, t \models \alpha$;
  - $M, t \models \Pi$ if for every rule r in Π it holds that $M, t \models r$
- Differentiate between a data stream $D_I = (T, \eta_D)$ of extensional atoms $\mathcal{A}^E$ and an interpretation stream $S_{Int} = (T, \eta)$ for D_I where $\eta(a) = \eta_D(a)$, for all $a \in \mathcal{A}^E$.
- **Answer stream** for a data stream $D_I = (T, \eta_D)$ and a plain LARS program Π is an interpretation stream $S_{Int} = (T, \eta)$ such that
  - $S_{Int}, t \models \Pi$, and

Logic-based framework for Analytical Reasoning over Streams (LARS)

LARS is a language for stream reasoning that combines advantages of ASP with *temporal* $\diamond$, $\Box$, $@_t$ and *window* operators $\boxplus$

- **Plain LARS rules** are of the form
  $r: \ \alpha \leftarrow \beta_1, \ldots, \beta_m, \text{not } \beta_{m+1}, \ldots, \text{not } \beta_n$ where
  - the head α is an atom a or $@_t a$, and
  - β_i are streaming atoms that occur in positive $B^+(r) = \{\beta_1, \ldots, \beta_m\}$ or negative $B^-(r) = \{\beta_{m+1}, \ldots, \beta_n\}$ body
- **Satisfaction**: Given a structure $M = (S_I, W)$ and the evaluation time point $t \in T$, we define that:
  - $M, t \models \alpha$, where $\alpha \in \mathcal{A}^+$, if α holds in S_I at t;
  - $M, t \models B(r)$ if $M, t \models \beta_i$ and $M, t \not\models \beta_j$, where $\beta_i \in B^+(r)$ and $\beta_j \in B^-(r)$;
  - $M, t \models r$ if $M, t \models B(r)$ implies $M, t \models \alpha$;
  - $M, t \models \Pi$ if for every rule r in Π it holds that $M, t \models r$
- Differentiate between a data stream $D_I = (T, \eta_D)$ of extensional atoms $\mathcal{A}^E$ and an interpretation stream $S_{Int} = (T, \eta)$ for D_I where $\eta(a) = \eta_D(a)$, for all $a \in \mathcal{A}^E$.
- **Answer stream** for a data stream $D_I = (T, \eta_D)$ and a plain LARS program Π is an interpretation stream $S_{Int} = (T, \eta)$ such that
  - $S_{Int}, t \models \Pi$, and
  - every substream S'_{Int} of S_{Int} that is an interpretation stream for D_I such that $S'_{Int}, t \models \Pi^{S_{Int}, t} = \{r \in \Pi \mid S_{Int}, t \models B(r)\}$ is equivalent to S_{Int}

Ticker (Beck *et al.*, 2017)

- Engine time specifies duration of a time point
- Different output options: push-based, pull-based, based on model change
- Code (Scala): `https://github.com/hbeck/ticker`

Two reasoning modes

1. Repeated single-shot solving with static encoding (Clingo)
2. Incremental evaluation using Truth Maintenance System (TMS) techniques (Doyle, 1979)
   - Input: model M for program P, rule r
   - Output: model M' for $P \cup \{r\}$ $\Longrightarrow$ for new incremental rules
   - Extension: model M' for $P \setminus \{r\}$ $\Longrightarrow$ for expired rules

   exploit Elkan's [1990] result:
   - the answer sets of normal programs P correspond to the admissible models of TMS $JTMS(P)$
   - excludes constraints/odd loops in TMS

Laser (Bazoobandi *et al.*, 2017b)

- **Plain Fragment**
  - Like Ticker, *Plain LARS* + sliding windows, but aiming at:
  - high performance / throughput
  - $\Rightarrow$ focus on deterministic programs (positive & stratified)
- **Fast model update**
  - efficient substitution management
  - extend semi-naive evaluation (used e.g. for Datalog)
  - incorporate temporal dimension
  - track intervals how long (sub)formulas are guaranteed to hold
  - $\Rightarrow$ avoids redundant re-derivations
  - $\Rightarrow$ efficient removal of expired derivations
- **Performance**
  - Laser outperforms Ticker, C-SPARQL and CQELS in micro-benchmarks
  - source code is available at `https://github.com/karmaresearch/laser`

Distributed Stream Reasoner (Eiter *et al.*, 2019)

