ECAI 2014
T. Schaub et al. (Eds.)

doi:10.3233/978-1-61499-419-0-159

Multi-Context Systems for Reactive Reasoning in Dynamic Environments[1]

Gerhard Brewka and **Stefan Ellmauthaler** and **Jörg Pührer**[2]

Abstract. We show in this paper how managed multi-context systems (mMCS) can be turned into a reactive formalism suitable for continuous reasoning in dynamic environments. We extend mMCS with (abstract) sensors and define the notion of a run of the extended systems. We then show how typical problems arising in online reasoning can be addressed: handling potentially inconsistent sensor input, modeling intelligent forms of forgetting, and controlling the reasoning effort spent by contexts. We also investigate the complexity of some important related decision problems.

1 Introduction

Research in knowledge representation (KR) faces two major problems. First of all, a large variety of different languages for representing knowledge - each of them useful for particular types of problems - has been produced. There are many situations where the integration of the knowledge represented in diverse formalisms is crucial, and principled ways of achieving this integration are needed. Secondly, Most of the tools providing reasoning services for KR languages were developed for offline usage: given a knowledge base (KB) computation is one-shot, triggered by a user, through a specific query or a request to compute, say, an answer set. This is the right thing for specific types of applications where a specific answer to a particular problem instance is needed at a particular point in time. However, there are different kinds of applications where a reasoning system is continuously online and receives information about a particular system it observes. Consider an assisted living scenario where people in need of support live in an apartment equipped with various sensors, e.g. smoke detectors, cameras, and body sensors measuring relevant body functions (e.g. pulse, blood pressure). A reasoning system continuously receives sensor information. The task is to detect emergencies (health problems, forgotten medication, overheating stove,...) and cause adequate reactions (e.g. turning off the electricity, calling the ambulance, ringing an alarm). The system is continuously online and has to process a continuous stream of information rather than a fixed KB.

This poses new challenges on KR formalisms. Most importantly, the available information continuously grows. This obviously cannot go on forever as the KB needs to be kept in a manageable size. We thus need principled ways of forgetting/disregarding information. In the literature one often finds sliding window techniques [9] where information is kept for a specific, predefined period of time and forgotten if it falls out of this time window. We believe this approach is far too inflexible. What is needed is a dynamic, situation dependent way of determining whether information needs to be kept or can be given up. Ideally we would like our online KR system to guarantee specific response times; although it may be very difficult to come up with such guarantees, it is certainly necessary to find means to identify and focus on relevant parts of the available information. Moreover, although the definition of the semantics of the underlying KR formalism remains essential, we also need to impose procedural aspects reflecting the necessary modifications of the KB. This leads to a new, additional focus on *runs* of the system, rather than single evaluations.

Nonmonotonic multi-context systems (MCS) [3] were explicitly developed to handle the integration problem. In a nutshell, an MCS consists of reasoning units - called contexts for historical reasons [11] - where each unit can be connected with other units via so-called bridge rules. The collection of bridge rules associated with a context specifies additional beliefs the context is willing to accept depending on what is believed by connected contexts. The semantics of the MCS is then defined in terms of equilibria. Intuitively, an equilibrium is a collection of belief sets, one for each context, which fit together in the sense that the beliefs of each context adequately take into account what the other contexts believe.

The original framework was aimed at modeling the flow of information among contexts, consequently the addition of information to a context was the only possible operation on KBs. To capture more general forms of operations MCS were later generalized to so called managed MCS (mMCS) [5]. The main goal of this paper is to demonstrate that this additional functionality makes managed MCS are particularly well-suited as a basis for handling the mentioned problems of online reasoning systems as well. The main reason is that the operations on the knowledge bases allow us to control things like KB size, handling of inconsistent observations, focus of attention, and even whether a particular context should be idle for some time.

However, to turn mMCS into a reactive online formalism we first need to extend the framework to accommodate observations. We will do so by generalizing bridge rules so that they have access not only to belief sets of other contexts, but also to sensor data. This allows systems to become *reactive*, that is to take information about a dynamically changing world into account and to modify themselves to keep system performance up.

The rest of the paper is organized as follows. We first give the necessary background on mMCS. We then extend the framework to make it suitable for dynamic environments, in particular we show how observations can be accommodated, and we define the notion of a run of an MCS based on a sequence of observations. The subsequent sections address the following issues: handling time and the frame problem; dynamic control of KB size; focus of attention; control of computation (idle contexts). We finally discuss the complexity of

[1] This work has been partially supported by the German Research Foundation (DFG) under grants BR-1817/7-1 and FOR 1513.
[2] Institute of Computer Science, Leipzig University, Germany, email: {brewka,ellmauthaler,puehrer}@informatik.uni-leipzig.de

some important decision problems.[3]

2 Background: Multi-Context Systems

We now give the necessary background on managed MCS [5] which provide the basis for our paper. We present a slightly simplified variant of mMCS here as this allows us to better highlight the issues relevant for this paper. However, if needed it is rather straightforward (albeit technically somewhat involved) to extend all our results to the full version of mMCS. More specifically we make 2 restrictions: 1) we assume all contexts have a single logic rather than a logic suite as in [5]; 2) we assume that management functions are deterministic.

