# Streaming Multi-Context Systems*

**Minh Dao-Tran** and **Thomas Eiter**
Institute of Information Systems, Vienna University of Technology
Favoritenstraße 9-11, A-1040 Vienna, Austria
{dao,eiter}@kr.tuwien.ac.at

## Abstract

Multi-Context Systems (MCS) are a powerful framework to interlink heterogeneous knowledge bases under equilibrium semantics. Recent extensions of MCS to dynamic data settings either abstract from computing time, or abandon a dynamic equilibrium semantics. We thus present streaming MCS, which have a run-based semantics that accounts for asynchronous, distributed execution and supports obtaining equilibria for contexts in cyclic exchange (avoiding infinite loops); moreover, they equip MCS with native stream reasoning features. Ad-hoc query answering is NP-complete while prediction is PSpace-complete in relevant settings (but undecidable in general); tractability results for suitable restrictions.

## 1 Introduction

Rooted in McCarthy's seminal work [1993], multi-context systems (MCS) model knowledge bases (KBs) interlinked by bridge rules, cf. [Giunchiglia et al., 1994; Ghidini et al., 2001; Brewka et al., 2007; Bikakis et al., 2010]; in particular, Brewka and Eiter [2007] provided a powerful generic framework to interlink heterogeneous information sources in an equilibrium semantics. Towards handling dynamic data, MCS have recently been extended to reactive MCS (rMCS) [Brewka et al., 2014] and evolving MCS (eMCS) [Goncalves et al., 2014] which feature sequences of equilibria obtained synchronously, and to asynchronous MCS (aMCS) [Ellmauthaler and Pührer, 2015] in which controllers trigger local KB to update and to output rules to push beliefs to other contexts.

However, no extension of MCS has features for handling *streaming* data [Della Valle et al., 2009; Arasu et al., 2006; Heintz, 2010]. In contrast to processing (sequences of) static data, data stream processing deals with continuous computation, which typically requires some explicit reference to time.

**Example 1** Fig. 1 depicts two robots $R_1$ (at node 3) and $R_2$ (at node 9), in an indoor environment (e.g., department store). They must deliver packages $P_1$ (at node 9) and $P_2$ (at node 4) to destinations $D_1$ (node 7) and $D_2$ (node 1), resp. Only $R_1$ can travel from 6 to 7, so he could take the path 3-4-5-8-9 to pick up $P_1$ and then deliver via 9-8-6-7. Similarly, $R_2$ would take 9-8-5-4-2-1. To minimize the combined travel distance, they can coordinate and alternatively pick up the packages close by and exchange them at node 5. Reaching an agreement may be challenging when they are already moving, and e.g. a link is not usable as too many people were recently observed on it. This may be inferred from a time-based window of a stream with position data of people. ■

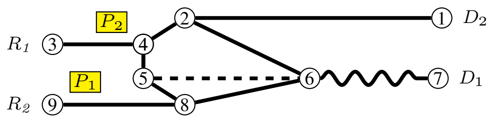

Figure 1: Robot scenario

To reason over such asynchronous information exchange systems, communication and local computation time must be considered, and only recently sent/received data will be relevant. Moreover, in general no stable state of information exchange may be achieved, e.g. an agreement on a meeting point for the robots. In the static case, but also in rMCS and in eMCS, computing equilibria is considered to be timeless; in our dynamic setting, however, data (also new requests) keep streaming *while* any computation is ongoing; this may render the result upon completion invalid. While aMCS account for controlled information exchange, physical computation and transfer time are disregarded and no baseline mechanism to achieve an equilibrium is considered.

To tackle these issues, we introduce *Streaming Multi-Context Systems* (sMCS), with the following contributions.

- To equip MCS for data streams, we base sMCS on managed MCS (mMCS) [Brewka et al., 2014] and extend bridge rules with *window atoms* for snapshots of input streams at contexts. Although it is possible extending rMCS/eMCS to emulate window atoms, having them as first-class citizens in the formalism allows sMSC while add streaming features to MCS, still retain MCS' principles, namely: bridge rules are simple to import knowledge and contexts are abstract. We discuss this issue in more detail in Section 6.
- Our framework models transfer time of data between and computation time at contexts, to reflect the asynchronous behavior that arises in general, and has an internal execution control at the contexts.

*This research has been supported by the Austrian Science Fund (FWF) project P26471.

• We define a run-based semantics of sMCS as special sequences of states. In that, we introduce *feedback equilibria*, which are a non-trivial lifting of the key notion of MCS equilibria to the asynchronous setting, by which local stability in runs can be enforced. They provide semantic middle-ware to overcome potentially infinite loops; e.g., if the robots in Ex. 1 would mutually send proposals and keep adjusting them, they may never agree on a meeting point.

• Finally, we consider reasoning in sMCS and characterize the complexity of ad-hoc queries and prediction. To our knowledge, the latter has not been considered for MCS so far.

## 2 Streams

To set up a streaming environment, we borrow some formal notions from the LARS framework [Beck *et al.*, 2015]. We assume an underlying set $\mathcal{A}$ of ground (propositional) atoms.

**Definition 1** A stream $S = (T, \upsilon)$ consists of a *timeline* $T$, which is a closed interval $T \subseteq \mathbb{N}$ of integers called *time points*, and an *evaluation function* $\upsilon \colon \mathbb{N} \mapsto 2^{\mathcal{A}}$.

Intuitively, a stream $S$ associates with each time point a set of atoms. If $S = (T, \upsilon)$ and $S' = (T', \upsilon')$ are streams such that $T' \subseteq T$ and $\upsilon'(t') \subseteq \upsilon(t')$ for all $t' \in T'$, we write $S' \subseteq S$ and call $S'$ a *substream* or *window* of $S$.

We call streams that provide input also *data streams*. To cope with large input volume, one usually considers only recent atoms by applying window mechanisms on streams.

**Definition 2 (Window Function)** Any (computable) function $w$ that returns, given a stream $S = (T, \upsilon)$ and a time point $t \in T$, a substream $S'$ of $S$ is called a *window function*.

Prominent are *time-based* window functions, which select the atoms of the latest $k$ time points. Formally, the *(sliding) time-based window of size* $k$ is $\tau(k)(S,t) = (T', \upsilon|_{T'})$, a substream of $S$, where $S{=}(T,\upsilon)$ is any stream and $t$ any time point in $T{=}[t_1, t_2]$ and $T' = [t', t]$ such that $t' = \max\{t_1, t-k\}$.

To express formulas on streams, LARS offers window and temporal operators. For sMCS, we borrow *plain LARS window atoms* (briefly *window atoms*), of the form ($a \in \mathcal{A}$ and $t' \in \mathbb{N}$):

$$\alpha ::= \boxplus^w @_{t'} a \mid \boxplus^w \Diamond a \mid \boxplus^w \Box a\,. \qquad (1)$$

A *streaming atom* is either an atom or a window atom. Entailment of a streaming atom from a stream $S = (T, \upsilon)$ at a time point $t$ is defined as follows, where $S' = (T', \upsilon') = w(S,t)$:

$$\begin{array}{lll} S,t \Vdash a & \text{iff} & a \in \upsilon(t) \ \text{ and } t \in T \\ S,t \Vdash \boxplus^w \Diamond a & \text{iff} & a \in \upsilon'(t') \text{ for some } t' \in T' \\ S,t \Vdash \boxplus^w \Box a & \text{iff} & a \in \upsilon'(t') \text{ for all } t' \in T' \\ S,t \Vdash \boxplus^w @_{t'} a & \text{iff} & a \in \upsilon'(t') \text{ and } t' \in T' \end{array}$$

For a streaming atom $\alpha$, $S,t \Vdash \text{not}\,\alpha$ iff $S,t \nVdash \alpha$. Furthermore, let $S,t \Vdash \top$ for all $t \in T$, where $\top$ is a special atom. For convenience, we abbreviate $\boxplus^{\tau(k)}$ with $\boxplus^k$.