- LARS engines Ticker and Laser do monolithic evaluation using a clock (ticks)
- Performance issues under load
- As in stream processing, distribute computation

Distributed Stream Reasoner (Eiter *et al.*, 2019)

- LARS engines Ticker and Laser do monolithic evaluation using a clock (ticks)
- Performance issues under load
- As in stream processing, distribute computation

Distributed LARS (Outline):

- Streaming atoms: $a \mid @_{t'}a \mid \boxplus @_{t'}a \mid \boxplus \Diamond a \mid \boxplus \Box a$

  cast *time-point* to *interval* semantics (support *triggers*)
- Decompose program P using a (stream) *dependency graph*
- A *component graph* over it yields a network of subprograms $P_1, \ldots, P_m$
  - each P_i is run by a stream reasoner
    - publishes streaming atoms to its successors,
    - requests streaming atoms from its predecessors (for itself or successors)
  - a special *master node* interfaces the outside world (publishes all external atoms, wants all internal atoms)
- *stream-stratification* (no cycle through windows) ensures a data pipeline

Component Graph: Network Administration

- LARS Encoding

$$\mathrm{high} \leftarrow \mathrm{value}(V), \boxplus^{k\,\mathsf{sec}} @_T\, \mathrm{alpha}(V), 18 \leq V.$$
$$\mathrm{mid} \leftarrow \mathrm{value}(V), \boxplus^{k\,\mathsf{sec}} @_T\, \mathrm{alpha}(V), 12 \leq V < 18.$$
$$\mathrm{low} \leftarrow \mathrm{value}(V), \boxplus^{k\,\mathsf{sec}} @_T\, \mathrm{alpha}(V), V \leq 12.$$
$$\mathrm{lfu} \leftarrow \boxplus^{k\,\mathsf{sec}} \Box\, \mathrm{high}.$$
$$\mathrm{lru} \leftarrow \boxplus^{k\,\mathsf{sec}} \Box\, \mathrm{mid}.$$
$$\mathrm{fifo} \leftarrow \boxplus^{k\,\mathsf{sec}} \Box\, \mathrm{low}, \boxplus^{[k\,\mathsf{sec}]} \Diamond\, \mathrm{rtm50}.$$
$$\mathrm{done} \leftarrow \mathrm{lfu} \vee \mathrm{lru} \vee \mathrm{fifo}.$$
$$\mathrm{random} \leftarrow \mathrm{not\ done}.$$

Component Graph: Network Administration

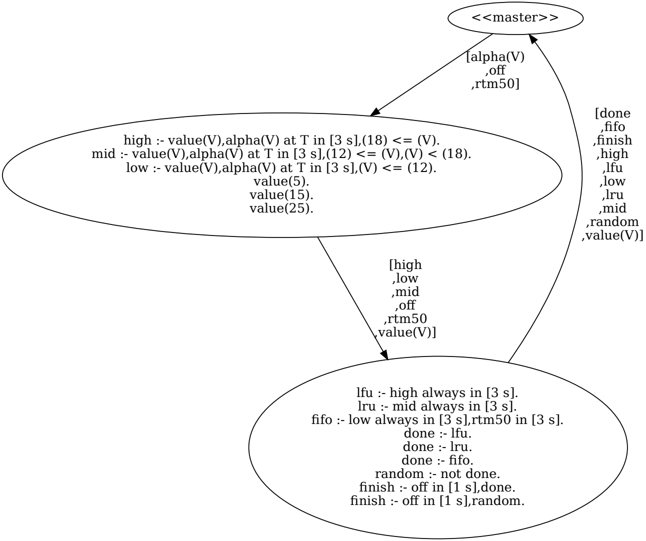
- LARS Encoding

$$\text{high} \leftarrow \text{value}(V), \boxplus^{k\,\text{sec}} @_T\, \text{alpha}(V), 18 \leq V.$$
$$\text{mid} \leftarrow \text{value}(V), \boxplus^{k\,\text{sec}} @_T\, \text{alpha}(V), 12 \leq V < 18.$$
$$\text{low} \leftarrow \text{value}(V), \boxplus^{k\,\text{sec}} @_T\, \text{alpha}(V), V \leq 12.$$
$$\text{lfu} \leftarrow \boxplus^{k\,\text{sec}} \Box\, \text{high}.$$
$$\text{lru} \leftarrow \boxplus^{k\,\text{sec}} \Box\, \text{mid}.$$
$$\text{fifo} \leftarrow \boxplus^{k\,\text{sec}} \Box\, \text{low}, \boxplus^{[k\,\text{sec}]} \Diamond\, \text{rtm50}.$$
$$\text{done} \leftarrow \text{lfu} \vee \text{lru} \vee \text{fifo}.$$
$$\text{random} \leftarrow \text{not done}.$$

