Stream Reasoning and Multi-Context Systems

Thomas Eiter

Institute of Logic and Computation
Vienna University of Technology (TU Wien)

joint work with M. Dao-Tran¹, A. Falkner³, P. Ogris², K. Schekotihin²,
    P. Schneider¹,³, P. Schüller¹, A. Weinzierl¹

Stream Reasoning Workshop 2019,
Linköping, Sweden, April 16-17, 2019

---

Austrian Science Fund (FWF) grant P26471
Austrian Research Promotion Agency (FFG), 588655
Outline

1. Multi-Context Systems

2. MCS and Data Streams

3. MCS for Smart Cyber-Physical Systems

4. DynaCon: Dynamic Configuration

5. Conclusion
Multi-Context Systems

- **Contextual Reasoning**: model information interlinkage of knowledge bases / agents
  - information flow between KBs via *bridge rules*
    
    *Mr. 1*: \( \text{row}(X) \leftarrow (\text{Mr. 2}: \text{sees}_\text{row}(X)) \)
    
    *Mr. 2*: \( \text{col}(Y) \leftarrow (\text{Mr. 1}: \text{sees}_\text{col}(Y)) \)

  - equilibrium ensures aligned information

- Different early varieties
  - Trento School (Giunchiglia, Serafini et al.):
    - Heterogeneous MCS [Giunchiglia and Serafini, 1994]
    - Nonmonotonic bridge rules [Roelofsen and Serafini, 2005]
    - Extension to Contextual Default Logic [Brewka et al., 2007]
  - *nonmonotonic multi-context systems (MCS)* [Brewka and E_, 2007]
  - *managed MCS (mMCS)* [Brewka et al., 2011]
Nonmonotonic Multi-Context Systems (MCS)

**Multi-Context System**

Formally, a Multi-Context System

\[ M = (C_1, \ldots, C_n) \]

consists of contexts

\[ C_i = (L_i, kb_i, br_i), \ i \in \{1, \ldots, n\}, \]

where

- each \( L_i \) is a “logic,”
- each \( kb_i \) is a knowledge base in \( L_i \), and
- each \( br_i \) is a set of \( L_i \)-bridge rules over \( M \)’s logics.
Logic

A logic $L$ is a tuple $L = (KB_L, BS_L, ACC_L)$, where

- $KB_L$ is a set of well-formed knowledge bases, each being a set (of “formulas”)
- $BS_L$ is a set of possible belief sets, each being a set (of “formulas”)
- $ACC_L : KB_L \rightarrow 2^{BS_L}$ assigns each KB a set of acceptable belief sets

Thus, logic $L$ caters for multiple extensions of a knowledge base.

Bridge Rules

A $L_i$-bridge rule over logics $L_1, \ldots, L_n$, $1 \leq i \leq n$, is of the form

$$s \leftarrow (r_1 : p_1), \ldots, (r_j : p_j), \text{not}(r_{j+1} : p_{j+1}), \ldots, \text{not}(r_m : p_m)$$

where $kb \cup \{s\} \in KB_i$ for each $kb \in KB_i$, each $r_k \in \{1, \ldots, n\}$, and each $p_k$ is in some belief set of $L_{r_k}$.

Note: such rules are akin to rules of normal logic programs
Example (Authors)

Suppose a MCS $M = (C_1, C_2)$ has contexts that express the individual views of a paper by the two authors.

- $C_1$:
  - $L_1 = \text{Classical Logic}$
  - $kb_1 = \{ \text{unhappy} \supset \text{revision} \}$
  - $br_1 = \{ \text{unhappy} \leftarrow (2 : \text{work}) \}$

- $C_2$:
  - $L_2 = \text{Reiter's Default Logic}$
  - $kb_2 = \{ \text{good} : \text{accepted/accepted} \}$
  - $br_2 = \{ \text{work} \leftarrow (1 : \text{revision}), \text{good} \leftarrow \text{not}(1 : \text{unhappy}) \}$
Equilibrium Semantics

- **Belief State**

  A belief state is a sequence $S = (S_1, \ldots, S_n)$ of belief sets $S_i$ in $L_i$

- **Applicable Bridge Rules**

  For $M = (C_1, \ldots, C_n)$ and belief state $S = (S_1, \ldots, S_n)$, the bridge rule
  
  $s \leftarrow (r_1 : p_1), \ldots, (r_j : p_j), not(r_{j+1} : p_{j+1}), \ldots, not(r_m : p_m)$
  
  is applicable in $S$ if (1) $p_i \in S_{r_i}$, for $1 \leq i \leq j$, and (2) $p_k \not\in S_{r_k}$, for $j < k \leq m$.