**Example 2** Let $block(X,Y)$ states that a link $(X,Y)$ is blocked; then $\boxplus^3 \Box block(a,b)$ holds at $t$ if the link $(a,b)$ has been always blocked in the last 3 time units. ■

For any window atom $\alpha$ of form (1), let $at(\alpha)$ be the ordinary atom extracted from $\alpha$, that is, $at(\alpha) = a$.

## 3 Streaming Multi-Context Systems

We now define streaming Multi-Context Systems as a generalization of mMCS [Brewka *et al.*, 2011] to handle streaming data. The main idea is that contexts, interlinked by bridge rules, continuously receive input from neighbors, compute, and then send output to other contexts. To govern data flows coming to contexts, we extend bridge rules with window atoms.

To better highlight the issues relevant for this paper, we confine to mMCS in which (1) each context has a single logic rather than a logic suite, (2) the management functions are deterministic. An extension to arbitrary mMCS is not difficult but technically somewhat involved.

First, we recall the notion of logics and contexts in mMCS. A *logic* $L$ is a triple $\langle \mathbf{KB}_L, \mathbf{BS}_L, \mathbf{ACC}_L \rangle$, where $\mathbf{KB}_L$ is the set of admissible knowledge bases (KBs) of $L$, $\mathbf{BS}_L \subseteq 2^{BS_L}$ is the set of possible belief sets over a set of beliefs $BS_L$, and $\mathbf{ACC}_L \colon \mathbf{KB}_L \to 2^{\mathbf{BS}_L}$ represents the semantics of $L$ by assigning to each $kb \in \mathbf{KB}_L$ a set of acceptable belief sets.

Without loss of expressiveness, we consider a setting in which beliefs are ordinary ground atoms.

A *context* has the form $C = \langle L, ops, mng \rangle$, where $L$ is a logic, $ops$ is a set of operations, and $mng \colon 2^{ops} \times \mathbf{KB}_L \to \mathbf{KB}_L$ is a management function. For an indexed context $C_i$ we write $L_i$, $ops_i$, and $mng_i$ to denote its components.

Important in a streaming environment are sensors that continuously emit readings: this can be modeled by *sensor logics* of the form $L_{\mathcal{D}}{=}\langle \{\emptyset\}, \mathbf{BS}_{\mathcal{D}}, \mathbf{ACC}_{\mathcal{D}} \rangle$, where $\emptyset$ is a dummy local KB, $\mathcal{D}$ is a value domain (possible readings) and $\mathbf{BS}_{\mathcal{D}}{=}\{s \subseteq \mathcal{D} \mid |s| \leq 1\}$, $\mathbf{ACC}_{\mathcal{D}}(\emptyset){=}\mathbf{BS}_{\mathcal{D}}$. Intuitively, either the empty set or a set with a single value from $\mathcal{D}$ can be an acceptable belief set of $L_{\mathcal{D}}$.

We now extend bridge rules with *streaming bridge atoms*.

**Definition 3** Let $\mathsf{C} = \langle C_1, \ldots, C_n \rangle$ be a *tuple of contexts*. A *streaming bridge rule* $r$ for $C_i$ over $\mathsf{C}$ is of the form

$$\mathsf{op} \leftarrow \beta_1, \ldots, \beta_j, \text{not } \beta_{j+1}, \ldots \text{not } \beta_m, \qquad (2)$$

where $\mathsf{op} \in ops_i$ and each $\beta_\ell = (c_\ell{:}\alpha_\ell)$ is a *bridge atom* where $c_\ell \in \{1, \ldots, n\}$ and $\alpha_\ell$ is a streaming atom for $C_{c_\ell}$, i.e., $\alpha_\ell \in BS_{L_{c_\ell}}$ or $at(\alpha_\ell) \in BS_{L_{c_\ell}}$. We denote by $H(r) = \mathsf{op}$ the *head* and by $B(r) = \{\beta_1, \ldots, \beta_j, \text{not } \beta_{j+1}, \ldots, \text{not } \beta_m\}$ the *body* of $r$.

*Sensor contexts* are contexts with sensor logics, no bridge rules, no operation, and the management function satisfying that $mng(\emptyset, \emptyset) = \emptyset$. We can now define streaming multi-context systems.

**Definition 4** A *streaming multi-context system* (sMCS) $M = \langle \mathsf{C}, \mathsf{BR}, \mathsf{KB} \rangle$ consists of

- a tuple $\mathsf{C} = \langle C_1, \ldots, C_n \rangle$ of contexts,
- a tuple $\mathsf{BR} = \langle br_1, \ldots, br_n \rangle$ of sets $br_i$ of streaming bridge rules for $C_i$ over $\mathsf{C}$,
- a tuple $\mathsf{KB} = \langle kb_1, \ldots, kb_n \rangle$ of KBs $kb_i \in \mathbf{KB}_{L_i}$.

**Example 3** Fig. 2 shows Ex. 1 as sMCS $M = (C_1, \ldots, C_6)$:

• $C_i$, $4 \leq i \leq 6$, is a sensor context that feeds data to $C_{i-3}$; $C_4$ (resp. $C_5$) tells the current position $pos(X,Y,L)$ of robot $R_1$ ($R_2$), meaning it is at $L$% on the way from $X$ to $Y$. $C_6$ provides data $n(X,Y,N)$ that $N$ people were sensed on the link $(X,Y)$.

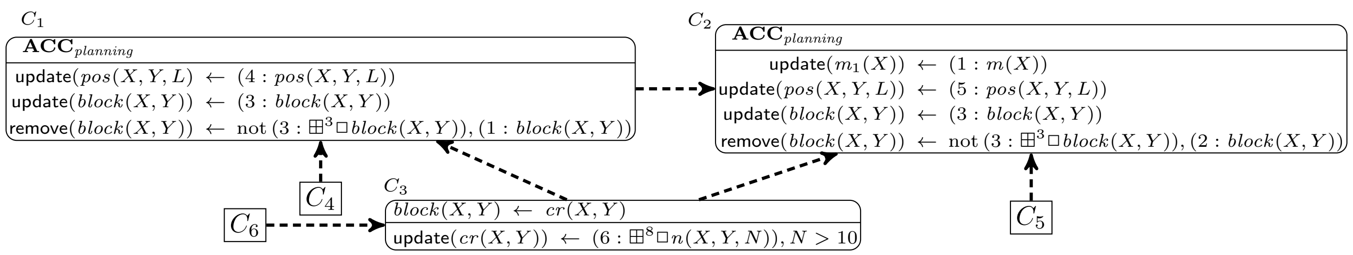
$C_1$
$\mathbf{ACC}_{planning}$
$\mathsf{update}(pos(X,Y,L)) \leftarrow (4 : pos(X,Y,L))$
$\mathsf{update}(block(X,Y)) \leftarrow (3 : block(X,Y))$
$\mathsf{remove}(block(X,Y)) \leftarrow \text{not}\,(3 : \boxplus^3 \Box block(X,Y)), (1 : block(X,Y))$
$C_2$
$\mathbf{ACC}_{planning}$
$\mathsf{update}(m_1(X)) \leftarrow (1 : m(X))$
$\mathsf{update}(pos(X,Y,L)) \leftarrow (5 : pos(X,Y,L))$
$\mathsf{update}(block(X,Y)) \leftarrow (3 : block(X,Y))$
$\mathsf{remove}(block(X,Y)) \leftarrow \text{not}\,(3 : \boxplus^3 \Box block(X,Y)), (2 : block(X,Y))$
$C_4$
$C_3$
$block(X,Y) \leftarrow cr(X,Y)$
$\mathsf{update}(cr(X,Y)) \leftarrow (6 : \boxplus^8 \Box n(X,Y,N)), N > 10$
$C_6$
$C_5$

