Stream Packing for Asynchronous Multi-Context Systems

Jörg Pührer

Intelligent Systems Group, Department of Computer Science
Leipzig University


Joint work with Stefan Ellmauthaler
Multi-Context Systems for Stream Reasoning

Variants of heterogeneous nonmonotonic Multi-Context Systems (MCS) [Brewka and Eiter, 2007] for SR:

- Reactive MCS [Brewka, Ellmauthaler, and P_, 2014]
- Evolving MCS [Gonçalves, Knorr, and Leite, 2014]
- Asynchronous MCS [Ellmauthaler, and P_, 2014]
Multi-Context Systems for Stream Reasoning

Variants of heterogeneous nonmonotonic Multi-Context Systems (MCS) [Brewka and Eiter, 2007] for SR:

- Reactive MCS [Brewka, Ellmauthaler, and P__, 2014]
- Evolving MCS [Gonçalves, Knorr, and Leite, 2014]
- Asynchronous MCS [Ellmauthaler, and P__, 2014]
Multi-Context Systems for Stream Reasoning

Variants of heterogeneous nonmonotonic Multi-Context Systems (MCS) [Brewka and Eiter, 2007] for SR:

- Reactive MCS [Brewka, Ellmauthaler, and P_, 2014]

- Evolving MCS [Gonçalves, Knorr, and Leite, 2014]

- Asynchronous MCS [Ellmauthaler, and P_, 2014]

- common goals: integrate KR formalisms, online semantics, access to sensor and stream data
(Offline) Semantics of rMCS
(Offline) Semantics of rMCS

Belief State

Belief Set 2
Belief Set 1
Belief Set 2

Bridge Rules

$C_1$

$ACC_1$

$kb_1$

$C_2$

$ACC_2$

$kb_2$

$C_3$

$ACC_3$

$kb_3$

Bridge Rules
(Offline) Semantics of rMCS

Belief Set 1
Belief Set 2
Belief State

$kb_1$
$ACC_1$
$C_1$
Bridge Rules

$kb_2$
$ACC_2$
$C_2$
Bridge Rules

$kb_3$
$ACC_3$
$C_3$
Bridge Rules
(Offline) Semantics of rMCS
(Offline) Semantics of rMCS

Belief Set 2
Belief Set 1
Belief Set 2

Belief State

$ACC_1$

$kb'_1$

$C_1$
Bridge Rules

$ACC_2$

$kb'_2$

$C_2$
Bridge Rules

$ACC_3$

$kb'_3$

$C_3$
Bridge Rules
(Offline) Semantics of rMCS
(Offline) Semantics of rMCS
Online Semantics of Multi-Context Systems for SR

Online semantics Reactive and Evolving Multi-Context Systems in terms of *streams of equilibria*:
Online Semantics of Multi-Context Systems for SR

Online semantics Reactive and Evolving Multi-Context Systems in terms of *streams of equilibria*:

1. compute an equilibrium given current stream input

![Diagram of contexts and time]
Online Semantics of Multi-Context Systems for SR

Online semantics Reactive and Evolving Multi-Context Systems in terms of *streams of equilibria*:

1. compute an equilibrium given current stream input
Online Semantics of Multi-Context Systems for SR

Online semantics Reactive and Evolving Multi-Context Systems in terms of *streams of equilibria*:

1. compute an equilibrium given current stream input
2. update context knowledge bases based on equilibrium
Online Semantics of Multi-Context Systems for SR

Online semantics Reactive and Evolving Multi-Context Systems in terms of *streams of equilibria*:

1. compute an equilibrium given current stream input
2. update context knowledge bases based on equilibrium
3. compute next equilibrium
Online Semantics of Multi-Context Systems for SR

Online semantics Reactive and Evolving Multi-Context Systems in terms of *streams of equilibria*:

1. compute an equilibrium given current stream input
2. update context knowledge bases based on equilibrium
3. compute next equilibrium

![Diagram of multi-context systems](image)
Online Semantics of Multi-Context Systems for SR

Online semantics Reactive and Evolving Multi-Context Systems in terms of *streams of equilibria*:

1. compute an equilibrium given current stream input
2. update context knowledge bases based on equilibrium
3. compute next equilibrium

→ all contexts must run synchronously
Asynchronous Multi-Context Systems

Asynchronous MCS [Ellmauthaler, and P__, 2014] follow a looser coupling of contexts:

- each context may run at its own pace (asynchronously)
Asynchronous Multi-Context Systems

Asynchronous MCS [Ellmauthaler, and P_, 2014] follow a looser coupling of contexts:

- each context may run at its own pace (asynchronously)
- computation of belief sets may take time
Asynchronous Multi-Context Systems

Asynchronous MCS [Ellmauthaler, and P_, 2014] follow a looser coupling of contexts:

▶ each context may run at its own pace (asynchronously)
▶ computation of belief sets may take time
▶ contexts compute their belief sets independently
Asynchronous Multi-Context Systems

Asynchronous MCS [Ellmauthaler, and P__, 2014] follow a looser coupling of contexts:

- each context may run at its own pace (asynchronously)
- computation of belief sets may take time
- contexts compute their belief sets independently

⇒ semantics *not based* on equilibria
Asynchronous Multi-Context Systems

Asynchronous MCS [Ellmauthaler, and P_, 2014] follow a looser coupling of contexts:

- each context may run at its own pace (asynchronously)
- computation of belief sets may take time
- contexts compute their belief sets independently

⇒ semantics *not based* on equilibria

Consequence:

- no defined basis for activating bridge rules
Asynchronous Multi-Context Systems

Asynchronous MCS [Ellmauthaler, and P_, 2014] follow a looser coupling of contexts:

- each context may run at its own pace (asynchronously)
- computation of belief sets may take time
- contexts compute their belief sets independently

\[\text{semantics \textit{not based} on equilibria}\]

Consequence:

- no defined basis for activating bridge rules
- use output rules instead of bridge rules
Communication in Asynchronous Multi-Context Systems

An output rule $r$ for context $C$ is of the form

$$\langle n', i \rangle \leftarrow b_1, \ldots, b_j, \text{not } b_{j+1}, \ldots, \text{not } b_m$$

- $n'$ is the name of a context or an output stream (called the stakeholder of $r$)
- $i$ is a piece of information
- every $b_\ell$ is a belief for $C$
Communication in Asynchronous Multi-Context Systems

An output rule $r$ for context $C$ is of the form

$$\langle n', i \rangle \leftarrow b_1, \ldots, b_j, \text{not } b_{j+1}, \ldots, \text{not } b_m$$

- $n'$ is the name of a context or an output stream (called the stakeholder of $r$)
- $i$ is a piece of information
- every $b_\ell$ is a belief for $C$

In contrast to bridge rules
- activation depends on a belief set of $C$ instead of a belief state
Communication in aMCSs (ctd.)

Each context has an *input buffer* representing its currently available input:

**Definition**

A *data set* is a pair \( d = \langle n, I \rangle \)

- \( n \) is either a context or a sensor name stating the *source* of \( d \)
- \( I \) is a set of pieces of information

An *input buffer* is a sequence of data sets.
Communication in aMCSs (ctd.)

Each context has an *input buffer* representing its currently available input:

**Definition**

A *data set* is a pair \( d = \langle n, I \rangle \)

- \( n \) is either a context or a sensor name stating the *source* of \( d \)
- \( I \) is a set of pieces of information

An *input buffer* is a sequence of data sets.

When a context computes a belief set the information derived in the output rules is sent to the stackholders:

**Definition**

Let \( C \) be a context named \( n \) with output rules \( OR \). The *output* of \( C \) under a belief set \( S \) relevant for \( n' \) is the data set

\[
\langle n, \{ i \mid r \in OR, hd(r) = \langle n', i \rangle, S \models bd(r) \} \rangle
\]
Structure of Asynchronous Multi-Context Systems

Further components of a context:

- **Computation Controller (cc):** relation that decides when context starts to compute
- **Context Update Function (cu):** function that modifies the whole context before computing
Structure of Asynchronous Multi-Context Systems