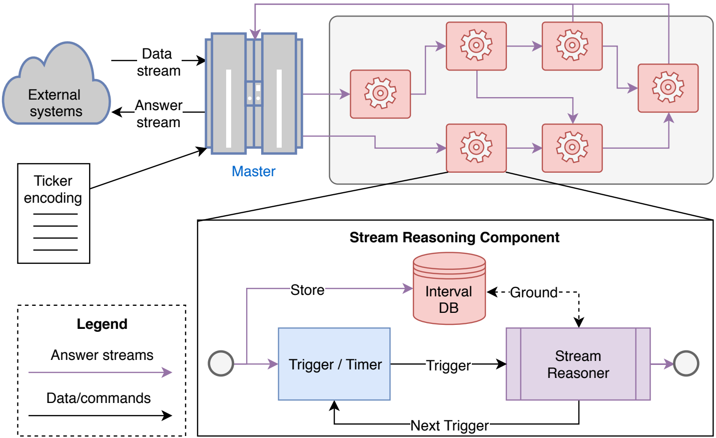
- Ticker encoding (for $k = 3$):

```
high :- value(V), alpha(V) at T [3 sec], 18 <= V.
mid :- value(V), alpha(V) at T [3 sec], 12 <= V, V < 18.
low :- value(V), alpha(V) at T [3 sec], V <= 12.
lfu :- high always [3 sec].
lru :- mid always [3 sec].
fifo :- low always [3 sec], rtm50 [3 sec].
done :- lfu.
done :- lru.
done :- fifo.
random :- not done.
value(5). value(15). value(25).
```

Component Graph: Network Administration, cont’d



Distributed Stream Reasoning System



Master: computes the component graph and spawns nodes in the network

Outline

streaming Multi-Context Systems (Dao-Tran and Eiter, 2017)

- streaming MCS are extending managed MCS by
  - allowing *window atoms* in bridge rules (as in plain LARS rules) $\Rightarrow$ can process input streams, contexts remain abstract

    $$\mathsf{op} \leftarrow \beta_1, \ldots, \beta_j, \mathsf{not}\ \beta_{j+1}, \ldots, \mathsf{not}\ \beta_m$$

    where $\mathsf{op} \in ops_i$ is an operation, $\beta_i = (c_i : \alpha_i)$ is a bridge atom, $c_i \in \{1, \ldots, n\}$ and α_i is a streaming atom for a context C_{c_i}
  - Example consider a simple setup in which an edge router r_1 reports about its caching strategy to a router r_2 located in the middle of the network

Context C_1 (router r_1)
$\mathbf{ACC}_{router}$

$\longrightarrow$

Context C_2 (router r_2)
$\mathbf{ACC}_{router}$
$\mathsf{update}(strategy(r_1, fifo)) \leftarrow (1 : \boxplus^5 \Box fifo)$

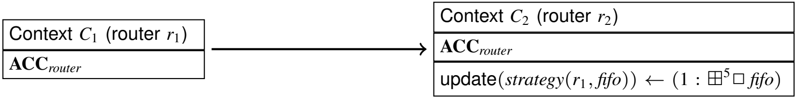
  - the bridge rule updates the local KB of the router r_2 by accessing the output of the context C_1 for the neighbor router r_1 that used *fifo* strategy for the last 5 time units
  - In the example above the set of **acc**eptable belief sets $\mathbf{ACC}_{router}$ is defined by the LARS encoding presented above

streaming Multi-Context Systems (Dao-Tran and Eiter, 2017)

- streaming MCS are extending managed MCS by
  - allowing *window atoms* in bridge rules (as in plain LARS rules) $\Rightarrow$ can process input streams, contexts remain abstract

    $$\text{op} \leftarrow \beta_1, \ldots, \beta_j, \text{not } \beta_{j+1}, \ldots, \text{not } \beta_m$$

    where op $\in ops_i$ is an operation, $\beta_i = (c_i : \alpha_i)$ is a bridge atom, $c_i \in \{1, \ldots, n\}$ and α_i is a streaming atom for a context C_{c_i}
  - Example consider a simple setup in which an edge router r_1 reports about its caching strategy to a router r_2 located in the middle of the network