- **Equilibrium**

  A belief state $S = (S_1, \ldots, S_n)$ of $M$ is an equilibrium iff for all $i = 1, \ldots, n$,
  
  $S_i \in ACC_i(kb_i \cup \{head(r) \mid r \in br_i \text{ is applicable in } S\})$. 
Equilibrium Semantics, cont’d

Example, cont’d

Reconsider $M = (C_1, C_2)$:

- $kb_1 = \{ \text{unhappy} \vdash \text{revision} \}$ (Classical Logic)
- $br_1 = \{ \text{unhappy} \leftarrow (2: \text{work}) \}$

- $kb_2 = \{ \text{good} : \text{accepted/accepted} \}$ (Default Logic)
- $br_2 = \{ \text{work} \leftarrow (1: \text{revision})$, \text{good} \leftarrow \text{not}(1: \text{unhappy}) \}$

$M$ has two equilibria:

- $E_1 = (Th(\{\text{unhappy, revision}\}), Th(\{\text{work}\}))$ and
- $E_2 = (Th(\{\text{unhappy} \vdash \text{revision}\}), Th(\{\text{good, accepted}\}))$
Managed MCS

- MCS: pure information alignment, fully static
- introduce context manager, to update/change the KB
  - Bridge rules:
    \[
    op(f) \leftarrow (c_1 : p_1), \ldots, (c_j : p_j), \text{not}(c_{j+1} : p_{j+1}), \ldots, \text{not}(c_m : p_m). \]
  - management function \( mng : 2^{F_{LS}} \times KB_{LS} \rightarrow 2^{(KB_{LS} \times ACC_{LS}) \setminus \{\emptyset\}} \)
    assigns updates commands + KB a follow-up KB + evaluation semantics

- managed context \( C_i = (LS_i, kb_i, br_i, OP_i, mng_i) \) with
  - \( LS_i = (BS_{LS_i}, KB_{LS_i}, ACC_{LS_i}) \) a logic suite,
  - \( kb_i \in KB_{LS_i} \) a knowledge base,
  - \( br_i \) a set of bridge rules for \( C_i \),
  - \( OP_i \) a management base (commands), and
  - \( mng_i \) a management function over \( LS_i \) and \( OP_i \).

- Managed Multi-Context System (mMCS) \( M = (C_1, \ldots, C_n) \) are stateful, form the basis of other MCS (eMCs, rMCS, aMCS, sMCS, dMCS, tMCS)
Managed MCS, cont’d

Example (Diseases)

- $C_1$: relational database on disease treatments
  
  $kb_1 = \{ \text{treat}(\text{pen}, \text{str}_\text{pneu}, \text{pneu}, \text{evd}), \text{treat}(\text{azith}, \text{leg}_\text{pneu}, \text{leg}, \text{evd}), \text{ineff}(\text{pen}, \text{leg}_\text{pneu})\}$

  conclude likely effects using $C_2$.

  $br_1 = \{ \text{treat}(X, B, I, \text{likely}) \leftarrow (1: \text{treat}(X, B, _, _)), (2: B \text{ rdf:causes } I). \}$

- $C_2$: RDF-triple store on disease causations.
  
  $kb_2 = \{ \text{str}_\text{pneu} \text{ rdf:causes } \text{men}, \text{leg}_\text{pneu} \text{ rdf:causes } \text{atyp}_\text{pneu} \}$

- $C_3$: bacteria ontology (DL)

- $C_4$: generalized logic program deriving possible medication effects:

  $br_4 = \{ \text{add}(\text{isa}(X, Y)) \leftarrow (3: (X \sqsubseteq Y)).$
  
  $\text{add}(\text{eff}(X, B)) \leftarrow (1: \text{eff}(X, B)).$
  
  $\text{upd}(\text{not eff}(X, B)) \leftarrow (1: \text{ineff}(X, B)). \}$

Semantics

- Applicable bridge rule heads: $app_i(S) = \{ \text{hd}(r) \mid r \in br_i \land S \models body(r) \}$.

- Equilibrium: $S = (S_1, \ldots, S_n)$ iff for every $1 \leq i \leq n$ some $\text{(}kb_i', ACC_{LS_i}\text{)} \in mng_i(app_i(S), kb_i)$ exists s.t. $S_i \in ACC_{LS_i}(kb_i')$. 
1. Multi-Context Systems

2. MCS and Data Streams

3. MCS for Smart Cyber-Physical Systems

4. DynaCon: Dynamic Configuration

5. Conclusion
Streaming World

- Sensors, networks, mobile devices:
  - getting to a connected world...
- Pushing rather than pulling of data
- Dynamic streams of data, potentially infinite
  - low frequency changes (meter reading)
  - high frequency changes (stock trading)
- Continuous computation / evaluation
  - synchronous vs. asynchronous
- Reference to time
- **Poses challenges to MCSs**
Example: Cooperative Robots

- In a mall, robots must deliver packages to destinations
- $R_A$ must deliver package $P_1$ (at 9) to destination $D_1$ (7)
- $R_B$ must deliver package $P_2$ (at 4) to destination $D_2$ (1)
- Minimize travel distance: agree to pick up other package and exchange (e.g. at node 5)
- Agreement may be challenging: robots already move, connections turn out unusable (too many people around), ...