Important role of abstract components cc and cu:
- decide when and what parts of the incoming data in the input buffer should be processed
- prepare data in the input buffer for use in the context logic
Structure of Asynchronous Multi-Context Systems

Important role of abstract components cc and cu:

- decide when and what parts of the incoming data in the input buffer should be processed
- prepare data in the input buffer for use in the context logic

**Goal:** a language realising core aspects of cc and cu:
Structure of Asynchronous Multi-Context Systems

Important role of abstract components cc and cu:

- decide when and what parts of the incoming data in the input buffer should be processed
- prepare data in the input buffer for use in the context logic

**Goal:** a language realising core aspects of cc and cu:

- rearrange and filter buffered data (from different input streams)
- decide whether there is sufficient and right information available to start processing
Structure of Asynchronous Multi-Context Systems

Important role of abstract components cc and cu:

- decide when and what parts of the incoming data in the input buffer should be processed
- prepare data in the input buffer for use in the context logic

Goal: a language realising core aspects of cc and cu:

- rearrange and filter buffered data (from different input streams)
- decide whether there is sufficient and right information available to start processing

use Answer-Set Programming (ASP)
Idea

The aMCS engine provides information about the data sets on a context’s input stream as facts. They are the input of an answer-set program that decides

- if the data in the buffer is sufficient for processing
- which data sets should form a package that will be processed
- which data sets should be deleted from or remain in the buffer
Example aMCS - Emergency Team Management

- task planner context needs to assign ambulances to emergency cases
- desired input: one new case, all available ambulances
Representation of a possible input buffer by facts:

ds_avail(ca_ds11). ds_avail(ca_ds12).
ds_avail(am_ds54). ds_avail(am_ds55). ds_avail(am_ds56).
source(ca_ds11,ctxt_case_anl). source(ca_ds12,ctxt_case_anl).
source(am_ds54,ctxt_amb_mng). source(am_ds55,ctxt_amb_mng).
source(am_ds56,ctxt_amb_mng).

ASP language for package specification:

<table>
<thead>
<tr>
<th>Atom</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ds_avail(ds)</td>
<td>Input</td>
<td>Data set ds is available on the input buffer</td>
</tr>
<tr>
<td>source(ds,ctxt)</td>
<td>Input</td>
<td>Data set ds originates from context ctxt</td>
</tr>
</tbody>
</table>
Answer-Set Program for Packing

1  aux_case_avail :- ds_avail(DS), source(DS, ctxt_case_anl).
2  aux_ambulance_avail :- ds_avail(DS), source(DS, ctxt_amb_mng).
3  process_as_schema(sch1) :- aux_case_avail, aux_ambulance_avail.
4  in_pack(DS) :- ds_avail(DS), source(DS, ctxt_amb_mng).
5  1 { aux_case_in_pack(DS) : ds_avail(DS), source(DS, ctxt_case_anl) } 1.
6  in_pack(DS) :- aux_case_in_pack(DS).
7  rm_pack.

<table>
<thead>
<tr>
<th>Atom</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in_pack(ds)</td>
<td>Directive</td>
<td>Data set ds is considered part of the package</td>
</tr>
<tr>
<td>process_as_schema(sch)</td>
<td>Directive</td>
<td>The data sets defined by in_pack/1 atoms form a package of schema sch and are passed on for processing</td>
</tr>
<tr>
<td>rm_pack</td>
<td>Directive</td>
<td>Remove all data sets of processed packages from the input buffer</td>
</tr>
</tbody>
</table>
Tagging

For flexible control which data sets form a package
- use a *tagging system* for data sets (and computations)
- data sets associated with clingo terms that serve as tags

Tags can be assigned by
- the sending context: E.g., tagging optimal solutions
Tagging

For flexible control which data sets form a package

▶ use a *tagging system* for data sets (and computations)
▶ data sets associated with *clingo* terms that serve as tags

Tags can be assigned by

▶ the sending context: E.g., tagging optimal solutions
▶ the receiving context: E.g., different contexts receive same data sets but require different tagging
Tagging