In addition we will slightly rearrange the components of an mMCS which makes them easier to use for our purposes. In particular, we will keep bridge rules and knowledge bases separate from their associated contexts. The latter will change dynamically during updates, as we will see later, and it is thus convenient to keep them separate.

mMCS build on an abstract notion of a *logic* L as a triple (KB_L, BS_L, ACC_L), where KB_L is the set of admissible knowledge bases (KBs) of L, which are sets of KB-elements ("formulas"); BS_L is the set of possible belief sets, whose elements are beliefs; and $ACC_L : KB_L \to 2^{BS_L}$ is a function describing the semantics of L by assigning to each KB a set of acceptable belief sets.

Definition 1 *A context is of the form $C = \langle L, ops, mng\rangle$ where*

- *L is a logic,*
- *ops is a set of operations,*
- *$mng : 2^{ops} \times KB_L \to KB_L$ is a management function.*

For an indexed context C_i we will write L_i, ops_i, and mng_i to denote its components.

Definition 2 *Let $\mathsf{C} = \langle C_1, \ldots, C_n\rangle$ be a tuple of contexts. A bridge rule for C_i over C $(1 \leq i \leq n)$ is of the form*

$$\mathsf{op} \leftarrow a_1, \ldots, a_j, \text{not } a_{j+1}, \ldots, \text{not } a_m, \qquad (1)$$

such that $\mathsf{op} \in ops_i$ and every a_ℓ $(1 \leq \ell \leq m)$ is an atom of form c:b, where $c \in \{1, \ldots, n\}$, and b is a belief for C_c, i.e., $\mathrm{b} \in S$ for some $S \in BS_{L_c}$.

For a bridge rule r, the operation $hd(r) = \mathsf{op}$ is the *head* of r, while $\mathrm{bd}(r) = \{a_1, \ldots, a_j, \text{not } a_{j+1}, \ldots, \text{not } a_m\}$ is the *body* of r.

Definition 3 *A* managed multi-context system (mMCS) $M = \langle \mathsf{C}, \mathsf{BR}, \mathsf{KB}\rangle$ *is a triple consisting of*

1. *a tuple of contexts $\mathsf{C} = \langle C_1, \ldots, C_n\rangle$,*
2. *a tuple $\mathsf{BR} = \langle br_1, \ldots, br_n\rangle$, where each br_i is a set of bridge rules for C_i over C,*
3. *a tuple of KBs $\mathsf{KB} = \langle kb_1, \ldots, kb_n\rangle$ such that $kb_i \in KB_{L_i}$.*

A belief state $\mathsf{S} = \langle S_1, \ldots, S_n\rangle$ for M consists of belief sets $S_i \in BS_{L_i}$, $1 \leq i \leq n$. Given a bridge rule r, an atom c:p $\in \mathrm{bd}(r)$ is satisfied by S if $\mathrm{p} \in S_c$ and a negated atom not c:p $\in \mathrm{bd}(r)$ is satisfied by S if $\mathrm{p} \notin S_c$. A literal is an atom or a negated atom. We say that r is applicable wrt. S, denoted by $\mathsf{S} \models \mathrm{bd}(r)$, if every literal $l \in \mathrm{bd}(r)$ is satisfied by S. We use $app_i(\mathsf{S}) = \{hd(r) \mid r \in br_i \wedge \mathsf{S} \models \mathrm{bd}(r)\}$ to denote the heads of all applicable bridge rules of context C_i wrt. S.

The semantics of an mMCS M is then defined in terms of equilibria, where an *equilibrium* is a belief state $\mathsf{S} = \langle S_1, \ldots, S_n\rangle$ satisfying the following condition: the belief set chosen for each context must be acceptable for the KBs obtained by applying the management function to the heads of applicable bridge rules and the KB associated with the context. More formally, for all contexts $C_i = \langle L_i, ops_i, mng_i\rangle$: let S_i be the belief set chosen for C_i. Then S is an equilibrium if, for $1 \leq i \leq n$,

$$S_i \in ACC_i(kb') \text{ for } kb' = mng_i(app_i(\mathsf{S}), kb_i).$$

Management functions allow us to model all sorts of modifications of a context's KB and thus make mMCS a powerful tool for describing the influence contexts can have on each other.

3 Reactive Multi-Context Systems

To make an mMCS M suitable for reactive reasoning in dynamic environments, we have to achieve two tasks:

1. we must provide means for the MCS to obtain information provided by sensors, and
2. we have to formally describe the behavior of the MCS over time.