Figure 2: Modeling the Robot Scenario with sMCS

- $C_3$ takes the sensor data from $C_6$, reasons about blocked links and sends this information to $C_1$ and $C_2$.
- $C_1$ (resp. $C_2$) aims to find a shortest route from $R_1$'s (resp. $R_2$'s) position to its destination respecting blocked links and in $C_2$ possible meeting points $m(X)$.

Focusing on the streaming aspects, we omit the details of the local planning logics at $C_1$, $C_2$ , except a notice that if a new plan proposes the same meeting point as the previous plan, then this information is not resent to the other context. $\mathbf{ACC}_{planning}$ in $C_1$ can either remember the latest proposed meeting point on its own, or self-import this information by adding the following bridge rule into $br_1$:

$$\mathsf{update}(m_{old}(X) \leftarrow (1 : m(X))).$$

We now give a brief explanation on the bridge rules. Intuitively, $\mathsf{update}(p(\mathbf{X}))$ drops the facts of predicate $p$ and then adds a new ground fact $p(\mathbf{X})$, while $\mathsf{remove}(p(\mathbf{X}))$ removes $p(\mathbf{X})$. The bridge rules of $br_1$ and $br_2$ with remove in the head drop blocking information for a link $(X, Y)$, if it was not continuously reported blocked in the last 3 time units.

The bridge rule in $br_3$ informs $kb_3$ about crowded links using facts $cr(X, Y)$. To keep $C_3$'s local reasoning simple for the moment, such links are viewed as blocked. ■

## 3.1 Semantics of Streaming MCS

We define the semantics of sMCS in terms of runs, which are sequences of states with constraints on the information exchange between contexts and their local semantics, as well as on the time spent for that. Before going into the details of the semantics, we recall two important aspects of stream processing, namely *execution modes* and *execution policies*.

In stream processing, there are two execution modes:

- *Pulling*: an engine periodically executes, e.g. every 5 seconds, regardless of when input arrives; this amounts to *time-driven* evaluation.
- *Pushing*: an engine executes immediately when input arrives; this is also known as *eager* mode, and amounts to *data-driven* resp. *event-driven* evaluation.

However, an engine might still be busy with a current computation while an execution should be triggered according to its execution mode, for example, when the time reaches a period in pulling mode or when a new input arrives in pushing mode.

An engine can follow either the *ignore* or *restart* policy in such situations. The former means that the engine continutes its computation while the latter means that it abandons the current computation and starts with a new one according to the respective execution mode.

The combination of "pushing" and "ignore" can lead to situations in which an execution is carried out immediately after a previous execution finished because some input arrived during the previous execution. That is, the actual start of an execution is different from the intention to start it according to the execution mode. We introduce in the following the notion of *states* to model such situation.

**States.** A *state* of context $C_i$ is a triple $\mathbf{s}_i{=}(s_i, o_i, kb_i)$ where

- $s_i \subseteq \{IE, SE\}$ is the *execution status*: $IE$ means an intent to start an execution, either proactively (*pulling*), or reactively on new input (*pushing*); $SE$ means a computation actually starts. Sensor contexts always have status $\{IE, SE\}$;
- $o_i \in \mathbf{BS}_{L_i} \cup \{\epsilon\}$ is the *output*, which is streamed to other contexts, unless $o_i = \epsilon$ (meaning that nothing will be sent to other contexts);
- $kb_i \in \mathbf{KB}_{L_i}$ is the local KB (which can evolve).

A *(global) state* of $M$ is any tuple $\mathbf{s} = (\mathbf{s}_1, \ldots, \mathbf{s}_n)$ of states $\mathbf{s}_i$ of context $C_i$, $1 \leq i \leq n$. By $\mathbf{s}_i(t') = (s_i(t'), o_i(t'), kb_i(t'))$ we denote the state of $C_i$ at time $t'$. Towards runs, we define:

**Definition 5** Given a timeline $T$, a *state sequence of* $M$ up to $t \in T$ is any sequence $\mathbf{s} = \mathbf{s}(0), \ldots, \mathbf{s}(t)$ of global states $\mathbf{s}(t') = (\mathbf{s}_1(t'), \ldots, \mathbf{s}_n(t'))$.

Arbitrary state sequences may not align belief output with respective KB evaluation. For that, input from bridge rules must be respected, as well as the actual start of computations: if an intent arises while $C_i$ is busy, it may (by some policy) *restart* the computation or *ignore* the intent until the current computation ends. Runs will enforce these conditions to faithfully capture the evolution of sMCS. We next address the issue of input.

**Input streams of contexts.** Any state sequence $\mathbf{s} = \mathbf{s}(0), \ldots, \mathbf{s}(t_{end})$ naturally induces an output stream $S_k = (T, \upsilon_k)$ of $C_k$, where $T = [0, t_{end}]$ and $\upsilon_k(t) {=} o_k(t)$ for all $t \in T$. However, due to transfer time this output is not immediately available as input at other contexts $C_i$. To simplify presentation, we assume in our basic model that this transfer takes $\Delta_{ki}$ units of time. This can be generalized to functions with different parameters, e.g., the time point $t$ or the size of $o_k(t)$. We define input streams as follows.

Suppose that $C_i$ *accesses* $C_k$, i.e., some bridge rule $r \in br_i$ has some atom $(k{:}\alpha)$ in the body. Then the input stream of $C_i$

from $C_k$ at time $t$ based on $\mathbf{s}$ is $S_{ki}^{\mathbf{s},t} = (T, \iota\upsilon_{ki}^{\mathbf{s},t})$, where

$$\iota\upsilon_{ki}^{\mathbf{s},t}(t_{in}) = \begin{cases} \emptyset & \text{if } t_{in} < \Delta_{ki} \text{ or } t_{in} > t \\ \upsilon_k(t_{in} - \Delta_{ki}) & \text{otherwise.} \end{cases} \quad (3)$$

Intuitively, due to the delay $\Delta_{ki}$, no output of $C_k$ can be expected at $C_i$ from time points 0 to $\Delta_{ki}$; for further time points up to $t$, $C_i$ receives the output from $C_k$ with delay.

We also omit $\mathbf{s}$ and/or $t$ if clear from the context.