For flexible control which data sets form a package
- use a *tagging system* for data sets (and computations)
- data sets associated with clingo terms that serve as tags

Tags can be assigned by
- the sending context: E.g., tagging optimal solutions
- the receiving context: E.g., different contexts receive same data sets but require different tagging
- the aMCS engine: E.g., assigning time stamps
Tagging

For flexible control which data sets form a package
- use a tagging system for data sets (and computations)
- data sets associated with clingo terms that serve as tags

Tags can be assigned by
- the sending context: E.g., tagging optimal solutions
- the receiving context: E.g., different contexts receive same data sets but require different tagging
- the aMCS engine: E.g., assigning time stamps
- the answer-set program itself to make packing stateful:
  - two directives add_tag and rm_tag for this purpose
  - applications: E.g., avoid starvation of data sets, marking computations as trusted or untrusted
Conclusion

New method for packing incoming data before processing

- use in asynchronous multi-context systems
- based on answer-set programming
Conclusion

New method for packing incoming data before processing

▶ use in asynchronous multi-context systems
▶ based on answer-set programming

Why ASP?

▶ rich declarative representation language
▶ elaboration tolerance
▶ very efficient ASP fragments will often suffice for our purposes
▶ exploit expressive power of ASP when needed
▶ exploit ASP optimisation frameworks
Thank you!
<table>
<thead>
<tr>
<th>Atom</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ds_avail(ds)</code></td>
<td>Input</td>
<td>Data set <code>ds</code> is available on the input buffer</td>
</tr>
<tr>
<td><code>ds_comp(ds,comp)</code></td>
<td>Input</td>
<td>Data set <code>ds</code> belongs to computation <code>comp</code></td>
</tr>
<tr>
<td><code>source(comp,ctxt)</code></td>
<td>Input</td>
<td>Computation <code>comp</code> is a computation of context <code>ctxt</code></td>
</tr>
<tr>
<td><code>source(ds,ctxt)</code></td>
<td>Input</td>
<td>Data set <code>ds</code> originates from context <code>ctxt</code></td>
</tr>
<tr>
<td><code>source(ds,therm)</code></td>
<td>Input</td>
<td>Data set <code>ds</code> originates from sensor <code>therm</code></td>
</tr>
<tr>
<td><code>eoc(comp)</code></td>
<td>Input</td>
<td>Computations <code>comp</code> has ended.</td>
</tr>
<tr>
<td><code>tag(comp,solves(probl1))</code></td>
<td>Input</td>
<td>Computation <code>comp</code> is tagged with function <code>solves(probl1)</code></td>
</tr>
<tr>
<td><code>tag(ds,&quot;optimum&quot;)</code></td>
<td>Input</td>
<td>Data set <code>ds</code> is tagged with string &quot;optimum&quot;</td>
</tr>
<tr>
<td><code>time(1000)</code></td>
<td>Input</td>
<td>An external clock provides <code>1000</code> as current time</td>
</tr>
<tr>
<td><code>ignore(comp)</code></td>
<td>Directive</td>
<td>Ignore future data sets of computation <code>comp</code></td>
</tr>
<tr>
<td><code>add_tag(comp,best(3))</code></td>
<td>Directive</td>
<td>Tag computation <code>comp</code> with function <code>best(3)</code></td>
</tr>
<tr>
<td><code>rm_tag(comp,&quot;trusted&quot;)</code></td>
<td>Directive</td>
<td>Remove the tag &quot;trusted&quot; from computation <code>comp</code></td>
</tr>
<tr>
<td><code>rm(comp)</code></td>
<td>Directive</td>
<td>Remove all data sets of computation <code>comp</code> from the input buffer</td>
</tr>
<tr>
<td><code>rm(ds)</code></td>
<td>Directive</td>
<td>Remove data set <code>ds</code> from the input buffer</td>
</tr>
<tr>
<td><code>rm_pack</code></td>
<td>Directive</td>
<td>Remove all data sets of processed packages from the input buffer</td>
</tr>
</tbody>
</table>