**Setting**: *dynamic monitoring* of usability

sensors for position, occupation etc.
MCS Features

- **(static) MCS, mMCS**: have an equilibrium (fixpoint) semantics

- **(dynamic) reactive MCS (rMCS)** [Brewka et al., 2014, 2018],
  evolving MCS (eMCS) [Gonçalves et al., 2014]:
  - computing equilibria is timeless

- **(dynamic) asynchronous MCS (aMCS)** [Ellmauthaler and Pührer, 2015]:
  - physical computation time, transfer time are disregarded
  - no baseline mechanism to achieve equilibrium

- **streaming MCS (sMCS)** [Dao-Tran and E_, 2017]:
  - bridge rules with *window atoms* (simple LARS formulas [Beck et al., 2018]) to access input streams
  - model *computation time* and *data transfer time*
  - internal *asynchronous execution control* (restart/wait on eval requests)
  - run-based semantics, with *feedback equilibria* to enforce *local stability* in runs (avoid infinite loops, and generalize rMCS, eMCS)

  additional stream reasoning inside contexts possible!
sMCS by Example (cont’d)

- sensor context \( C_i \), \( 4 \leq i \leq 6 \) feeds sensor input to \( C_{i-3} \)
- \( C_4 \) (\( C_5 \)) tells position \( pos(X, Y, L) \) of \( RA \) (\( RB \)) on \( X \rightarrow Y \), \( L \in \{0\% \ldots , 100\%\} \)
- \( C_3 \) gets sensor data of \( C_6 \), infers blocked connections and sends this info to \( C_1 \), \( C_2 \)
- \( C_1 \) (\( C_2 \)): shortest route for \( RA \) (\( RB \)) to \( D_1 \) (\( D_2 \)), with blocked connections, meeting point \( m(X) \)

**bridge rules**: *window atoms* \((6: \square^{\text{8}} \nabla n(X, Y, N)), (3: \square^{\text{3}} \nabla block(X, Y))\)

- \( \text{update}(cr(X, Y)) \leftarrow (6: \square^{\text{8}} \nabla n(X, Y, N)), N > 10 \) accesses \( C_6 \)'s output
  - “store link \( X \rightarrow Y \) is crowded, if \( C_6 \) reported in the last 8 mins always 10 people on it.”
- \( \text{remove}(block(X, Y)) \leftarrow \text{not} (3: \square^{\text{3}} \nabla block(X, Y)), (2: block(X, Y)) \)
  - “unblock link \( X \rightarrow Y \) unblocked, if \( C_3 \) didn’t report it the last 3 mins always blocked.”
Run-based Semantics

- A state of $C_i$ is a triple $s_i=(s_i, o_i, kb_i)$ where
  - $s_i \subseteq \{IE, SE\}$ is the execution status (intend to execute / start to execute)
  - $o_i \subseteq Bel_i$ is the output belief set streamed to other contexts, unless $o_i = \epsilon$;
  - $kb_i$ is the local KB (which can evolve).

- runs are constrained state sequences $s = s(0), \ldots, s(t)$ of global states
  $s(t') = (s_1(t'), \ldots, s_n(t'))$, where each $s_i(t')$ is state of $C_i$:
  - delay intention $IE$ to actual start $SE$ (busy) or restart
  - respect data transfer time $\Delta_{ki}$, computation time $f_i(br_i, kb_i)$

Example (run trace) (focus on $C_1, C_2$; $ISE_{1,2} = \{IE_1, SE_1, IE_2, SE_2\}$, $ISE_2 = \{IE_2, SE_2\}$)

- sensors $C_4, C_5$ stream at $5k \geq 0$, $C_6$ streams continuously
- $C_1, C_2, C_3$ run in pushing mode; $C_1$ ignores new input when busy; $C_2, C_3$ restart.
- $\Delta_{12} = \Delta_{31} = 1$ 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 $kb_i$
- $C_3$ sends block$(5, 6)$ at $t = 10$ (ignored by $C_1$ at 11)
Idealized Runs

- **Aim:** capture rMCS and eMCS which feature step-wise equilibrium computation at zero cost (= infinite speed)

- **Naive Approach:** set in runs $\Delta_{ki} = f_i = 0$
  - does not work: local KBs $kb_i$ could not change (but rMCS/eMCS runs are stateful)

- **Way out:** extend time ontology with *infinitesimally small chronon* $\varepsilon$
  - computation time is $\varepsilon$
  - $t < t + \varepsilon$ and $t + \varepsilon = t + k\varepsilon$ for each $k \in \mathbb{N} > 0$
  - transfer time $f_i = 0$

- In an *idealized run* $s = (s_0, \ldots, s_{\text{end}})$,
  
  
  $o_i(t+1) = o_i(t+\varepsilon)$ and $kb_i(t+1) = kb_i(t+\varepsilon)$,

  where
  - $o_i(t + \varepsilon) = \text{ACC}(kb_i(t + \varepsilon))$ and
  - $kb_i(t + \varepsilon) = \text{mng}_i(app^\varepsilon_i(s, t), kb_i(t))$

  **Emulation Property:** runs of an rMCS $M$ (resp. eMCS $M$ from a natural class) can be emulated by idealized runs of a corresponding sMCS $M^s$
Feedback Equilibria