  - the bridge rule updates the local KB of the router r_2 by accessing the output of the context C_1 for the neighbor router r_1 that used *fifo* strategy for the last 5 time units
  - enabling extensions with *streaming contexts* as in the Distributed Stream Reasoner
  - In the example above the set of **acc**eptable belief sets $\mathbf{ACC}_{router}$ is defined by the LARS encoding presented above

streaming Multi-Context Systems, cont'd

- Further extensions include:
  - sMCS can model computation time f_i for a context C_i and data transfer time Δ_{ik} between contexts C_i and C_k $\Rightarrow$ timestamps of atoms in the stream are corrected
  - *asynchronous execution* as well as *ignore* and *restart* policies for incoming data
  - supports both *pushing* and *pulling* execution modes
- sMCS describe different modes in which a context C_i might occur using its *state* $\mathbf{s}_i = (s_i, o_i, kb_i)$, where
  - $s_i \subseteq \{IE, SE\}$ is the execution status IE – *Intended Execution* with pull or push – and SE – *Start Execution* like idle or busy.
  - o_i is an *output* belief set or ϵ if no output is streamed to other contexts
  - kb_i is a local KB that can be changed
- A *run* of an sMCS is a sequence $\mathbf{s} = \mathbf{s}(0), \ldots, \mathbf{s}(t)$ of global states $\mathbf{s}(i) = (\mathbf{s}_1(i), \ldots, \mathbf{s}_n(i))$ where each $\mathbf{s}_j(i)$ is a state of C_i
- Semantics sMCS can simulate rMCS using the notion of an *idelized run*:
  - Idea: model an idealized system that can compute equilibria between two consecutive time points $\Rightarrow$ $\mathbf{ACC}$ can be computed finitely often between these points
  - Let the transferring time be 0 and δ be an *infinitesimally small chronon* denoting computation time, i.e., $t < t + \delta$ and $t + \delta = t + k\delta$ for any $k \in \mathbb{N} - \{0\}$
  - Then, for every state s

$$o_i(t+\delta) = \mathbf{ACC}(kb_i(t+\delta)) \text{ and } kb_i(t+1) = kb_i(t+\delta)$$

sMCS: Feedback Equilibria

- Idealized settings are very rare in practice and completely asynchronous computations can be uncomfortable, e.g., mutually dependent contexts
- Idea of *Feedback Equilibria*:
  - focus on *Strongly Connected Components* (SCC) of a context dependency graph, i.e., $C_i \to C_j$ if a bridge atom $(j : A)$ occurs in br_i
  - ignore/delay data appearing in the stream from outside of the system
  - while computing, any context C can request stability of its SCC $\mathcal{C}$ at a time t_e with a timeout time t_o
    - the contexts in $\mathcal{C}$ are restarted with the state of a stream at t_e
    - at t_o the SCC $\mathcal{C}$ either reports an equilibrium or restarts its contexts with input collected from the stream at t_o
- Given a run $\mathbf{s}$ at a time t_e, the *Feedback Equilibrium* of $\mathcal{C}$ is defined as follows:

$$\forall\ C_i \in \mathcal{C} \text{ belief set } BS_i \in \mathbf{ACC}_i(mng_i(app_i^\delta(\mathbf{s}, t_e), kb_i(t_e)))$$

where

- mng is a management function and
- app_i^δ is a set of all applicable bridge rules that considers all data in the input stream till t_e as well as data streamed within $\mathcal{C}$ during δ – cyclic information flow is respected

Outline

Summary

- Stream Reasoning is a topic of growing interest
- A variety of multi-context systems that can be used to in stream reasoning scenarios exist:
  - reactive Multi-Context Systems Brewka *et al.* (2018)
  - asynchronous Multi-Context Systems (Ellmauthaler and Pührer, 2015; Ellmauthaler, 2018)
  - streaming Multi-Context Systems (Dao-Tran and Eiter, 2017)
  - timed MCS (Cabalar *et al.*, 2019)
- Stream Reasoning is related to temporal reasoning, but distinctive features
  - windows
  - incrementality
  - push vs pull
  - ...
- Important uses cases and applications
  - monitoring & control
  - prediction
  - diagnosis/configuration