**Example 4 (cont'd)** Consider contexts $C_1, C_2, C_3$, and $C_6$ with output streams $S_i = ([0, 20], \upsilon_i)$, where $\upsilon_3(10) = \{block(5,6)\}$, $\upsilon_1(2) = \{m(5)\}$, $\upsilon_1(14) = \{m(6)\}$ and $\upsilon_6(t') = \{n(5,6,15)\}$ for $t' \in \{4, \dots, 8\}$. Let $t = 20$; then

- for $\Delta_{12}=1$, $S_{12}$ has $\iota\upsilon_{12}(3)=\{m(5)\}$, $\iota\upsilon_{12}(15)=\{m(6)\}$;
- for $\Delta_{31}=1$, $S_{31}$ has $\iota\upsilon_{31}(11) = \{block(5,6)\}$;
- for $\Delta_{63}=1$, $S_{63}$ has $\iota\upsilon_{63}(t')=\{n(5,6,15)\}$, $5 \le t' \le 9$.

A bridge rule $r \in br_i$ is *applicable* wrt. a state sequence $\mathbf{s}$ at $t$ (in symbols, $\mathbf{s} \Vdash_t^i r$), if $S_{ki}, t \Vdash \alpha$ for every $(k\!:\!\alpha) \in B(r)$ and $S_{ki}, t \nVdash \alpha$ for every $\mathbf{not}\,(k\!:\!\alpha) \in B(r)$. We denote with $app_i(\mathbf{s}, t)=\{H(r) \mid r \in br_i \wedge \mathbf{s} \models_t^i B(r)\}$ the heads of all applicable bridge rules of $C_i$ wrt. $\mathbf{s}$ at $t$.

**Evaluation time.** As for asynchronous execution, we adopt that besides the transfer times $\Delta_{ki}$, functions $f_i$ measure the time that $C_i$ needs to (1) evaluate the bridge rules, (2) run the management functions for local KB update, and (3) evaluate the updated KBs. Clearly, such functions may depend on various parameters, and different levels of detail are possible. As a baseline, we assume that $f_i$ assigns each pair $(br_i, kb_i)$ a natural number. This can be generalized with further parameters (e.g., time) and/or intervals bounding the evaluation times; the latter allows us to express uncertainty and reduces to the baseline via all induced point measurements. Alternatively, probabilistic estimations can be considered. For sensor contexts, we simply may adopt $f_i$=0.

**From state sequences to runs.** Given a state sequence $\mathbf{s}$, a context $C_i$, and a time point $t$, the latest time point when $C_i$ was triggered to execute is denoted by ($\max \emptyset = -1$):

$$tr(\mathbf{s}, C_i, t) = \max\{t' \le t \mid SE \in s_i(t')\}.$$

Runs are state sequences where context output stems from its local KB and the input at the closest triggered time. Formally,

**Definition 6** A *run* for an sMCS $M$ is a state sequence $\mathbf{s}=(\mathbf{s}_0, \dots, \mathbf{s}_{t_{end}})$ such that $o_i(0) = \epsilon$, $kb_i(0) = kb_i$, and for all $t \in [1, t_{end}]$:

$$o_i(t) \neq \epsilon \text{ iff } o_i(t) \in \mathbf{ACC}_i(kb_i(t)), \quad (4)$$

where:

(i) $kb_i(t) = mng_i(app_i(\mathbf{s}, t_{ex}), kb_i(t_{ex}))$,

(ii) $t_{ex} = tr(\mathbf{s}, C_i, t)$,

(iii) $t = t_{ex} + f_i(br_i, kb_i(t_{ex}))$,

(iv) $kb_i(t') = kb_i(t_{ex})$ for all $t' \in (t_{ex}, t)$.

Intuitively, when a context $C_i$ produces some output $o_i(t)$ at a time point $t$, this output must be computed on (i) the updated local KB based on input stream at (ii) the closest triggered time point $t_{ex}$. Furthermore, the distance from $t_{ex}$ to $t$ must be (iii) the computation time of the respective input. Finally, during computation, the update of the local KB is hidden to the outside of the context (iv).

**Example 5 (cont'd)** Fig. 3 illustrates a run of the sMCS where subscript $_i$ denotes output of $C_i$, $i = 1, 2$ (we focus on the robots). We have $\Delta_{12}$=$\Delta_{31}$=1 and $\Delta_{ij}$=0 otherwise; $f_1(br_1, kb_1)$=2, $f_2(br_2, kb_2)$=4, and $f_3(br_3, kb_3)$=1 for all $kb_i \in \mathbf{KB}_{L_i}$. Contexts $C_1, C_2, C_3$ operate in pushing mode; $C_1$ ignores new input if it is busy, while $C_2$ and $C_3$ restart.

The sensor contexts $C_4, C_5$ stream data at times $5k$, $k \ge 0$, $C_6$ streams data at each time point. As $\Delta_{41}$=$\Delta_{52}$=0, $C_1$ and $C_2$ receive input and intend to start execution at times $5k$.

At $t = 2$, $C_1$ finds a plan to meet at node 5. This arrives at $C_2$ at $t$=3, and $C_2$ restarts its computation. At $t$=5, the contexts start executions wrt. new incoming data. $C_2$ again aborts its previous computation. At $t$=7 and $t$=9, resp., the contexts come up with plans to meet at node 5.

Assume $C_3$ sends $block(5,6)$ at $t = 10$, which $C_2$ receives at 10 and $C_1$ at 11 (both started to execute at 10). $C_1$ ignores $block(5,6)$ at time 11 and starts executing with this input at time 12; this results in a new plan of $C_1$ at $t = 14$ to meet at node 6. A respective request arrives at $C_2$ at $t = 15$. $C_1$ and $C_2$, with input from $C_4, C_5$, start executions at $t = 15$ that end at time 17, resp. 19, and yield plans to meet at node 6. ■

To capture step-wise runs with equilibria semantics in the sense of rMCS and eMCS, we need to model an idealized setting in which contexts have unlimited power to compute equilibria between two consecutive time points (akin to flip-flops in clocked hardware circuits). We can achieve this with a computation time so small such that the $\mathbf{ACC}$ function can be run finitely often (as an algorithm to compute an equilibrium will do) between two consecutive time points. Formally, we let the transferring time be 0 and the computation time be an *infinitesimally small* value $\varepsilon$ such that (i) $t < t + \varepsilon$ and (ii) $\varepsilon + \varepsilon = \varepsilon$; thus $t + n \times \varepsilon = t + \varepsilon$, for each integer $n > 0$.

We now adapt the definition of input streams $S_{ki}^{\mathbf{s},t}$ to $S_{ki}^{\mathbf{s},t,\varepsilon}$ with $\varepsilon$ by adapting Equation (3) with $\Delta_{ki} = 0$ as follows:

$$\iota\upsilon_{ki}^{\mathbf{s},t,\varepsilon}(t_{in}) = \begin{cases} o_k(t_{in}), & \text{if } t_{in} < t, \\ o_k(t_{in} + \varepsilon), & \text{if } t_{in} = t. \end{cases} \quad (3')$$

Moving output $o_k(t_{in} + \varepsilon)$ to $\iota\upsilon_{ki}^{\mathbf{s},t,\varepsilon}(t_{in})$ is just a technique to respect the cyclic dependency between contexts in defining equilibria. Another approach that "shifts" the time line with $\varepsilon$ is possible but more cumbersome (as non-integer time lines must be introduced) and thus not further considered here.