- Total asynchrony can be uncomfortable if contexts depend on each other.
- Suppose also $C_2$ suggests a meeting point, imported by $C_1$ into $kb_1$ via
  \[ \text{update}(m_2(X)) \leftarrow (2:m(X)) \]
- Meeting mismatch: do a further round (e.g., follow other proposal).
- Assume $f_1(br_1, kb_1) = 2$, $f_2(br_2, kb_2) = 3$ for all $kb_i$, $\Delta_{12} = 1$, other costs=0:

  \[
  \begin{array}{cccccc}
  0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
  ISE_{1,2} & \{m(5)\}_1 & ISE_{1,2} & \{m(6)\}_2 & ISE_{1,2} & \{m(5)\}_2 & ISE_{1,2} & \{m(6)\}_2 \\
  \end{array}
  \]

*computing an agreed meeting point loops indefinitely*
Feedback Equilibria, cont’d

Key ideas:
- consider strongly connected components (SCCs) via an import graph ($C_i \rightarrow C_j$, if $(j:A)$ occurs in $br_i$)
- for equilibrium computation, dispense streaming data from outside
- any $C_i$ can request, while computing, at time $t$ stability of its SCC $C_i$:
  - the contexts in $C_i$ are restarted with input at $t_{exe}$
  - at some $t'' \geq t$, either $C_i$ reports an equilibrium, or $C_i$ restarts its contexts with input at time $t''$ if no equilibrium exists

Feedback Equilibrium of $C_i$ wrt. run $s$ at time $t$: for every $C_{ij} \in C_i$,
belief set $BS_{ij} \in ACC_{ij}(mng_{ij}(app^e_{ij}(s, t), kb_{ij}(t)))$

- intuitively, run in idealized mode
- input to $C_i$ is frozen at $t$
- inside $C_i$, cyclic information flow is respected
Locally Stable Runs

- \( \rho \) denotes a stability request, \( EQ \) is a new status

- Informally a *locally stable run* for \( C \) is a run \( s = (s_0, \ldots, s_{t_{\text{end}}}) \), s.t.
  - requests for local stability of \( C \) at time \( t \) are granted at \( t' \geq t \);
  - to serve a request, all \( C \in C_i \) switch to equilibrium computation from \( t \);
  - a feedback eq. is returned at output time \( t_{\text{out}} \) if one exists, else \( C \) is restarted.

- A run \( s \) is *locally stable*, if it is locally stable for each SCC

Example

\[
\begin{array}{cccccccc}
  & & & & & & & \\
  & ISE_{1,2} & \{m(5)\}_1 & ISE_{1,2} & EQ_{1,2} & EQ_{1,2} & EQ_{1,2} & \{m(6)\}_1 \\
 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\end{array}
\]

- \( C_1 \) realizes at \( t = 3 \) that \( C_2 \)'s suggestion does not match his of \( t = 2 \).
- \( C_1 \) requests local stabilization for \( C_1 = \{C_1, C_2\} \)
- meeting at node 6 yields an equilibrium at time 7
### Reasoning

**Setting:** sMCS $M = (C_1, \ldots, C_n)$
- sensor contexts $O = C_j, C_{j+1}, \ldots, C_n$
- reading $r = r(0), \ldots, r(t)$: sensor data stream
- periodic decision to execute, permanent ignore/restart
- algs to evaluate $br_i, mng_i, ACC_i$ (compute some $BS_i \in ACC_i(kb_i)$)

**Monitoring:** watch the past
- $M, r \models_b C_i(a)$: $a$ is believed at $C_i$ at time $t$ in some run $s = (s_0, \ldots, s_t)$ for $r$
- $M, r \models_{bs} C_i(a)$: \ldots some locally stable run \ldots

**Prediction:** consider future data
- $M, r \models_x^{t'} C_i(a)$: does $M, r' \models_x C_i(a)$ for an extension $r'$ of $r$ up to $t'$?
- $M, r \models_{\infty} C_i(a)$: does $M, r \models_x^{t'} C_i(a)$ for some $t'$?
Reasoning: Complexity

- **Monitoring** and **Prediction** with bounded horizon \((M, r \models^t_x C_i(a))\) is decidable, but intractable in general

- **Unbounded Prediction** \((M, r \models_\infty C_i(a))\) is undecidable in general, due to
  - (U1) \(kbi\), (U2) unbounded streams, (U3) pathologic window evaluation

- **Prediction** is in PSpace, if
  - each \(kbi\) remains in *polynomial size*,
  - context evaluation \((br_i, mng_i, ACC_i)\) feasible in *polynomial space*,
  - the bridge rules \(br_i\) are *plain* (roughly, time references at eval time are fixed offsets) and all windows are *regular* (e.g., small time/tuple-based windows)
    \[\Rightarrow\] streams manageable in pol. space (intuitively, recent input)

- Tractability (moreover logspace feasibility) needs severe restrictions

- Links to work on streams in databases [Gurevich *et al.*, 2007] and ontologies [Özçep, 2017], and recent work at Oxford and Linköping
1. Multi-Context Systems