Outline

Open Issues

- Model management
  - incremental evaluation, yield similar models
  - View maintenance for stratified Datalog, e.g., (Motik *et al.*, 2019)
- Algorithms suitable for distributed fail-safe stream reasoning
- Computation of justifications and debugging of stream reasoning systems
- Deal with quantitative versions of semantics
  - deal with noise, uncertainty
  - probabilistic semantics, e.g., (Nickles and Mileo, 2014)
  - optimization
- Integration with other languages and formalisms (automata, regular expressions)
- More powerful windows (e.g. aggregates, returning results)
- Integration in more complex environments (CPS, MCS)
- Benchmarking and applications

Resources

- General website providing information about Stream Reasoning resources, events, competitions, etc.:
  `http://streamreasoning.org/`
- RDF Stream Reasoning Community Group:
  `https://www.w3.org/community/rsp/`
- An overview of Stream Processing software:
  `https://bit.ly/2m3RCnG`
- Selected software:
  - ETALIS Complex Event Processing system:
    `https://github.com/sspider/etalis`
  - Example of a multi-shot Stream Reasoner based on clasp
    `https://github.com/potassco/aspStream`
  - Ticker LARS reasoner:
    `https://github.com/hbeck/ticker`
  - Laser LARS reasoner:
    `https://github.com/karmaresearch/Laser`
  - Distributed LARS reasoner:
    `https://git-ainf.aau.at/Paul.Ogris/distributed-sr`
  - Experiments Codes ans Data for StreamRule Parallization:
    `https://github.com/ThuLePham/SR_Experiments`

References I

Daniel J. Abadi, Donald Carney, Ugur Cetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Michael Stonebraker, Nesime Tatbul, and Stanley B. Zdonik. Aurora: a new model and architecture for data stream management. *VLDB J.*, 12(2):120–139, 2003.

James F. Allen. Maintaining knowledge about temporal intervals. *Commun. ACM*, 26(11):832–843, November 1983.

Peter Alvaro, William R. Marczak, Neil Conway, Joseph M. Hellerstein, David Maier, and Russell Sears. Dedalus: Datalog in time and space. In *Datalog*, volume 6702 of *Lecture Notes in Computer Science*, pages 262–281. Springer, 2010.

Darko Anicic, Sebastian Rudolph, Paul Fodor, and Nenad Stojanovic. Stream reasoning and complex event processing in ETALIS. *Semantic Web Journal*, 2012.

Arvind Arasu, Brian Babcock, Shivnath Babu, Mayur Datar, Keith Ito, Rajeev Motwani, Itaru Nishizawa, Srivastava Utkarsh, Dilys Thomas, Rohit Varma, and Jennifer Widom. STREAM: the stanford stream data manager. *IEEE Data Eng. Bull.*, 26(1):19–26, 2003.

Arvind Arasu, Shivnath Babu, and Jennifer Widom. The CQL continuous query language: semantic foundations and query execution. *VLDB J.*, 15(2):121–142, 2006.

Alexander Artikis, Marek Sergot, and Georgios Paliouras. An event calculus for event recognition. *IEEE Trans. Knowl. Data Eng.*, 2014.

Davide Francesco Barbieri, Daniele Braga, Stefano Ceri, Emanuele Della Valle, and Michael Grossniklaus. C-SPARQL: a continuous query language for rdf data streams. *Int. J. Semantic Computing*, 4(1):3–25, 2010.

Roger S. Barga, Jonathan Goldstein, Mohamed H. Ali, and Mingsheng Hong. Consistent streaming through time: A vision for event stream processing. In *CIDR*, pages 363–374. www.cidrdb.org, 2007.

Marianne Baudinet, Jan Chomicki, and Pierre Wolper. Temporal deductive databases. In *Temporal Databases*, pages 294–320. Benjamin/Cummings, 1993.

Hamid R. Bazoobandi, Harald Beck, and Jacopo Urbani. Expressive stream reasoning with laser. In *The Semantic Web - ISWC 2017 - 16th International Semantic Web Conference, Vienna, Austria, October 21-25, 2017, Proceedings, Part I*, pages 87–103, 2017.