Then, we adapt the applicability relationship $\Vdash_t^i$ to $\Vdash_{t,\varepsilon}^i$ by substituting $S_{ki}^{\mathbf{s},t}$ with $S_{ki}^{\mathbf{s},t,\varepsilon}$, and respectively, adapt $app_i(\mathbf{s}, t)$ to $app_i^\varepsilon(\mathbf{s}, t)$. Finally, idealized runs can be defined as follows:

**Definition 7** An *idealized run* for an sMCS $M$ is a state sequence $\mathbf{s} = (\mathbf{s}_0, \dots, \mathbf{s}_{t_{end}})$ such that $o_i(0) = \epsilon$, $kb_i(0) = kb_i$, and for all $t \in [0, t_{end})$:

$o_i(t+1) = o_i(t+\varepsilon)$ and $kb_i(t+1) = kb_i(t+\varepsilon)$, where

- $o_i(t+\varepsilon) \in \mathbf{ACC}(kb_i(t+\varepsilon))$ and
- $kb_i(t+\varepsilon) = mng_i(app_i^\varepsilon(\mathbf{s}, t), kb_i(t))$.

Intuitively, at $t + \varepsilon$ we have infinite power to respect the cyclic dependency and thus to compute equilibria. The result at this step is then placed at the next time point $t + 1$.

Note that simply setting $\Delta_{ki} = f_i = 0$ in Definition 6 does not help us defining idealized runs from runs, as the latter are

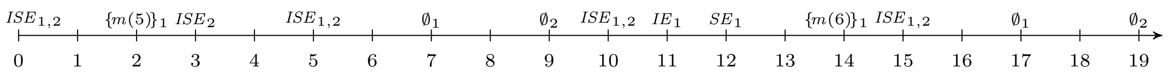

Figure 3: Run trace ($ISE_{1,2} = \{IE_1, SE_1, IE_2, SE_2\}, ISE_2 = \{IE_2, SE_2\}$)

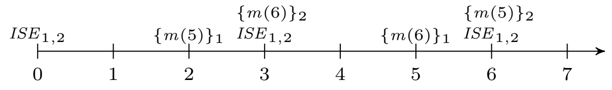

Figure 4: Communication loop ($ISE_{1,2} = \{IE_1, SE_1, IE_2, SE_2\}$)

stateful with changing local KBs. Here, the introduction of $\varepsilon$ is necessary as it fulfills the intuition that even in idealized settings computation still takes time, but infinitesimally small so that one can compute equilibria between to consecutive time points.

Our sMCS relate to mMCS, eMCS and rMCS as follows. First, sMCS generalize mMCS.

**Proposition 1** *Let $M$ be an sMCS where contexts run in pulling mode and bridge rules have no window atoms. Then $M$ can be viewed as an mMCS $M^{\mathsf{m}}$ that outputs in idealized runs at each time an equilibrium obtained by $M^{\mathsf{m}}$.*

Given an rMCS (resp. eMCS) $M$, we can construct a corresponding sMCS $M^{\mathsf{s}}$ by turning sensors (resp. observation contexts) into sensor contexts and their observations into respective output streams. This allows one to capture rMCS.

**Proposition 2** *The runs of an rMCS M [Brewka et al., 2014] amount to the idealized runs of its corresponding sMCS $M^{\mathsf{s}}$.*

For eMCS, however, this construction needs some condition.[1]

**Proposition 3** *Let $M$ be an eMCS such that* $\mathsf{op}(a) \leftarrow B$ *is in $br_i$ iff* $next(\mathsf{op}(a)) \leftarrow B$ *is in $br_i$. Then, the evolving equilibria of $M$ amount to idealized runs of the sMCS $M^{\mathsf{s}}$.*

Here, $next(\mathsf{op}(a))$ belongs to the evolving operational formulas introduced in eMCS. Intuitively, such operations update the local KBs while operations $\mathsf{op}(a)$ are only used to provide the KB for computing the equilibria at a single step.

## 4 Feedback Equilibria

As shown in Prop. 1, traditional mMCS equilibria can only be achieved with idealized runs, which impose extreme conditions. Yet they apply if the stream granularity is significantly larger than evaluation and transfer time at resp. between contexts, such that an equilibrium computation can be carried out between two consecutive time points. Asynchronous runs are closer to practice. However, total asynchrony can evoke uncomfortable situations if contexts depend on each other.

**Example 6 (cont'd)** Suppose also $C_2$ suggests a meeting point and $C_1$ has a further bridge rule $\mathsf{update}(m_2(X)) \leftarrow (2{:}m(X))$ to import the suggestion to $kb_1$. Moreover, the local planning logic checks whether the suggestions of the two robots match; if not, another round of planning is carried out and new suggestions are sent. Then if $f_1(br_1, kb_1)$=2, $f_2(br_2, kb_2) = 3$ for all $kb_i \in \mathbf{KB}_{L_i}$ ($i = 1, 2$), $\Delta_{12} = 1$, and all other costs are 0, the contexts might get stuck in a loop just to agree on a meeting point, as depicted in Figure 4. ■

In practice, it is important to avoid that subsystems of an sMCS run into a (possibly infinite) loop due to asynchronous information exchange. For this we may want to obtain stability, i.e., a local equilibrium of such subsystems; however, how to accomplish this is not straightforward.

The key idea is to respect that an equilibrium is intrinsically timeless, and to dispense streaming data from outside for its computation; this is a trade-off for success, comparable to ignoring requests during fast interrupt handling in CPUs.

We consider strongly connected components (SCCs) in an sMCS $M$, i.e., in the graph $G(M)$ whose nodes are the contexts $C_i$ of $M$ and with edges $C_i \to C_j$ if $C_i$ accesses $C_j$. We extend the run semantics of sMCS as follows:

($\star$) $C_i$ can request stability of the SCC in which it occurs, denoted $\mathcal{C}_i$. When $C_i$ raises this request at time $t = t_{ex} + \Delta$ after it started execution at $t_{ex}$, the system restarts all other contexts in $\mathcal{C}_i$ to compute with their input at $t_{ex}$ and allows them to disclose output in $\mathcal{C}_i$ until reaching an equilibrium.

($\star\star$) Later at $t_{out} \geq t$, $\mathcal{C}_i$ reports an equilibrium, or restarts its contexts with input at time $t_{out}$ if no equilibrium exists.

From $t$ to $t_{out}$, the contexts in $\mathcal{C}_i$ are in an internal solving mode and output no beliefs to contexts outside $\mathcal{C}_i$; the latter still run asynchronously, unless stability is requested elsewhere. If simultaneous stability requests arise in $\mathcal{C}_i$, one is nondeterministically followed. We formalize this as follows.

**Definition 8** Let $\mathcal{C} = \{C_{i_1}, \dots, C_{i_\ell}\}$ be an SCC of an sMCS $M$. A *belief state of* $\mathcal{C}$ is a tuple $(BS_{i_1}, \dots, BS_{i_\ell})$ of belief sets $BS_{i_j} \in \mathbf{BS}_{i_j}$, $1 \leq j \leq \ell$; it is an *feedback equilibrium of* $\mathcal{C}$ *wrt. a run* $\mathsf{s}$ *of* $M$ *at time* $t$ if for all $j \in [1, \ell]$:

$$BS_{i_j} \in \mathbf{ACC}_{i_j}(mng_{i_j}(app^{\varepsilon}_{i_j}(\mathsf{s}, t), kb_{i_j}(t))).$$

Intuitively, a feedback equilibrium of $\mathcal{C}$ at $t$ is achieved by running its contexts in idealized mode with $\varepsilon$ computation time. Thus, input to $\mathcal{C}$ is fixed at $t$, and cyclic information flow inside $\mathcal{C}$ is respected.