Let us first show how sensors can be modeled abstractly. We assume that a sensor Π is a device which is able to provide new information in a given language L_Π specific to the sensor. From an abstract point of view, we can identify a sensor with its observation language and a current sensor reading, that is $\Pi = \langle L_\Pi, \pi\rangle$ where $\pi \subseteq L_\Pi$. Given a tuple of sensors $\Pi = \langle \Pi_1, \ldots, \Pi_k\rangle$, an observation Obs for Π (Π-observation for short) consists of a sensor reading for each sensor, that is $Obs = \langle \pi_1, \ldots, \pi_k\rangle$ where for $1 \leq i \leq k$, $\pi_i \subseteq L_{\Pi_i}$.

Each context must have access to its relevant sensors. Contexts already have means to obtain information from outside, namely the bridge rules. This suggests that the simplest way to integrate sensors is via an extension of the bridge rules: we will assume that bridge rules in their bodies can not only refer to contexts, but also to sensors.

Definition 4 *A reactive multi-context system (rMCS) over sensors $\Pi = \langle \Pi_1, \ldots, \Pi_k\rangle$ is a tuple $M = \langle \mathsf{C}, \mathsf{BR}, \mathsf{KB}\rangle$, as in Def. 3 except that the atoms a_ℓ $(1 \leq \ell \leq m)$ of bridge rules in BR for context C_i of form (1) can either be a* context atom *of form c:b as in Def. 2, or a* sensor atom *of form $\mathrm{o}@s$, where s is an index determining a sensor $(1 \leq s \leq k)$ and $\mathrm{o} \in L_{\Pi_s}$ is a piece of sensor data.*

The applicability of bridge rules now also depends on an observation:

Definition 5 *Let Π be a tuple of sensors and $Obs = \langle \pi_1, \ldots, \pi_k\rangle$ a Π-observation. A sensor atom $\mathrm{o}@s$ is* satisfied *by Obs if $\mathrm{o} \in \pi_s$; a literal* not $\mathrm{o}@s$ *is* satisfied *by Obs if $\mathrm{o} \notin \pi_s$.*

Let $M = \langle \mathsf{C}, \mathsf{BR}, \mathsf{KB}\rangle$ be an rMCS with sensors Π and S a belief state for M. A bridge rule r in BR is applicable *wrt. S and Obs, symbolically $\mathsf{S} \models_{Obs} \mathrm{bd}(r)$, if every context literal in $\mathrm{bd}(r)$ is satisfied by S and every sensor literal in $\mathrm{bd}(r)$ is satisfied by Obs.*

Instead of $app_i(\mathsf{S})$ we use $app_i(\mathsf{S}, Obs) = \{hd(r) \mid r \in br_i \wedge \mathsf{S} \models_{Obs} \mathrm{bd}(r)\}$ to define an equilibrium of an rMCS in a similar way as for an mMCS:

Definition 6 *Let $M = \langle \mathsf{C}, \mathsf{BR}, \mathsf{KB}\rangle$ be an rMCS with sensors Π and Obs a Π-observation. A belief state $\mathsf{S} = \langle S_1, \ldots, S_n\rangle$ for M is an equilibrium of M under Obs if, for $1 \leq i \leq n$,*

$$S_i \in ACC_i(mng_i(app_i(\mathsf{S}, Obs), kb_i)).$$

[3] The paper is based on preliminary ideas described in the extended abstract [2] and in [8]. However, the modeling techniques as well as the formalization presented here are new. A key difference in this respect is the handling of sensor data by means of bridge rules.

Definition 7 *Let $M = \langle \mathsf{C}, \mathsf{BR}, \mathsf{KB}\rangle$ be an rMCS with sensors Π, Obs a Π-observation, and $\mathsf{S} = \langle S_1, \ldots, S_n\rangle$ an equilibrium of M under Obs. The tuple of KBs generated by S is defined as $\mathsf{KB}^{\mathsf{S}} = \langle mng_1(app_1(\mathsf{S}, Obs), kb_1), \ldots, mng_n(app_n(\mathsf{S}, Obs), kb_n)\rangle$. The pair $\langle \mathsf{S}, \mathsf{KB}^{\mathsf{S}}\rangle$ is called full equilibrium of M under Obs.*

We now introduce the notion of a run of an rMCS induced by a sequence of observations:

Definition 8 *Let $M = \langle \mathsf{C}, \mathsf{BR}, \mathsf{KB}\rangle$ be an rMCS with sensors Π and $O = (Obs^0, Obs^1, \ldots)$ a sequence of Π-observations. A run of M induced by O is a sequence of pairs $R = (\langle \mathsf{S}^0, \mathsf{KB}^0\rangle, \langle \mathsf{S}^1, \mathsf{KB}^1\rangle, \ldots)$ such that*

- *$\langle \mathsf{S}^0, \mathsf{KB}^0\rangle$ is a full equilibrium of M under Obs^0,*
- *for $\langle \mathsf{S}^i, \mathsf{KB}^i\rangle$ with $i > 0$, $\langle \mathsf{S}^i, \mathsf{KB}^i\rangle$ is a full equilibrium of $\langle C, \mathsf{BR}, \mathsf{KB}^{i-1}\rangle$ under Obs^i.*

To illustrate the notion of a run, let's discuss a simple example. We want to model a clock which allows other contexts to add time stamps to sensor information they receive. We consider two options. We will first show how a clock can be realized which generates time internally by increasing a counter whenever a new equilibrium is computed. We later discuss a clock based on a sensor having access to "objective" time. In both cases we use integers as time stamps.