2. MCS and Data Streams

3. MCS for Smart Cyber-Physical Systems

4. DynaCon: Dynamic Configuration

5. Conclusion
MCS for Smart Cyber-Physical Systems (CPS)

- Distributed systems with lots of sensors
- Possible embedding in the Internet of Everything
- “Friendly & Kind” (F&K) systems, e.g. in e-health
- Bridge rules as vital elements for knowledge exchange
- mMCS are nicely abstract, but limitations require extensions; cf. [Costantini and Gasperis, 2016], [Cabalar et al., 2017]

Proposals:

- **dynamic mMCS (dMCS)** [Costantini and Gasperis, 2016], [Dao-Tran et al., 2011]: link at runtime
- **timed mMCS (tMCS)** [Cabalar et al., 2017]: update state prior to bridge rule evaluation
MCS for Smart Cyberphysical Systems, cont’d

MCS Rationale

- stay at the abstract level
- use MCS more as *modeling tool*
  - heterogeneous components as contexts
  - interlinkage and exchange
- also useful for simulation

Scenario: Dynamic Configuration

- go beyond monitoring / prediction
- dynamically change / adapt the behavior of components
Content-Centric Networks

- The content in the network is addressed by “name” – physical location is irrelevant
- **Content-Centric Routers (CCR)** can route interest packages, cache and adapt media chunks in highly dynamic conditions
- Cache sizes are limited – efficient caching strategies needed

Example

- **Factor: Current daytime**
  - **Morning**: few users interested in different media
  - **Evening**: many 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 a random recent chunk
  - **LFU**: a new chunk replaces the *Least Frequently Used* chunk
**Extended CCR – Intelligent Caching Agent**

- **Legacy components of a CCR:**
  - networking unit: implements network interfaces
  - controller: manages content adaptation, routing and caching

- **Extended architecture:**
  - **KB system:** choose controller’s decision making strategy
  - **desired:** human-readable KR language for admin actions
CCR Administration Problem

- **LARS Encoding**

  \[\text{high} \leftarrow \text{value}(V), \bigotimes^k \text{sec} @ T \alpha (V), 18 \leq V.\]

  \[\text{mid} \leftarrow \text{value}(V), \bigotimes^k \text{sec} @ T \alpha (V), 12 \leq V < 18.\]

  \[\text{low} \leftarrow \text{value}(V), \bigotimes^k \text{sec} @ T \alpha (V), V \leq 12.\]

  \[\text{lfu} \leftarrow \bigotimes^k \text{sec} \square \text{high}.\]

  \[\text{lru} \leftarrow \bigotimes^k \text{sec} \square \text{mid}.\]

  \[\text{fifo} \leftarrow \bigotimes^k \text{sec} \square \text{low}, \bigotimes^k \text{sec} \Diamond \text{rtm50}.\]

  \[\text{done} \leftarrow \text{lfu} \lor \text{lru} \lor \text{fifo}.\]

  \[\text{random} \leftarrow \neg \text{done}.\]

- **A simple prototype, using ndnSIM (a general network simulator) and solver.hex (implements a LARS fragment usng dlvhex (hybrid ASP) was done**

- **Later, LARS Ticker [Beck et al., 2017] encodings (for \(k = 3\):**

  \[\text{high} \leftarrow \text{value}(V), \alpha (V) \text{ at } T \ [3 \text{ sec}], 18 \leq V.\]

  \[\text{mid} \leftarrow \text{value}(V), \alpha (V) \text{ at } T \ [3 \text{ sec}], 12 \leq V, V < 18.\]

  \[\text{low} \leftarrow \text{value}(V), \alpha (V) \text{ at } T \ [3 \text{ sec}], V \leq 12.\]

  \[\text{lfu} \leftarrow \text{high} \text{ always} \ [3 \text{ sec}].\]

  \[\text{lru} \leftarrow \text{mid} \text{ always} \ [3 \text{ sec}].\]

  \[\text{fifo} \leftarrow \text{low} \text{ always} \ [3 \text{ sec}], \text{rtm50} \ [3 \text{ sec}].\]

  \[\text{done} \leftarrow \text{lfu}.\]

  \[\text{done} \leftarrow \text{lru}.\]

  \[\text{done} \leftarrow \text{fifo}.\]

  \[\text{random} \leftarrow \neg \text{done}.\]

  \[\text{value} (5). \ \text{value} (15). \ \text{value} (25).\]
1. Multi-Context Systems

2. MCS and Data Streams

3. MCS for Smart Cyber-Physical Systems

4. DynaCon: Dynamic Configuration

5. Conclusion
DynaCon: Dynamic Knowledge-Based (Re)configuration of Cyber-Physical Systems

Use Cases

- Traffic control (Siemens)
- Power distribution grid (Kelag)
- Network threat mgmt (Net4You)
- Rail transportation mgmt (LTE)
DynaCon: Dynamic Knowledge-Based (Re)configuration of Cyber-Physical Systems