Based on this, we define stability of runs as follows where a special atom $\rho$ denotes a stability request and a new status $EQ$ the equilibrium computation mode.

**Definition 9** Given an SCC $\mathcal{C}$=$\{C_{i_1}, \dots, C_{i_\ell}\}$ of an sMCS $M$, a run $\mathsf{s}$ of $M$ on a timeline $T = [0, t_{end}]$ is *locally stable* wrt. $\mathcal{C}$, if whenever at time $t$, the set $req(\mathcal{C}, t) := \{C_{i_j} \in \mathcal{C} \mid \rho \in o_{i_j}(t)\}$ is nonempty, then either

($\star$) $t_{out} \geq t$ and $t_{ex} = tr(\mathsf{s}, C_{i_j}, t)$ exist, where $C_{i_j} \in req(\mathcal{C}, t)$ such that

(i) $\mathcal{C}$ has at $t_{ex}$ wrt. $\mathsf{s}$ either (a) a feedback equilibrium $(BS_{i_1}, \dots, BS_{i_\ell})$ such that $o_{i_j}(t_{out}) = BS_{i_j}$ for $j \in$

[1]Extending sMCS to fully capture eMCS is not difficult but rather technically involved (by introducing streaming bridge rules with the *next* operator and keeping two local KB states as in eMCS). This is, however, not the main goal of this paper.

$[1,\ell]$, or (b) no feedback eq. and $s_{i_j}(t')=\{SE\}$, for $j\in[1,\ell]$, and

(ii) for all $t'' \in (t, t_{out})$ and $i_j \in \{i_1, \ldots, i_\ell\}$, we have $s_{i_j}(t'') = \{EQ\}$ and $o_{i_j}(t'') = \epsilon$;

or

($\star\star$) for all $t'' \in (t, t_{end}]$ and $i_j \in \{i_1, \ldots, i_\ell\}$, we have $s_{i_j}(t'') = \{EQ\}$ and $o_{i_j}(t'') = \epsilon$.

Furthermore, $\mathbf{s}$ is *locally stable*, if it is locally stable wrt. every SCC of $M$.

**Example 7 (cont'd)** Reconsider Fig. 4. In a locally stable run, $C_1$ realizes at $t=3$ that $C_2$'s suggestion does not match his sent at $t=2$. It can request local stabilization for its SCC $\mathcal{C}_1=\{C_1, C_2\}$ to find an equilibrium wrt. the input to $\mathcal{C}_1$ at $t_{ex}=0$. At some $t_{out}>3$, an equilibrium is computed which guarantees an agreed meeting point, e.g. at node 6. ■

In practice, we may bound $t_{out}$ such that the SCC $\mathcal{C}$ will be reset at $t_{out}$ the latest: if stabilization takes too long, the result may be outdated (in particular, for reasoning on recent data).

## 5 Reasoning with Streaming MCSs

For reasoning from an sMCS $M=(C_1, \ldots, C_n)$, we assume sensor contexts are $D=C_j, \ldots, C_n$. A *reading* for $M$ up to $t$ is a state sequence $\mathbf{r}=\mathbf{r}(0), \ldots, \mathbf{r}(t)$ such that $o_i^{\mathbf{r}}(t') = \emptyset$ for all $i \in [1, j)$ and $t' \in [0, t]$. Models are defined as follows.

**Definition 10** Given an sMCS $M$ with sensor contexts $D$, a run $\mathbf{s}$ for $M$ is a *model* wrt. a reading $\mathbf{r}$ up to $t$, if $|\mathbf{s}| = |\mathbf{r}|$ and $o_i^{\mathbf{s}}(t') = o_i^{\mathbf{r}}(t')$ for all $C_i \in D$ and $t' \in [0, t]$.

We consider *brave reasoning*: given $M$, a reading $\mathbf{r}$ up to $t$, a context $C_i$ and an atom $a$, decide if $a$ is true at time $t$ in some model $\mathbf{s}$ of $M$ wrt. $\mathbf{r}$, i.e., $a \in o_i^{\mathbf{s}}(t)$ holds; we denote this by $M, \mathbf{r} \models_b C_i(a)$. Model existence (*consistency*) and *cautious reasoning* easily reduce to this problem resp. its complement.

**Setting.** We assume that (a) each $C_i$ periodically decides in constant time, by looking at its input stream, whether to execute, and is in permanent ignoring or restarting mode; (b) algorithms are available (given as code) to evaluate the bridge rules $br_i$, to update the local $kb_i$ by the management function $mng_i$, and to compute some $BS_i \in \mathbf{ACC}_i(kb_i)$ (where each possible $BS_i$ may result); unless stated otherwise, the algorithms for $br_i$ and $mng_i$ run in polynomial time. The instance size is dominated by the sizes of $\mathbf{r}$, all $br_i$ and $kb_i$, and of the underlying atom set $\mathcal{A}$ (which is part of the input).

For reasoning with ordinary runs, we can show:

**Theorem 1** *Deciding $M, \mathbf{r} \models_b C_i(a)$ is* NP*-complete, and* NP*-hard even if $M$ is acyclic, the $br_i$ are* not*-free and without window atoms, and one of the following is not bounded by a constant: (i) $|M|$, (ii) the size of the input streams $S_{ki}$, (iii) the size of the local KBs $kb_i$.*

Informally, we can guess a model $\mathbf{s}$ for $\mathbf{r}$ that witnesses $M, \mathbf{r} \models_b C_i(a)$ and simulate the run from 0 up to $t$ matching $\mathbf{s}$, using the algorithms for $br_i$, $mng_i$ and $\mathbf{ACC}_i$: naturally the streaming time maps to physical time by a (constant) factor. Note that the precise KB formats (propositional, non-ground) is here irrelevant. The NP-hardness is shown by various reductions from SAT. For determined contexts, tractability results.

**Proposition 4** *Deciding $M, \mathbf{r} \models_b C_i(a)$ is in* P*, if $M$ is deterministic, i.e., $|\mathbf{ACC}_i(kb_i)| \leq 1$ holds for all $C_i$ and $kb_i$.*

**Locally stable runs.** We denote by $M, \mathbf{r} \models_{bs} C_i(a)$ the restriction of $M, \mathbf{r} \models_b C_i(a)$ to locally stable models. As computing locally stable runs requires to recognize given acceptable belief sets, we suppose that deciding $BS_i \in \mathbf{ACC}_i(kb_i)$ has complexity in C. We then obtain the following result.