Example 1 *Consider a context C_c whose KBs (and belief sets) are of the form $\{\mathrm{now}(t)\}$ for some integer t. Let $kb^0 = \{\mathrm{now}(0)\}$. Assume the single bridge rule of the context is $\mathsf{incr} \leftarrow$, which intuitively says time should be incremented whenever an equilibrium is computed. The management function is thus defined as*

$$mng_c(\{\mathsf{incr}\}, \{\mathrm{now}(t)\}) = \{\mathrm{now}(t+1)\}$$

for each t. Since the computation of the (full) equilibrium is independent of any other contexts and observations, the context just increments its current time whenever a new equilibrium is computed. Each run of an rMCS with context C_c will thus contain for C_c the sequence of belief sets $\{\mathrm{now}(1)\}$, $\{\mathrm{now}(2)\}$, $\{\mathrm{now}(3)\}, \ldots$. The example illustrates that the system may evolve over time even if there is no observation made at all.

It is illustrative to compare this with a context $C_{c'}$ which is like the one we discussed except for the bridge rules which now are the instances of the schema

$$\mathsf{set}(\mathrm{now}(T+1)) \leftarrow c'{:}\mathrm{now}(T).$$

The management function correspondingly becomes

$$mng_{c'}(\{set(\mathrm{now}(t+1))\}, \{\mathrm{now}(t)\}) = \{\mathrm{now}(t+1)\}$$

for all t. Note that in this case no equilibrium exists! The reason for this is that by replacing $\mathrm{now}(0)$ with $\mathrm{now}(1)$ the precondition for the rule sanctioning this operation goes away. Special care thus needs to be taken when defining the operations.

In the rest of the paper we will often use an alternative approach where "objective" time is entered into the system by a particular sensor Π_{t}. In this case each update of the system makes time available to each context via the current sensor reading of Π_{t}.

In Example 1 we already used a bridge rule schema, that is a bridge rule where some of the parts are described by parameters (denoted by uppercase letters). We admit such schemata to allow for more compact representations. A bridge rule schema is just a convenient abbreviation for the set of its ground instances. The ground instances are obtained by replacing parameters by adequate ground terms. We will admit parameters for integers representing time, but also for formulas and even contexts. In most cases it will be clear from the discussion what the ground instances are, in other cases we will define them explicitly. We will also admit some kind of basic arithmetic in the bridge rules and assume the operations to be handled by grounding, as is usual, say, in answer set programming. For instance, the bridge rule schema

$$\mathsf{add}(\mathrm{p}(T+1)) \leftarrow c{:}\mathrm{p}(T), \mathrm{not}\ c{:}\neg\mathrm{p}(T+1)$$

which we will use to handle the frame problem in the next section has ground instances $\mathsf{add}(\mathrm{p}(1)) \leftarrow c{:}\mathrm{p}(0), \mathrm{not}\ c{:}\neg\mathrm{p}(1)$, $\mathsf{add}(\mathrm{p}(2)) \leftarrow c{:}\mathrm{p}(1), \mathrm{not}\ c{:}\neg\mathrm{p}(2)$, etc.

Although in principle parameters for integers lead to an infinite set of ground instances, in our applications only ground instances up to the current time (or current time plus a small constant, see Sect. 6) are needed, so the instantiations of time points remain finite.

In the upcoming sections we describe different generic modeling techniques for rMCSs. For concrete applications, these techniques can be refined and tailored towards the specific needs of the problem domain at hand. To demonstrate this aspect, we provide a more specific example from an assisted living application.

Example 2 *Although Bob suffers from dementia, he is able to live in his own apartment as it is equipped with an assisted living system that we model by means of an rMCS. Assume Bob starts to prepare his meal. He leaves the kitchen to go to the bathroom. After that, he forgets he has been cooking, goes to bed and falls asleep. The rMCS should be able to recognize a potential emergency based on the data of different sensors in the flat that monitor, e.g., the state of the kitchen equipment and track Bob's position.*

Our rMCS M has three contexts $\mathsf{C} = \langle C_{kt}, C_{hu}, C_{ig}\rangle$ and sensors $\Pi = \langle \Pi_{pow}, \Pi_{tmp}, \Pi_{pos}\rangle$. C_{kt} is the kitchen equipment context that monitors Bob's stove. Its logic $L_{kt} = \langle 2^{at_{kt}}, 2^{at_{kt}}, ACC_{id}\rangle$ has a very simple semantics ACC_{id} in which every knowledge base kb has only one accepted belief set coinciding with the formulas of kb, i.e., $ACC_{id}(kb) = \{kb\}$. The formulas (and beliefs) of C_{kt} are atoms from $at_{kt} = \{\mathrm{pw}(on), \mathrm{pw}(off), \mathrm{tm}(cold), \mathrm{tm}(hot)\}$ representing the stove's power status (on/off) and a qualitative value for its temperature (cold/hot). The bridge rules for C_{kt} over C are