Hamid R. Bazoobandi, Harald Beck, and Jacopo Urbani. Expressive stream reasoning with laser. In Claudia d'Amato, Miriam Fernández, Valentina A. M. Tamma, Freddy Lécué, Philippe Cudré-Mauroux, Juan F. Sequeda, Christoph Lange, and Jeff Heflin, editors, *The Semantic Web - ISWC 2017 - 16th International Semantic Web Conference, Vienna, Austria, October 21-25, 2017, Proceedings, Part I*, volume 10587 of *Lecture Notes in Computer Science*, pages 87–103. Springer, 2017.

References II

Harald Beck, Thomas Eiter, and Christian Folie. Ticker: A system for incremental asp-based stream reasoning. *TPLP*, 17(5-6):744–763, 2017.

William P. Bejeck and Neha Narkhede. *Kafka Streams in Action*. Manning Publications, 2018.

S. Brandt, E. Güzel Kalayci, R. Kontchakov, V. Ryzhikov, G. Xiao, and M. Zakharyaschev. Ontology-based data access with a horn fragment of metric temporal logic. In *Proceedings 31st Conference on Artificial Intelligence (AAAI '17)*. AAAI Press, 2017.

Lars Brenna, Alan J. Demers, Johannes Gehrke, Mingsheng Hong, Joel Ossher, Biswanath Panda, Mirek Riedewald, Mohit Thatte, and Walker M. White. Cayuga: a high-performance event processing engine. In *SIGMOD Conference*, pages 1100–1102. ACM, 2007.

Gerhard Brewka and Thomas Eiter. Equilibria in heterogeneous nonmonotonic multi-context systems. In *AAAI*, pages 385–390. AAAI Press, 2007.

Gerd Brewka, Thomas Eiter, and Miroslaw Truszczyński. Answer set programming at a glance. *Communications of the ACM*, 54(12):92–103, 2011. doi>10.1145/2043174.2043195.

Gerhard Brewka, Thomas Eiter, Michael Fink, and Antonius Weinzierl. Managed multi-context systems. In *IJCAI*, pages 786–791. IJCAI/AAAI, 2011.

Gerhard Brewka, Stefan Ellmauthaler, and Jörg Pührer. Multi-context systems for reactive reasoning in dynamic environments. In *ECAI*, volume 263 of *FAIA*, pages 159–164. IOS Press, 2014.

Gerd Brewka, Thomas Eiter, and Miroslaw Truszczyński, editors. *AI Magazine: special issue on Answer Set Programming*. AAAI Press, 2016. Volume 37, number 3. Editorial pp. 5-6.

Gerhard Brewka, Stefan Ellmauthaler, Ricardo Gonçalves, Matthias Knorr, João Leite, and Jörg Pührer. Reactive multi-context systems: Heterogeneous reasoning in dynamic environments. *Artificial Intelligence*, 256:68–104, 2018.

Pedro Cabalar, Stefania Costantini, Giovanni De Gasperis, and Andrea Formisano. Multi-context systems in dynamic environments. *Ann. Math. Artif. Intell.*, 86(1-3):87–120, 2019.

Jean-Paul Calbimonte, José Mora, and Óscar Corcho. Query rewriting in RDF stream processing. In *Proc. of ESWC 2016*, pages 486–502, 2016.

References III

Don Carney, Uğur Çetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Greg Seidman, Michael Stonebraker, Nesime Tatbul, and Stan Zdonik. Monitoring streams: a new class of data management applications. In *Proc. of VLDB 2002*, pages 215–226, 2002.

Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In Dexter Kozen, editor, *Logics of Programs, Workshop, Yorktown Heights, New York, May 1981*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer, 1981.

Corinna Cortes, Kathleen Fisher, Daryl Pregibon, Anne Rogers, and Frederick Smith. Hancock: A language for analyzing transactional data streams. In *Data Stream Management - Processing High-Speed Data Streams*, pages 387–408. 2016.

Gianpaolo Cugola and Alessandro Margara. TESLA: a formally defined event specification language. In *DEBS*, pages 50–61. ACM, 2010.

Gianpaolo Cugola and Alessandro Margara. Complex event processing with T-REX. *Journal of Systems and Software*, 85(8):1709–1728, 2012.

Minh Dao-Tran and Thomas Eiter. Streaming multi-context systems. In *IJCAI*, pages 1000–1007. ijcai.org, 2017.

U. Dayal, E. Hanson, and J. Widom. Active database systems. In W. Kim, editor, *Modern Database Systems*, pages 434–456. Addison Wesley, 1995.