**Variant: one package per answer set**

<table>
<thead>
<tr>
<th>Atom</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in_pack(ds)</code></td>
<td>Directive</td>
<td>Data set <code>ds</code> is considered part of the package</td>
</tr>
<tr>
<td><code>process_as_schema(sch)</code></td>
<td>Directive</td>
<td>The data sets defined by <code>in_pack/1</code> atoms form a package of schema <code>sch</code> and are passed on for processing</td>
</tr>
</tbody>
</table>

**Variant: multiple packages per answer set**

<table>
<thead>
<tr>
<th>Atom</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>process(sch,[ds1,ds3,ds7])</code></td>
<td>Directive</td>
<td>The data sets in the list <code>[ds1,ds3,ds7]</code> form a package of schema <code>sch</code> and are passed on for processing</td>
</tr>
</tbody>
</table>
Definition
A context is a pair $C = \langle n, \mathcal{L} \mathcal{S} \rangle$ where $n \in \mathbb{N}$ is the name of the context and $\mathcal{L} \mathcal{S}$ is a logic suite.
Definition
An aMCS (of length $n$ with $m$ output streams) is a pair $M = \langle C, O \rangle$, where $C = \langle C_1, \ldots, C_n \rangle$ is an $n$-tuple of contexts and $O = \langle o_1, \ldots, o_m \rangle$ with $o_j \in \mathcal{N}$ for each $1 \leq j \leq m$ is a tuple containing the names of the output streams of $M$. 
Definition

A **data set** is a pair $d = \langle s, I \rangle$, where $s \in \mathcal{N}$ is either a context name or a sensor name, stating the **source** of $d$, and $I \subseteq \mathcal{I}$ is a set of pieces of information. An **information buffer** is a sequence of data sets.
Definition
Let $C = \langle n, \mathcal{L} \mathcal{I} \rangle$ be a context. A computation controller for $C$ is a relation $cc$ between a KB $KB \in \mathcal{KB}_{\mathcal{L} \mathcal{I}}$ and a finite information buffer.
Definition

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

$$\langle n, i \rangle \leftarrow b_1, \ldots, b_j, \text{not } b_{j+1}, \ldots, \text{not } b_m,$$

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

Jörg Pührer
Stream Packing for Asynchronous Multi-Context Systems
Definition

Let $C = \langle n, \mathcal{L} \mathcal{S} \rangle$ be a context, OR a set of output rules for $C$, $\in \mathcal{BS}_\mathcal{L}\mathcal{S}$ a belief set, and $n' \in \mathcal{N}$ a name. Then, the data set

$$d_C(\cdot, \text{OR}, n') = \langle n, \{ i \mid r \in \text{OR}, \text{hd}(r) = \langle n', i \rangle, |= \text{bd}(r) \} \rangle$$

is the output of $C$ with respect to OR under relevant for $n$. 
Definition
Let \( C = \langle n, L \mathcal{I} \rangle \) be a context. A configuration of \( C \) is a tuple \( cf = \langle KB, ACC, ib, cm \rangle \), where \( KB \in KB_L \mathcal{I} \), \( ACC \in ACC_L \mathcal{I} \), \( ib \) is a finite information buffer, and \( cm \) is a context management for \( C \) which is a triple \( cm = \langle cc, cu, OR \rangle \), where

- \( cc \) is a computation controller for \( C \),
- \( OR \) is a set of output rules for \( C \), and
- \( cu \) is a context update function for \( C \) which is a function that maps an information buffer \( ib = d_1, \ldots, d_m \) and an admissible knowledge base of \( L \mathcal{I} \) to a configuration \( cf' = \langle KB', ACC', ib', cm' \rangle \) of \( C \) with \( ib' = d_k, \ldots, d_m \) for some \( k \geq 1 \).
Definition

Let \( M = \langle\langle C_1, \ldots, C_n\rangle, \langle o_1, \ldots, o_m\rangle\rangle \) be an aMCS. A configuration of \( M \) is a pair

\[ Cf = \langle\langle cf_1, \ldots, cf_n\rangle, \langle ob_1, \ldots, ob_m\rangle\rangle, \]

where

- for all \( 1 \leq i \leq n \) \( cf_i = \langle KB, ACC, ib, cm\rangle \) is a configuration for \( C_i \) and for every output rule \( r \in OR_{cm} \) we have \( n \in \mathcal{N}(M) \) for \( \langle n, i \rangle = hd(r) \), and