$$\begin{aligned} \mathsf{setPower}(P) &\leftarrow \mathrm{switch}(P)@pow. \\ \mathsf{setTemp}(cold) &\leftarrow T@tmp, T \leq 45. \\ \mathsf{setTemp}(hot) &\leftarrow T@tmp, 45 < T. \end{aligned}$$

that react to switching the stove on or off, registered by sensor Π_{pow}, respectively read numerical temperature values from sensor Π_{tmp} and classify the temperature value as cold or hot. The management function $mng_{kt}(app, kb) =$

$$\begin{aligned} &\{\mathrm{pw}(on) \mid \mathsf{setPower}(on) \in app \vee \\ &\qquad (\mathrm{pw}(on) \in kb \wedge \mathsf{setPower}(off) \notin app)\} \cup \\ &\{\mathrm{pw}(off) \mid \mathsf{setPower}(on) \notin app \wedge \\ &\qquad (\mathrm{pw}(on) \notin kb \vee \mathsf{setPower}(off) \in app)\} \cup \\ &\{\mathrm{tm}(t) \mid \ \mathsf{setTemp}(\mathrm{t}) \in app\} \end{aligned}$$

ensures that the stove is considered on when it is switched on or when it is not being switched off and already considered on in the old knowledge base kb. Otherwise, the KB constructed by the management function contains atom $\mathrm{pw}(off)$. Context C_{hu} keeps track of Bob's position. The language of sensor Π_{pos} is given by $L_{\Pi_{pos}} = \{\mathrm{enters}(kitchen), \mathrm{enters}(bathroom), \mathrm{enters}(bedroom)\}$

and non-empty sensor readings of Π_{pos} signal when Bob has changed rooms. The semantics of C_{hu} is also ACC_{id} and its bridge rules are given by the schema

$$\mathsf{setPos}(P) \leftarrow \mathrm{enters}(P)@pos.$$

The management function writes Bob's new position into the KB whenever he changes rooms and keeps the previous position, otherwise. $C_{ig} = \langle L_{ig}, ops_i, mng_{ig}\rangle$ is the context for detecting emergencies. It is implemented as an answer-set program, hence the acceptable belief sets of L_{ig} are the answer sets of its KBs. The bridge rules of C_{ig} do not refer to sensor data but query other contexts:

$$\mathsf{extVal}(\mathrm{oven}(P,T)) \leftarrow kt{:}\mathrm{pw}(P), kt{:}\mathrm{tm}(T).$$
$$\mathsf{extVal}(\mathrm{humanPos}(P)) \leftarrow hu{:}\mathrm{pos}(P).$$

The answer-set program kb_{ig} is given by the rule

$$\mathrm{emergency} \leftarrow \mathrm{oven}(on, hot), \mathrm{not\ humanPos}(kitchen).$$

The management function of C_{ig} that adds information from the bridge rules temporarily as input facts to the context's KB is given by $mng_{ig}(app, kb) =$

$$\begin{aligned}&(kb\backslash(\{\mathrm{oven}(P,T) \leftarrow\mid P \in \{on, off\}, T \in \{cold, hot\}\}\cup\\ &\quad\{\mathrm{humanPos}(R) \leftarrow\mid \mathrm{enters}(R) \in L_{\Pi_{pos}}\}))\cup\\ &\{\mathrm{oven}(p,t) \leftarrow\mid \mathsf{extVal}(\mathrm{oven}(p,t)) \in \mathsf{app}\}\cup\\ &\{\mathrm{humanPos}(r) \leftarrow\mid \mathsf{extVal}(\mathrm{humanPos}(r)) \in app\}.\end{aligned}$$

Consider the sequence $O = (Obs^0, Obs^1)$ of Π-observations with $Obs^i = \langle \pi^i_{pow}, \pi^i_{tmp}, \pi^i_{pos}\rangle$ for $0 \le i \le 1$, $\pi^0_{pow} = \{\mathrm{switch}(on)\}$, $\pi^0_{tmp} = \{16\}$, $\pi^1_{tmp} = \{81\}$, $\pi^0_{pos} = \{enters(kitchen)\}$, $\pi^1_{pos} = \{enters(bathroom)\}$, and $\pi^i_s = \emptyset$ for all other π^i_s. Then, $\langle \mathsf{S}^0, \mathsf{KB}^0\rangle$ is a full equilibrium of M under Obs^0, where

$$\begin{aligned}\mathsf{S}^0 = \langle &\{\mathrm{pw}(on), \mathrm{tm}(cold)\}, \{\mathrm{pos}(kitchen)\},\\ &\{\mathrm{oven}(on, cold), \mathrm{humanPos}(kitchen)\}\rangle.\end{aligned}$$

and KB^0 equals S^0 except for the last component which is $kb_{ig} \cup \{\mathrm{oven}(on, cold) \leftarrow, \mathrm{humanPos}(kitchen) \leftarrow\}$. Moreover, $(\langle \mathsf{S}^0, \mathsf{KB}^0\rangle, \langle \mathsf{S}^1, \mathsf{KB}^1\rangle)$ is a run of M induced by O, where

$$\begin{aligned}\mathsf{S}^1 = \langle &\{\mathrm{pw}(on), \mathrm{tm}(hot)\}, \{\mathrm{pos}(bathroom)\},\\ &\{\mathrm{oven}(on, hot), \mathrm{humanPos}(bathroom), \mathrm{emergency}\}\rangle.\end{aligned}$$