**Theorem 2** *Deciding $M, \mathbf{r} \models_{bs} C_i(a)$ is in $\mathsf{NP}^{\mathsf{NP}^{\mathsf{C}}}$. It is $\mathsf{NP}^{\mathsf{NP}}$-complete for $M$ in which the contexts use (normal) logic programs with supported resp. stable model semantics.*

Intuitively, local stability forces us to know for a correct reset that *no* equilibrium $BS_{\mathcal{C}}$ exists for a SCC $\mathcal{C}$; as this rises depending on the guessed prefix of the run, we resort to an oracle for $\mathsf{coNP}^{\mathsf{C}} = \mathsf{NP}^{\mathsf{C}}$ in the general setting (which proceeds with guess and check for an equilibrium). The $\mathsf{NP}^{\mathsf{NP}}$-hardness is shown by a reduction from QBF solving.

However, the following setting is tractable for $x \in \{b, bs\}$ ("small" is bounded by a constant):

**Theorem 3** *Deciding $M, \mathbf{r} \models_x C_i(a)$ is in* P*, if (i) $|M|$ is small, (ii) the input streams $S_{ki}$ are accessed via small-length windows, (iii) $br_i$, $mng_i$, and $\mathbf{ACC}_i(kb_i)$ are in logspace where $|\mathbf{ACC}_i(kb_i)|$ is polynomially bounded, and (iv) any $kb_i(t')$ results as $mng_i(chg, kb_i(0))$ for some small-size chg.*

Simple simulation does not work here. Informally, we can reduce the problem to reachability in a graph in polynomial time, whose nodes describe possible (adorned) states of a run. Further restrictions in (ii) allow to achieve NLSpace.

In the setting above, we have assumed that some algorithm to compute $BS_i \in \mathbf{ACC}_i(kb_i)$ is available. Alternatively, one could assume that, as for locally stable runs, only an oracle for deciding $BS_i \in \mathbf{ACC}_i(kb_i)$ is available. However, tractability gets lost resp. becomes unknown even for deterministic MCS and a LSpace oracle, as variants of integer factorization can be expressed via $\mathbf{ACC}_i(kb_i)$ such that problems in the class UP that are unknown to be in P can be solved.

### 5.1 Prediction

An important further reasoning task is *prediction*, i.e., to reason about the future states of an sMCS. We may ask whether $M, \mathbf{r}' \models_b C_i(a)$ holds for some extension $\mathbf{r}'$ of $\mathbf{r}$ up to a given time point $t' \geq t$, i.e., $|\mathbf{r}'| = t'+1$ and $\mathbf{r}'|_{\leq t} = \mathbf{r}|_{\leq t}$; we denote this by $M, \mathbf{r} \models_x^{t'} C_i(a)$, for $x \in \{b, bs\}$. Then

**Proposition 5** *Deciding $M, \mathbf{r} \models_{bs}^{t'} C_i(a)$ is in $\mathsf{NExpTime}^{\mathsf{C}}$ and deciding $M, \mathbf{r} \models_b^{t'} C_i(a)$ is* NExpTime*-complete.*

A witnessing model $\mathbf{s}$ for a suitable extension $\mathbf{r}'$ of $\mathbf{r}$ is exponentially bounded in $t'$, which can be guessed and verified. For locally stable runs, an SCC $\mathcal{C}$ has single exponential many candidate equilibria $BS_{\mathcal{C}}$, which can be checked with the C oracle one by one; this yields $\mathsf{NExpTime}^{\mathsf{C}}$ membership. The NExpTime-hardness is via a Turing machine encoding.

Whether $M, \mathbf{r} \models_x^{t'} C_i(a)$ holds for some arbitrary $t' \geq t$, denoted by $M, \mathbf{r} \models_x^{\infty} C_i(a)$, is clearly undecidable. This is because (U1) the local KBs and (U2) the data streams can increase unboundedly; (U3) pathologic window functions

$w(S,t)$ resp. management functions $mng_i(\{\mathsf{op}(t)\}, kb_i)$ can exploit the time argument $t$ to run unbounded computations.

Decidability thus requires tailored restrictions; without specific logics and contexts, the (traditional) generic nature of MCS permits only abstract computational conditions.

The following assumptions are practical: for (U1), the size of each $kb_i(t')$ is polynomial in $|kb_i(0)|$ (this limits storing actual time stamps in $kb_i$); for (U2), the bridge rules use windows to recent input; and for (U3), absolute time $t$ to evaluate window resp. management functions is not crucial.

Capturing the latter is nonobvious, as time should still play a role for evaluating windows and storing data in the $kb_i$. Fortunately, we can identify benign practical conditions.

A window function $w(S,t)$ is *regular*, if (i) $w(S,t)(t')$, i.e., the data in the window $w(S,t)$ at time $t'$, depends only on $S$ from $t', t'{+}1$ etc. onwards, and (ii) for some $l \geq p \geq 0$ polynomial in $|kb_i(0)|$, we have $w(S,t) = w(S', t+p)$ for every time $t$ and streams $S, S'$ that coincide on the past (future) $l$ time points around $t$ resp. $t{+}p$ *having data*. Informally, (i) enables us to drop past data; in (ii), $p$ reflects that $w$ is periodic and $l$ is a limit for evaluation. Time-based but also other common windows (e.g., based on tuples) are regular.

However simply memorizing the data within the limit *with their actual time points* is not feasible under a space constraint wrt. $|kb_i(0)|$. For schematic bridge rules as in the running example, where time reference is confined to the evaluation time ($\boxplus^0 @_Z \top$), or with a fixed offset $os$ to it ($Z{\pm}os$), which we call *plain*, this can be avoided.

**Lemma 1** *If $br_i$ is plain and any window occurring in it is regular, a sufficient fragment of each input stream $S_{ki}$ to evaluate $br_i$ can be maintained in polynomial space.*

We consider each $C_i$ has only timeless operators, schematic operators have only few (constantly many) variables.

**Theorem 4** *Deciding $M, \mathbf{r} \models^{\mu}_{x} C_i(a)$ is* PSpace*-complete for $\mu \in \{t', \infty\}$, if (i) each $kb_i$ always has polynomial size, (ii) context evaluation (i.e., of $br_i$, $mng_i$, $\mathbf{ACC}_i$) is in polynomial space, and (iii) all bridge rules $br_i$ are plain.*

Informally, the problems are in PSpace as a witnessing model s can be stepwise guessed and verified by simulating a run in polynomial space, using windows on the various streams. Equilibrium computation is also feasible in polynomial space. PSpace-hardness is immediate by a Turing machine encoding. We note that the results above are unaffected if one changes from the setting with an algorithm to compute some $BS_i \in \mathbf{ACC}_i(kb_i)$ to an oracle for $BS_i \in \mathbf{ACC}_i(kb_i)$ in PSpace.

## 6 Related Work

As already mentioned, rMCS [Brewka et al., 2014], eMCS [Goncalves et al., 2014] and aMCS [Ellmauthaler and Pührer, 2015] define semantics via sequential application of static logics on stepwise evolving KBs and observations; rMCS and eMCS can model logical time, which synchronously increases if a global equilibrium is reached; hence neither computation nor transfer time is considered, different from the asynchronous model of sMCS. Asynchrony in aMCS is due to peculiar output rules and controllers (abstract components to handle computation and transfer times) which decide when computation starts at a context.