Use Cases

- traffic control (Siemens)
- power distribution grid (Kelag)
- network threat mgmnt (Net4You)
- rail transportation mgmnt (LTE)

Idea

Cyber-physical system (CPS)

Streams of (re)configuration descriptions of neighbouring DynaCons

Stream of sensor information

(Re)configuration commands

DynaCon

Stream reasoner
Knowledge-base
Reasoning engine

ASP (re)configurator
Knowledge-base
Reasoning engine

CPS configuration controller

Streams of (re)configuration descriptions

Triggers & (re)configuration inputs
DynaCon: Dynamic Knowledge-Based (Re)configuration of Cyber-Physical Systems

Use Cases

- traffic control (Siemens)
- power distribution grid (Kelag)
- network threat mgmnt (Net4You)
- rail transportation mgmnt (LTE)

Idea

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

[Sarkar, 2011]
Dynamic Configuration as MCS

**Observation:** the MCS framework is suited to model dynamic configuration scenarios

- structuring into interlinked components, evolving over time
- modeling interlinkage through bridge rules
- logical *separation of concerns (SoC) / tasks*

  - **Producers:** contexts that produce information / output
    E.g. sensors can be viewed as such
  - **Monitors:** contexts that observe and aggregate data streams from producers, and report (feed information) to configurators
  - **Configurators:** contexts calculating the setup thru re-configuring the CPS; may involve complex decision component, richer high level stream reasoning
  - **Actuators:** contexts that change the setup in the CPS environment according to the output of the configurators

SoC may be weakened (integrate actuators into producers)
Scenario: Cooperative Intelligent Transportation Systems

- **Infrastructure as a CPS:**
  - communication via V2X
  - roadside units (RSU) at intersections
  - traffic participants are mobile sensors
  - central traffic control center (TCS) is connected to all RSUs

- **Producers:**
  - vehicles send their status
  - traffic lights send signal phases

- **Monitors:**
  - stream aggr./event detection on RSUs
  - high speed + volume sensor streams
  - local view of traffic

- **Configurators:**
  - configurator is in the TCS
  - optimize traffic flow via dynamic configurating of the traffic lights
  - global view of traffic

- **Actuators:**
  - on board of RSU
Component Interface

- **Monitor’s concern:**

  *Making the (variable data rate) input from the CPS accessible to the configurator by
  (i) detecting event, (ii) discretizing and accumulating data streams, (iii) sending the
  results via channels with limited data rate.*

- **Configurator’s Information Channels**

  - **sending information:**
    - *Command Channel*
    - *Information Request Channel*

  - **receiving information**
    - *Event Channel*
    - *Accumulated Process Information Channel*

- **Separate: Configuration Channel**

  - includes *adaptive monitoring*
Monitor vs. Configurator: Interface, cont’d

**Event representation**

- messages $m_e = (e, a, t, l, d, p)$,
- datalog encoding
  
  event(eventType, sourceID, targetID, locationID, time, parameter).

**Process information messages**

- messages with tuples $m_p = (i, a, t, l, p, u)$, $p = (d, v)$
- datalog encoding
  
  information(infoType, sourceID, targetID, locationID, time, value, unit).

**Commands**

- Set parameter (Parameter, Value)
- Get parameter (Parameter)
- Reset
- Activate/deactivate rules or queries
- Update knowledge base (Update Operation)

  datalog encodings
  
  - command(reset, sourceID, targetID).
  - command(setParameter, sourceID, targetID, parameterID, <filter>, value).
Refined DynaCon Architecture

Legend:
- Use Case Specific
- Generic
- Optional

Domain Model

Vocabulary

Configuration Channel

Fog Environment

Cloud Environment

User Action

User

Fog Request Channel

Process Information Channel

Information Request Channel

Command Channel

Decision Module

Controller Module

(i)

(ii)

Memory

Controller

Module

Fog Request Channel

Event Channel

Stream Reasoner B

Bridge B

IF

Information Request Channel

Command Channel

Fog Request Channel

Stream Reasoner A

Bridge A

IF

Stream Reasoner B

Bridge B

IF

Fog Request Channel

Stream Reasoner A

Bridge A

IF

Legend:
- Use Case Specific
- Generic
- Optional

IF

IF

IF

IF

IF

IF

IF

IF

IF

IF

IF 2

(i)

(ii)
Distributed Stream Processing

- LARS engines Ticker [Beck et al., 2017], Laser [Bazoobandi et al., 2017]: monolithic evaluation using a clock (ticks)
- performance issues under load
- as in stream processing, distribute computation

Distributed LARS (Outline):

- streaming atoms: $a \mid \tau, a \mid \bullet \tau, a \mid \circ \! a \mid \Box \! a$
  
  cast *time-point* to *interval* semantics (support *triggers*)