4 Handling sensor data

In this section we discuss how to model an rMCS where possibly inconsistent sensor data can be integrated into a context C_j. To this end, we add a time tag to the sensor information and base our treatment of time on the second option discussed in the last section, that is we assume a specific time sensor Π_{t} that yields a reading π_{t} of the actual time of the form $\mathrm{now}(t)$ where t is an integer.

Let $\Pi_{j_1}, \ldots, \Pi_{j_m}$ be the sensors which provide relevant information for C_j in addition to Π_{t}. Then C_j will have bridge rules of the form

$$\mathsf{add}(\mathrm{P}, T, j_r) \leftarrow \mathrm{P}@j_r, \mathrm{now}(T)@\mathsf{t}$$

where the operation add is meant to add new, time tagged information to the context.

We assume the readings of a single sensor at a particular time point to be consistent. However, it is a common problem that the readings of different sensors may be inconsistent with each other wrt. some context dependend notion of inconsistency. To handle this we foresee a management function mng_j that operates based on a total preference ranking of the available sensors. The third argument of the add operation provides information about the source of sensor information and thus a way of imposing preferences on the information to be added. Without loss of generality assume $j_1 > \ldots > j_m$, that is sensor Π_{j_1} has highest priority.

Now let $add(\mathsf{S})$ be the set of add-operations in the heads of bridge rules active in belief state S. We define

$$Add_{j_1}(\mathsf{S}) = \{(\mathrm{p}, t) \mid \mathsf{add}(\mathrm{p}, t, j_1) \in add(\mathsf{S})\}$$

and for $1 < i \le m$ we let $Add_{j_i}(\mathsf{S}) = Add_{j_{i-1}}(\mathsf{S}) \cup$

$$\{(\mathrm{p}, t) \mid \mathsf{add}(\mathrm{p}, t, j_i) \in add(\mathsf{S}), (\mathrm{p}, t) \text{ consistent with } Add_{j_{i-1}}(\mathsf{S})\}.$$

Finally, we define $mng_j(add(\mathsf{S}), kb) = kb \cup Add_{j_m}(\mathsf{S})$.

This shows how the management function can solve conflicts among inconsistent sensor readings based on preferences among the sensors. Of course, many more strategies of integrating inconsistent sensor data can be thought of which we are not going to discuss in the paper. Please also note that the bridge rules do not necessarily have to pass on sensor information as is to the context. They may as well provide the context with some abstraction of the actual readings. For instance, the sensor temperature information $temp = 55$ may be transformed into qualitative information by a rule schema like

$$\begin{aligned}\mathsf{add}(temp = high, T, j_r) \leftarrow &temp = x@j_r, 45 \le x \le 65,\\ &\mathrm{now}(T)@\mathsf{t}.\end{aligned}$$

We next present a way to address the frame problem using bridge rules when sensors are not guaranteed to provide complete information about the state of the environment in each step. In this case we want to assume, at least for some of the atoms or literals observed at time $T-1$ which we call persistent, that they also hold at time T.

Assume p is some persistent observable property. Persistence of p is achieved by the following bridge rule schema:

$$\mathsf{add}(\mathrm{p}(T)) \leftarrow \mathrm{now}(T)@\mathsf{t}, j{:}\mathrm{p}(T-1), \mathrm{not}\ j{:}\neg\mathrm{p}(T).$$

Please note that in order to avoid non-existence of equilibria as discussed at the end of Sect. 3 the use of this rule schema for the frame problem presupposes that information about p valid at time $T-1$ remains available and is not deleted by any other bridge rule.

5 Selective forgetting and data retention

To illustrate our approach we discuss in this section a context C_d which can be used for emergency detection in dynamic environments. Assume there are m potential emergencies $E_1, \ldots, E_m$ we want the context to handle. The role of C_d is to check, based on observations made, whether one or more of the emergencies E_i are suspected or confirmed. Based on information about potential emergencies C_d adjusts the time span observations are kept. This is the basis for intelligent forgetting based on dynamic windows.

We do not make any assumption about how C_d works internally apart from the following:

- C_d may signal that emergency E_i is suspected ($susp(E_i)$) or confirmed ($conf(E_i)$).
- C_d has information about default, respectively actual window sizes for different observations ($def.win(p, x)$, $win(p, x)$), and
- about the number of time steps observations are relevant for particular emergencies ($rel(p, e, x)$).