Streaming bridge rules with window operators as first-class citizens equip sMCS with light-weight, dedicated stream reasoning. To emulate this feature, rMCS, eMCS would require non-trivial extensions in either bridge rules or the local logic:

• To do it on the bridge rules, exchanged beliefs first need to be tagged with timestamps, that is, ordinary MCS beliefs of the form $b(\mathbf{c})$ should now be $b(\mathbf{c}, t)$ where $t$ represents a timestamp. Streaming atoms with time-based window operators of the form $(i : \boxplus^k b(\mathbf{c}))$ in sMCS can be emulated by an auxiliary bridge rule:

$$wb(\mathbf{c}) \leftarrow (i : b(\mathbf{c}, t)), (o : now(t_{now})), t_{now} - t \leq k,$$

where $o$ is a special observation that sends out the clock ticks at every time point.

For tuple-based window operators [Beck *et al.*, 2015], a more involved translation with three auxiliary bridge rules are required [Beck *et al.*, 2017].

Moreover, this approach requires rMCS, eMCS to split bridge rules into two kinds: those whose heads are imported to the local KB and those whose heads are just used to emulate the window operators (thus called auxiliary bridge rules). This creates further complication in defining the applicability of bridge rules, and somehow departs from the original idea of bridge rules as a means for quick filtering of neighbor beliefs into contexts.

• Another approach is to keep bridge rules as in rMCS, eMCS, but introduce the window mechanism in the local logic. This works straightforwardly with ASP semantics as one can simply take e.g. contexts with LARS [Beck *et al.*, 2015] as the local logic. However, for other logics such as description logics, an extension with window operators has not been investigated and is non-trivial. In other words, this approach is limited in the sense that not every existing logic can be readily put in such as streaming setting. It also interferes with the local logic of the contexts, which departs from the principle of MCS to keep the setting as abstract as possible.

Still, the capacity of management functions can be exploited to do the stream handling. However, from the implementation point of view this approach pushes all functionalities into the contexts; having window operators at the bridge rule level separates streaming features from reasoning features into different layers of an architecture. This makes implementing and maintaining sMCS easier, and moreover stream management (as needed e.g. in local equilibrium computation, or when contexts are busy) can be done more transparently and effectively.

Finally, we have provided feedback equilibria and local stability, which have no counterpart in related MCS extensions.

Clearly scenarios similar to the running example can be modeled as multi-agent systems. However, one will need to specify a suitable negotiation protocol among agents to achieve an equilibrium. The ensuing communication need and effort can be saved with an sMCS, as obtaining a (local) equilibrium is provided as primitive at a low level in the system.

Ameloot *et al.* [2015] used distributed Datalog under stable model semantics to describe the semantics of distributed systems (e.g., replicated databases) with asynchronous communication of indefinite delay between nodes, which are modeled as stratified datalog programs on fact bases. They gave an operational counterpart and refined the Dedalus approach [Alvaro *et al.*, 2010]. However, neither streaming aspects (e.g., windowing) were an issue, nor heterogeneity (e.g., KB semantics with multiple possible outputs); furthermore, no equilibria were considered. Alternatively one can model (distributed) state transition systems using temporal logics and assess properties via model checking. However, temporal logics are typically propositional; thus efficient modeling of sMCS in them with its features of (arbitrary) window operators, local KBs in expressive logics, asynchronous computation and feedback equilibria is a challenging issue, and remains for future work.

## 7 Conclusion

We have introduced streaming MCS as an extension of managed MCS with window operators as first-class citizens in the bridge rules. sMCS have a run-based semantics that accounts for asynchronous, distributed execution. For this setting, we defined the notion of equilibria, a central and key notion in MCS which has not been investigated in aMCS. We have presented the complexity of reasoning in sMCS: ad-hoc query answering is NP-complete while prediction is PSpace-complete in relevant settings, but undecidable in general.

For future work, we would like to (i) explore other refined notions of feedback equilibria, for example, equilibria that can be constructively computed in an operational way, (ii) study uniformed complexity of reasoning with sMCS, (iii) implement a prototype of sMCS based on the DMCS system [Dao-Tran *et al.*, 2015] for MCS evaluation, and (iv) investigate the transferability of some notions (computation/transfer time, feedback equilibria) into the aMCS framework.

## Acknowledgment

We are grateful to the reviewers for their comments, which helped to improve this paper.

## References

[Alvaro *et al.*, 2010] Peter Alvaro, William R. Marczak, Neil Conway, Joseph M. Hellerstein, David Maier, and Russell Sears. Dedalus: Datalog in time and space. In *Datalog Reloaded - First International Workshop, Datalog 2010, Oxford, UK, March 16-19, 2010. Revised Selected Papers*, LNCS, pages 262–281, 2010.

[Ameloot *et al.*, 2015] Tom J. Ameloot, Jan Van Den Bussche, William R. Marczak, Peter Alvaro, and Joseph M. Hellerstein. Putting logic-based distributed systems on stable grounds. *TPLP*, FirstView:1–40, 12 2015.

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

[Beck *et al.*, 2015] Harald Beck, Minh Dao-Tran, Thomas Eiter, and Michael Fink. LARS: A Logic-based Framework for Analyzing Reasoning over Streams. In *AAAI*, 2015.

[Beck *et al.*, 2017] Harald Beck, Thomas Eiter, and Christian Folie. Ticker: A System for Incremental ASP-based Stream Reasoning. Manuscript, 2017.

[Bikakis and Antoniou, 2010] Antonis Bikakis and Grigoris Antoniou. Defeasible contextual reasoning with arguments in ambient intelligence. *IEEE Transactions on Knowledge and Data Engineering*, 22(11):1492–1506, 2010.

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

[Brewka *et al.*, 2007] Gerhard Brewka, Floris Roelofsen, and Luciano Serafini. Contextual default reasoning. In *International Joint Conference on Artificial Intelligence (IJCAI'07)*, pages 268–273, 2007.

[Brewka *et al.*, 2011] Gerhard Brewka, Thomas Eiter, Michael Fink, and Antonius Weinzierl. Managed Multi-Context Systems. In *IJCAI*, pages 786–791, 2011.

[Brewka *et al.*, 2014] Gerhard Brewka, Stefan Ellmauthaler, and Jörg Pührer. Multi-Context Systems for Reactive Reasoning in Dynamic Environments. In *ECAI*, pages 159–164, 2014.

[Dao-Tran *et al.*, 2015] Minh Dao-Tran, Thomas Eiter, Michael Fink, and Thomas Krennwallner. Distributed Evaluation of Nonmonotonic Multi-Context Systems. *Journal of Artificial Intelligence Research*, 52:543–600, 2015.

[Della Valle *et al.*, 2009] 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.

[Ellmauthaler and Pührer, 2015] 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*, pages 141–156, 2015.

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

[Giunchiglia and Serafini, 1994] Fausto Giunchiglia and Luciano Serafini. Multilanguage hierarchical logics or: How we can do without modal logics. *Artificial Intelligence*, 65(1):29–70, 1994.

[Gonçalves *et al.*, 2014] Ricardo Gonçalves, Matthias Knorr, and João Leite. Evolving Multi-Context Systems. In *ECAI*, pages 375–380, 2014.

[Heintz *et al.*, 2010] Fredrik Heintz, Jonas Kvarnström, and Patrick Doherty. Stream-Based Reasoning in DyKnow. In *Cognitive Robotics, 21.02. - 26.02.2010*, volume 10081 of *Dagstuhl Seminar Proceedings*, 2010.

[McCarthy, 1993] John McCarthy. Notes on formalizing context. In *International Joint Conference on Artificial Intelligence (IJCAI'93)*, pages 555–562, 1993.