- \( ob_j = \ldots, d_{l-1}, d_l \) is an information buffer with a final element \( d_l \) that corresponds to the data on the output stream named \( o_j \) for each \( 1 \leq j \leq m \) such that for each \( h \leq l \) with \( d_h = \langle n, i \rangle \) we have \( n = n_{C_i} \) for some \( 1 \leq i \leq n \).
Definition
Let \( M = \langle \langle C_1, \ldots, C_n \rangle, \langle o_1, \ldots, o_m \rangle \rangle \) be an aMCS. A run structure for \( M \) is a sequence

\[
R = \ldots, Cf^t, Cf^{t+1}, Cf^{t+2}, \ldots,
\]

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

Let $M$ be an aMCS of length $n$ with $m$ output streams and $R$ a run structure for $M$. $R$ is a run for $M$ if the following conditions hold for every $1 \leq i \leq n$ and every $1 \leq j \leq m$:

(i) if $cf_i^t$ and $cf_i^{t+1}$ are defined, $C_i$ is neither busy nor waiting at time $t$, then

- $C_i$ is busy at time $t+1$,
- $cf_i^{t+1} = cu_{cm_i^t}(ib_i^t, KB_i^t)$

We say that $C_i$ started a computation for $KB_i^{t+1}$ at time $t+1$.

(ii) if $C_i$ started a computation for $KB$ at time $t$ then

- we say that this computation ended at time $t'$, if $t'$ is the earliest time point with $t' \geq t$ such that $\langle n_{C_i}, EOC \rangle$ is added to every stakeholder buffer $b$ of $C_i$ at $t'$; the addition of $d_{C_i}(, OR_{cm_i^{t''}}, n)$ to $b$ is called an end of computation notification.
- for all $t' > t$ such that $cf_i^{t'}$ is defined, $C_i$ is busy at $t'$ unless the computation ended at some time $t''$ with $t < t'' < t'$.
- if the computation ended at time $t'$ and $cf_i^{t'+1}$ is defined then $C_i$ is not busy at $t'+1$. 

Definition

(iii) if $C_i$ started a computation for KB at time $t$ that ended at time $t'$ then for every belief set $\in ACC^t_i$ there is some time $t''$ with $t \leq t'' \leq t'$ such that

- $d_{C_i}(\cdot, OR_{cm_{t''}^i}, n)$ is added to every stakeholder buffer $b$ of $C_i$ for $n$ at $t''$.

We say that $C_i$ computed at time $t''$. The addition of $d_{C_i}(\cdot, OR_{cm_{t''}^i}, n)$ to $b$ is called a belief set notification.

(iv) if $\text{obj}_{j}^{t}$ and $\text{obj}_{j}^{t+1}$ are defined and $\text{obj}_{j}^{t} = \ldots, d_{l-1}, d_{l}$ then $\text{obj}_{j}^{t+1} = \ldots, d_{l-1}, d_{l}, \ldots, d_{l'}$ for some $l' \geq l$. Moreover, every data set $d_{l''}$ with $l < l'' \leq l'$ that was added at time $t + 1$ results from an end of computation notification or a belief set notification.

(v) if $\text{cf}_{i}^{t}$ and $\text{cf}_{i}^{t+1}$ are defined, $C_i$ is busy or waiting at time $t$, and $\text{ib}_{i}^{t} = d_{1}, \ldots, d_{l}$ then we have $\text{ib}_{i}^{t+1} = d_{1}, \ldots, d_{l}, \ldots, d_{l'}$ for some $l' \geq l$. Moreover, every data set $d_{l''}$ with $l < l'' \leq l'$ that was added at time $t + 1$ either results from an end of computation notification or a belief set notification or $n \notin N(M)$ (i.e., $n$ is a sensor name) for $d_{l''} = \langle n, i \rangle$. 