Given facts of the form mentioned above, here is a possible collection of bridge rules for the task. The operation set sets the window size to a new value, deleting the old one. $alarm$ is an operation that adds information to the context KB signaling an alarm.

$$\begin{aligned}\mathsf{set}(\mathrm{win}(P,X)) &\leftarrow\ d{:}\mathrm{def}.\mathrm{win}(P,X), \mathrm{not}\ d{:}\mathrm{susp}(E)\\ \mathsf{set}(\mathrm{win}(P,Y)) &\leftarrow\ d{:}\mathrm{rel}(P,E,Y), d{:}\mathrm{susp}(E)\\ \mathsf{alarm}(E) &\leftarrow\ d{:}\mathrm{conf}(E)\end{aligned}$$

Finally, we have to make sure deletions of observations are performed in accordance with the determined window sizes:

$$\mathsf{del}(\mathrm{p}(T')) \leftarrow \mathrm{now}(T)@\mathsf{t}, d{:}\mathrm{win}(P,Z), T' < T - Z.$$

The management function just performs additions and deletions on the context KB. Since additions always are tagged with the current time, whereas deletions always refer to an earlier time, there can never be a conflict.

We have so far described a form of focusing where a time window is extended based on a specific suspected event. The next example shows a different form of focusing where specific information is generated and kept only during there is a potential danger in a particular room.

Example 3 *Continuing Example 2 we show how context C_{ig} can focus on specific rooms if there is a potential emergency. For the kitchen there is a threat if the stove is on, and it then becomes important to track whether someone is in the kitchen. Assume C_{ig} has a potential belief* $\mathrm{pw}(on, T)$ *expressing the stove is on since T. Focusing on the kitchen can be modeled by following the ASP-rule in C_{ig}'s KB:*

$$\mathrm{focus}(kitchen) \leftarrow \mathrm{pw}(on, T).$$

In addition we will need a bridge rule, which keeps track whether Bob is absent from a room in case that room is in the current focus:

$$\begin{aligned}\mathsf{add}(\mathrm{absence}(R,T)) \leftarrow & \mathrm{now}(T)@\mathsf{t}, ig{:}\mathrm{focus}(R),\\ & \mathrm{not}\ ig{:}\mathrm{humanpos}(R),\\ & \mathrm{not}\ ig{:}\mathrm{absence}(R,T'), T' < T.\end{aligned}$$

as well as a bridge rule to forget the absence in a room if it is no longer necessary. There the delAll operator removes all occurrences of absence with respect to a given room R from the KB of the context.

$$\begin{aligned}\mathsf{delAll}(\mathrm{absence}, R) &\leftarrow ig{:}\mathrm{humanpos}(R).\\ \mathsf{delAll}(\mathrm{absence}, R) &\leftarrow \mathrm{not}\ ig{:}\mathrm{focus}(R).\end{aligned}$$

With those modifications it is possible to generate an alert if Bob was too long away from the kitchen although the stove is active.

6 Control of computation

In this section we show how it is possible - at least to some extent - to control the effort spent on the computation of particular contexts. We introduce a specific control context C_0 which decides whether a context it controls should be idle for some time. An idle context just buffers sensor data it receives, but does not use the data for any other computations.

Let's illustrate this continuing the discussion of Sect. 5. Assume there are k different contexts for detecting potential emergencies as described earlier. The rMCS we are going to discuss is built on an architecture where each detector context C_i, $1 \leq i \leq k$ is connected via bridge rules with the control context. C_0 receives information about suspected emergencies and decides, based on this information, whether it is safe to let a context be idle for some time.

We now address the question what it means for a detector context to be idle. A detector context C_i receives relevant observations to reason whether an emergency is suspected or confirmed. In case C_i is idle, we cannot simply forget about new sensor information as it may become relevant later on, but we can buffer it so that it does not have an effect on the computation of a belief set, besides the fact that a buffered information shows up as an additional atom in the belief set which does not appear anywhere in the context's background knowledge.

To achieve this we have to modify C_i's original bridge rules by adding, to the body of each rule, the context literal $\mathrm{not}\ 0{:}\mathrm{idle}(i)$. This means that the bridge rules behave exactly as before whenever the control context does not decide to let C_i be idle.

For the case where C_i is idle, i.e. where the belief set of C_0 contains $\mathrm{idle}(i)$, we just make sure that observations are buffered. This means that for each rule of the form

$$\mathsf{add}(\mathrm{P}, T, j_r) \leftarrow \mathrm{P}@j_r, \mathrm{now}(T)@\mathsf{t}$$

in the original set of bridge rules we add

$$\mathsf{bf}(\mathrm{P}, T, j_r) \leftarrow \mathrm{P}@j_r, \mathrm{now}(T)@\mathsf{t}, 0{:}\mathrm{idle}(I).$$

The operation bf just adds the atom $\mathrm{bf}(p, t, j_r)$ to the context (we assume here that the language of the context contains constructs of this form). As mentioned above, this information is not used anywhere in the rest of the context's KB, it just sits there for later use.