- decompose program $P$ using an (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

\[
\begin{align*}
\text{high} & : \text{value}(V), \alpha(V) \text{ at } T \text{ in [3 s]}, 18 \leq (V). \\
\text{mid} & : \text{value}(V), \alpha(V) \text{ at } T \text{ in [3 s]}, 12 \leq (V), (V) < 18. \\
\text{low} & : \text{value}(V), \alpha(V) \text{ at } T \text{ in [3 s]}, (V) \leq 12. \\
\text{value}(5). \\
\text{value}(15). \\
\text{value}(25). \\
\text{lfu} & : \text{high always in [3 s]}. \\
\text{lru} & : \text{mid always in [3 s]}. \\
\text{fifo} & : \text{low always in [3 s]}, \text{rtm50 in [3 s]}. \\
\text{done} & : \text{lfu}. \\
\text{done} & : \text{lru}. \\
\text{done} & : \text{fifo}. \\
\text{random} & : \text{not done}. \\
\text{finish} & : \text{off in [1 s]}, \text{done}. \\
\text{finish} & : \text{off in [1 s]}, \text{random}. \\
\end{align*}
\]
Distributed Stream Reasoning System

Master: computes the component graph and spawns nodes in the network
1. Multi-Context Systems

2. MCS and Data Streams

3. MCS for Smart Cyber-Physical Systems

4. DynaCon: Dynamic Configuration

5. Conclusion
Conclusion

**Summary**

- MCS as versatile formalism, many extensions
- Streaming data as an increasing computation setting
  - cf. [Ellmauthaler, 2018] for MCS and streaming
- Cyber-Physical Systems (CPS) as application area of MCS
- DynaCon: dynamic configuration, a challenging need in CPS
- Distributed stream reasoning: LARS
  - BigSR, Strider [Ren, 2018]: hybrid adaptive distributed RSP engine, compile into Apache Spark / Flink

**Issues and Ongoing/Future Work**

- Picture the role of MCS
- Refined complexity
  - Communication, memory, parallelization
- Component interface languages
- Adaptive monitoring: control language
- Develop distributed LARS
Chitta Baral and V. S. Subrahmanian.  
Stable and extension class theory for logic programs and default logics.  

Hamid R. Bazoobandi, Harald Beck, and Jacopo Urbani.  
Expressive stream reasoning with laser.  

Harald Beck, Minh Dao-Tran, Thomas Eiter, and Michael Fink.  
Towards a logic-based framework for analyzing stream reasoning.  
Online http://ceur-ws.org/Vol-1303/.

Harald Beck, Minh Dao-Tran, and Thomas Eiter.  
Answer update for rule-based stream reasoning.  

Harald Beck, Minh Dao-Tran, Thomas Eiter, and Michael Fink.  
LARS: A logic-based framework for analyzing reasoning over streams.  
Harald Beck, Thomas Eiter, and Christian Folie.  
Ticker: A system for incremental asp-based stream reasoning.  

Harald Beck, Minh Dao-Tran, and Thomas Eiter.  
LARS: A logic-based framework for analytic reasoning over streams.  

Gerhard Brewka and Thomas Eiter.  
Equilibria in heterogeneous nonmonotonic multi-context systems.  

G. Brewka, F. Roelofsen, and L. Serafini.  
Contextual default reasoning.  
In International Joint Conference on Artificial Intelligence (IJCAI 07), 2007.

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

Gerhard Brewka, Stefan Ellmuthaler, and Jörg Pührer.  
Multi-Context Systems for Reactive Reasoning in Dynamic Environments.  
In ECAI, pages 159–164, 2014.

Gerhard Brewka, Stefan Ellmuthaler, Ricardo Gonçalves, Matthias Knorr, João Leite, and Jörg Pührer.  
Reactive multi-context systems: Heterogeneous reasoning in dynamic environments.  
Pedro Cabalar, Stefania Costantini, and Andrea Formisano.
Multi-context systems: Dynamics and evolution.

Stefania Costantini and Giovanni De Gasperis.
Bridge rules for reasoning in component-based heterogeneous environments.

Minh Dao-Tran and Thomas Eiter.
Streaming multi-context systems.

Minh Dao-Tran, Thomas Eiter, Michael Fink, and Thomas Krennwallner.
Dynamic distributed nonmonotonic multi-context systems.

Thomas Eiter, Mustafa Mehuljic, Christoph Redl, and Peter Schüller.
User guide: dlvhex 2.x.
Thomas Eiter, Tobias Kaminski, Christoph Redl, Peter Schüller, and Antonius Weinzierl. 
Answer set programming with external source access. 

Stefan Ellmauthaler and Jörg Pührer. 
Asynchronous multi-context systems. 

Stefan Ellmauthaler. 
Multi-Context Reasoning in Continuous Data-Flow Environments. 
http://nbn-resolving.de/urn:nbn:de:bsz:15-qucosa2-214577.

Chiara Ghidini and Fausto Giunchiglia. 
Local models semantics, or contextual reasoning = locality + compatibility. 

F. Giunchiglia and L. Serafini. 
Multilanguage hierarchical logics, or: How we can do without modal logics. 

Ricardo Gonçalves, Matthias Knorr, and João Leite. 
Evolving Multi-Context Systems. 
Yuri Gurevich, Dirk Leinders, and Jan Van den Bussche.
A theory of stream queries.