Emanuele Della Valle, Stefano Ceri, Davide Francesco Barbieri, Daniele Braga, and Alessandro Campi. A first step towards stream reasoning. In *FIS*, pages 72–81, 2008.

Emanuele Della Valle, Stefano Ceri, Frank van Harmelen, and Dieter Fensel. It's a streaming world! reasoning upon rapidly changing information. *IEEE Intelligent Systems*, 24:83–89, 2009.

Patrick Doherty, Jonas Kvarnström, and Fredrik Heintz. A temporal logic-based planning and execution monitoring framework for unmanned aircraft systems. *Autonomous Agents and Multi-Agent Systems*, 19(3):332–377, 2009.

Jon Doyle. A truth maintenance system. *Artif. Intell.*, 12(3):231–272, 1979.

Thomas Eiter, Stefano Germano, Giovambattista Ianni, Tobias Kaminski, Christoph Redl, Peter Schüller, and Antonius Weinzierl. The DLVHEX system. *KI*, 32(2-3):187–189, 2018.

Thomas Eiter, Paul Ogris, and Konstantin Schekotihin. A distributed approach to LARS stream reasoning (system paper). *TPLP*, 19(5-6):974–989, 2019.

References IV

Charles Elkan. A rational reconstruction of nonmonotonic truth maintenance systems. *Artif. Intell.*, 43(2):219–234, 1990.

Stefan Ellmauthaler and Jörg Pührer. Asynchronous multi-context systems. In *Advances in Knowledge Representation, Logic Programming, and Abstract Argumentation - Essays Dedicated to Gerhard Brewka on the Occasion of His 60th Birthday*, volume 9060 of *LNCS*. Springer, 2015.

Stefan Ellmauthaler. *Multi-Context Reasoning in Continuous Data-Flow Environments*. PhD thesis, Leipzig University, 2018.

ETSI. *TR 101 607; Intelligent Transport Systems (ITS); Cooperative ITS (C-ITS); Release 1.* 2013.

Wai Fu Fung, David Sun, and Johannes Gehrke. COUGAR: the network is the database. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, Madison, Wisconsin, USA, June 3-6, 2002*, page 621, 2002.

Chiara Ghidini and Fausto Giunchiglia. Local models semantics, or contextual reasoning=locality+compatibility. *Artificial Intelligence*, 127(2):221–259, 2001.

Ricardo Gonçalves, Matthias Knorr, and João Leite. Evolving multi-context systems. In *ECAI14*, volume 263 of *FAIA*, pages 375–380, 2014.

Georg Gottlob, Erich Grädel, and Helmut Veith. Datalog LITE: a deductive query language with linear time model checking. *ACM Trans. Comput. Log.*, 3(1):42–79, 2002.

Fredrik Heintz, Jonas Kvarnström, and Patrick Doherty. Bridging the sense-reasoning gap: Dyknow - stream-based middleware for knowledge processing. *Advanced Engineering Informatics*, 24(1):14–26, 2010.

Tomas Hrycej. A temporal extension of prolog. *J. Log. Program.*, 15(1&2):113–145, 1993.

Fabian Hueske and Vasiliki Kalavri. *Stream processing with Apache Flink*. OReilly Media, 2019.

Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.

Georg Lausen, Bertram Ludäscher, and Wolfgang May. On active deductive databases: The statelog approach. In Burkhard Freitag, Hendrik Decker, Michael Kifer, and Andrei Voronkov, editors, *Transactions and Change in Logic Databases, International Seminar on Logic Databases and the Meaning of Change, Schloss Dagstuhl, Germany, September 23-27, 1996 and ILPS '97 Post-Conference Workshop on (Trans)Actions and Change in Logic Programming and Deductive Databases, (DYNAMICS'97) Port Jefferson, NY, USA, October 17, 1997, Invited Surveys and Selected Papers*, volume 1472 of *Lecture Notes in Computer Science*, pages 69–106. Springer, 1998.

References V

David C. Luckham. Rapide: A language and toolset for simulation of distributed systems by partial orderings of events. In *Partial Order Methods in Verification*, volume 29 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 329–357. DIMACS/AMS, 1996.

David C. Luckham. *The power of events - an introduction to complex event processing in distributed enterprise systems*. ACM, 2005.

Samuel Madden, Mehul Shah, Joseph M. Hellerstein, and Vijayshankar Raman. Continuously adaptive continuous queries over streams. In *2002 ACM SIGMOD International Conference on Management of Data*, pages 49–60, 2002.

Alessandra Mileo, Ahmed Abdelrahman, Sean Policarpio, and Manfred Hauswirth. Streamrule: A nonmonotonic stream reasoning system for the semantic web. In Wolfgang Faber and Domenico Lembo, editors, *Web Reasoning and Rule Systems - 7th International Conference, RR 2013, Mannheim, Germany, July 27-29, 2013. Proceedings*, volume 7994 of *Lecture Notes in Computer Science*, pages 247–252. Springer, 2013.

Boris Motik, Yavor Nenov, Robert Piro, and Ian Horrocks. Maintenance of datalog materialisations revisited. *Artif. Intell.*, 269:76–136, 2019.

Matthias Nickles and Alessandra Mileo. Web stream reasoning using probabilistic answer set programming. In *RR*, volume 8741 of *Lecture Notes in Computer Science*, pages 197–205. Springer, 2014.

Philipp Obermeier, Javier Romero, and Torsten Schaub. Multi-shot stream reasoning in answer set programming: A preliminary report. *OJDB*, 6(1):33–38, 2019.

Mehmet A. Orgun and William W. Wadge. A relational algebra as a query language for temporal DATALOG. In *DEXA*, pages 276–281. Springer-Verlag, Wien, 1992.

Balaji Padmanabhan and Alexander Tuzhilin. Pattern discovery in temporal databases: A temporal logic approach. In *KDD*, pages 351–354. AAAI Press, 1996.

Thu-Le Pham, Muhammad Intizar Ali, and Alessandra Mileo. C-asp: Continuous asp-based reasoning over rdf streams. In Marcello Balduccini, Yuliya Lierler, and Stefan Woltran, editors, *Logic Programming and Nonmonotonic Reasoning*, pages 45–50, Cham, 2019. Springer International Publishing.

Thu-Le Pham, Muhammad Intizar Ali, and Alessandra Mileo. Enhancing the scalability of expressive stream reasoning via input-driven parallelization. *Semantic Web*, 10(3):457–474, 2019.

References VI

Danh Le Phuoc, Minh Dao-Tran, Josiane Xavier Parreira, and Manfred Hauswirth. A native and adaptive approach for unified processing of linked streams and linked data. In *ISWC (1)*, pages 370–388, 2011.

Danh Le Phuoc, Josiane Xavier Parreira, and Manfred Hauswirth. Linked stream data processing. In *Reasoning Web*, 2012.

Amir Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57. IEEE Computer Society, 1977.

Han Reichgelt. Semantics for reified temporal logic. In *On Advances in Artificial Intelligence*, pages 49–61, New York, NY, USA, 1987. John Wiley & Sons, Inc.

Alessandro Ronca, Mark Kaminski, Bernardo Cuenca Grau, Boris Motik, and Ian Horrocks. Stream reasoning in temporal datalog. In Sheila A. McIlraith and Kilian Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 1941–1948. AAAI Press, 2018.

Torsten Schaub and Stefan Woltran. Special issue on answer set programming. *KI*, 32(2-3):101–103, 2018.

Peter Schüller. *Inconsistency in multi-context systems: Analysis and efficient evaluation*. PhD thesis, Vienna University of Technology, 2012.

Paul Tarau, Jan Wielemaker, and Tom Schrijvers. Lazy stream programming in prolog. *CoRR*, abs/1907.11354, 2019. TPLP, special issue on ICLP 2019, to appear.

Douglas B. Terry, David Goldberg, David A. Nichols, and Brian M. Oki. Continuous queries over append-only databases. In *SIGMOD Conference*, pages 321–330. ACM Press, 1992.

Antonius Weinzierl. *Inconsistency management under preferences for multi-context systems and extensions*. PhD thesis, Ph. D. thesis, Vienna University of Technology, 2014.

J. Widom and S. Ceri, editors. *Active Database Systems: Triggers and Rules For Advanced Database Processing*. Morgan Kaufmann, 1996.

Eugene Wu, Yanlei Diao, and Shariq Rizvi. High-performance complex event processing over streams. In *SIGMOD Conference*, pages 407–418. ACM, 2006.

Carlo Zaniolo. Logical foundations of continuous query languages for data streams. In *Datalog*, volume 7494 of *Lecture Notes in Computer Science*, pages 177–189. Springer, 2012.