Harald Beck, Minh Dao-Tran, and Thomas Eiter.
Contrasting RDF stream processing semantics.

Özgür L. Özçep.
Representation Theorems in Computer Science A Treatment in Logic Engineering.
Habilitation Thesis.

Xiangnan Ren.
Distributed RDF Stream Processing and Reasoning.

F. Roelofsen and L. Serafini.
Minimal and absent information in contexts.

S. Sarkar.
Autonomous Perception and Decision Making in Cyper-Physical Systems.
PhD thesis, Penn State University Graduate School, PA 16802, USA, 2011.
Data Snapshots: Window Functions

- Important aspect of stream processing: 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 \( S = (T, \nu) \) and \( t \in T \) a substream \( S' \subseteq S \), which means \( S' = (T', \nu') \) such that \( T' \subseteq T \) and \( \nu'(t) \subseteq \nu(t) \), for all \( t \in T' \)
Window Functions: Example

\[ S' = w(S, t) \]
Window Functions: Example

time-based window $w_{\tau}^{1,5(1)}$, looking back 1 and forward 5 steps (at most), with step size 1 (i.e., *sliding*)

$$S' = w_{\tau}^{1,5,1}(S, 40) = ([39, 45], \begin{cases} 
40 : \{\text{tram}(a_2, p_2)\}, \\
43 : \{\text{exp}(a_2, p_3)\}, \\
44 : \{\text{exp}(a_1, p_3)\} 
\end{cases})$$
LARS Formulas

LARS language: extend logic language stream access / processing

- Atoms from $\mathcal{A}$ (atomic formulas $a$)
- Boolean connectives $\land, \lor, \rightarrow, \neg$
- Window operators $\lhd$ (substream generation), $\sigma$ (reset to original stream)

$$\lhd w \iff w(S, t)$$

Examples

- $\lhd \tau 10 := \lhd_{w}^{w_{\tau} 10,0(1)}$ last 10 units (sliding time-based)
- $\lhd \tau + 5 := \lhd_{w}^{w_{\tau} 0.5(1)}$ next 5 units
- $\lhd \# n = \lhd_{w}^{w_{\# n}}$ last $n$ tuples (sliding tuple-based window)

- Temporal operators $\Diamond, \Box, \forall t$

$$\forall_{20} tram(a_2, p_1) \quad \lhd \tau + 5 \Diamond exp(a_1, p_3)$$

- Note: nesting of windows is possible!

$$\lhd \tau 60 \Box \lhd \tau 5 \Diamond tramAt(p_1) \quad \lhd \# n \lhd \tau 5 \Diamond tramAt(p_1)$$
Regular Windows & Plain Bridge Rules

■ Regular Windows

- pathologic window functions $w(S, t)$ may e.g. interpret $t$ as Gödel number of a computation; thus we hit undecidability.
- 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 (allows for data dropping)
  (ii) for some $l \geq p \geq 0$ polynomial in $|kb_i(0)|$, we have $w(S, t) = w(S', t + p)$ for every $t$ and streams $S, S'$ that coincide on the past (future) $l$ time points around $t$ resp. $t+p$ having data (informally, $w$ is periodic and $l$ is a limit for evaluation)
- small (polynomial-size) time-based, tuple-based windows are regular.

■ Plain Bridge Rules

- simply memorizing the data within the limit with their actual time points is not feasible under a space constraint wrt. $|kb_i(0)|$
- a (schematic) bridge rule is plain, if time references are to evaluation time ($\top^0@Z\bot$) with fixed offset $os$ ($Z\pm os$)
- For plain bridge rules (cf. running example), full memorization can be avoided

Lemma. 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.
MCS limitations for Smart CPS

- **Issues** [Costantini and Gasperis, 2016], [Cabalar *et al.*, 2017]
  - *Grounded (Propositional) Knowledge*
    ⇒ expand initial grounding of open rules gradually (fixpoint)
  - *Logical Omniscience and Unbounded Resources*
    ⇒ delay bridge rule application *with commitment*
  - *Update Problem (by Environment)*
    ⇒ environment update prior to mngmt update (tMCS)
  - *Full System Knowledge*
    ⇒ look up yellow pages for neighbors
  - *Static System*
    ⇒ yellow pages of current contexts (cf. dynamic configuration [Dao-Tran *et al.*, 2011])
  - *Unique Source*
    ⇒ dynamic name binding: pick suitable context (by role)
  - *Uniform Knowledge Representation Format*
    ⇒ model / KB alignment
  - *Equilibria Computation and Consistency Check*
    black box (privacy) vs glass box contexts (efficiency)

- **Proposal:** *dynamic mMCS (dmCS)*, special contexts (e.g. yellow pages)

- **But:** formalization amenable to analysis ?? (cf. aMCS)
CNN: Framework Implementation

- ndnSIM: a general network simulator
- solver.hex: implements a LARS fragment using the dlvhex solver (hybrid ASP)
- solver.py: comprises implementation of external predicates, e.g.
  - \( \alpha \): returns the estimated \( \hat{\alpha} \) value of the Zipf distribution

Later: Ticker implementation