The only missing piece is a bridge rule bringing back information from the buffer when the context is no longer idle. This can be done using the bridge rule $\mathsf{empty.buffer} \leftarrow \mathrm{not}\ 0{:}\mathrm{idle}(I)$. Whenever the management function has to execute this operation, it takes all information out of the buffer, checks whether it is still within the relevant time window, and if this is the case adds it to the KB, handling potential inconsistencies the way discussed in Sect. 4.

The control context uses formulas of the form $\mathrm{idle}(i, t)$ to express context i is idle until time t. We intend here to give a proof of concept, not a sophisticated control method. For this reason we simply assume the control context lets a detector context be idle for a specific constant time span c whenever the detector does not suspect an emergency. This is achieved by the following bridge rule schemata:

$$\begin{aligned}\mathsf{add}(\mathrm{suspicion}(K)) &\leftarrow K{:}\mathrm{susp}(E)\\ \mathsf{add}(\mathrm{idle}(K, T+c)) &\leftarrow \mathrm{now}(T)@\mathsf{t}, \mathrm{not}\ 0{:}\mathrm{suspicion}(K),\\ &\qquad \mathrm{not}\ 0{:}\mathrm{idle}(K, T'), T' < T + c\end{aligned}$$

Provided information of the form $\mathrm{idle}(i, t)$ is kept until the actual time is $t+2$, the last 2 conditions in the second rule schema guarantee that after being idle for period c the context must check at least once whether some emergency is suspected. To avoid a context staying idle forever, we assume the management function deletes information of this form whenever t is smaller than the current time minus 1. One more rule schema to makes sure information about idle contexts is available in the form used by detector contexts:

$$\mathsf{add}(\mathrm{idle}(K)) \leftarrow \mathrm{now}(T)@\mathsf{t}, 0{:}\mathrm{idle}(K, T'), T \leq T'.$$

7 Complexity

We want to analyze the complexity of queries on runs of rMCSs. For simplicity we do not consider parametrized bridge rules here, and assume that all knowledge bases in rMCS are finite and all management functions can be evaluated in polynomial time.

Definition 9 *The problem $Q^{\exists}$, respectively $Q^{\forall}$, is deciding whether for a given rMCS M with sensors Π, a context C_i of M, a belief b for C_i, and a finite sequence of Π-observations O it holds that $b \in S_i$ for some $\mathsf{S}^j = \langle S_1, \ldots, S_n \rangle$ ($0 \leq j \leq n$) for some run, respectively all runs, $R = (\langle \mathsf{S}^0, \mathsf{KB}^0 \rangle, \ldots, \langle \mathsf{S}^m, \mathsf{KB}^m \rangle)$ of M induced by O.*

As the complexity of an rMCS depends on that of its individual contexts we introduce the notion of *context complexity* along the lines of Eiter *et al.* [7]. To do so, we need to focus on relevant parts of belief sets by means of projection. Intuitively, among all beliefs, we only need to consider belief b that we want to query and beliefs that contribute to the application of bridge rules for deciding $Q^{\exists}$ and $Q^{\forall}$. Given M, Π, C_i, and b as in Definition 9, the *set of relevant beliefs* for a context C_j of M is given by $RB_j(M, i{:}\mathrm{b}) = \{b' \mid r \in br_j, h{:}b' \in \mathrm{bd}(r) \vee \mathrm{not}\ h{:}b' \in \mathrm{bd}(r)\} \cup \{b \mid i = j\}$. A *projected belief state* for M and $i{:}\mathrm{b}$ is a tuple $S^{i:\mathrm{b}}_{|M} = \langle S_1 \cap RB_1(M, i{:}\mathrm{b}), \ldots, S_n \cap RB_n(M, i{:}\mathrm{b}) \rangle$ where $\mathsf{S} = \langle S_1, \ldots, S_n \rangle$ is a belief state for M. The *context complexity* of C_j in M wrt. $i{:}\mathrm{b}$ for a fixed Π-observation Obs is the complexity of deciding whether for a given projected belief state S for M and $i{:}\mathrm{b}$, there is some belief state $\mathsf{S}' = \langle S'_1, \ldots, S'_n \rangle$ for M with ${\mathsf{S}'}^{i:\mathrm{b}}_{|M} = \mathsf{S}$ and $S'_j \in ACC_j(mng_j(app_j(\mathsf{S}, Obs), kb_j))$ for all $1 \leq j \leq n$. The system's context complexity $\mathcal{CC}(M, i{:}\mathrm{b})$ is a (smallest) upper bound for the context complexity classes of its contexts. Our complexity results are summarized in Table 1.

Table 1. Complexity of checking $Q^{\exists}$ and $Q^{\forall}$ (membership, completeness holds given hardness for $\mathcal{CC}(M, i{:}\mathrm{b})$.

$\mathcal{CC}(M, i{:}\mathrm{b})$	$Q^{\exists}$	$Q^{\forall}$
P	**NP**	**coNP**
$\mathbf{\Sigma^P_i}$ $(i \geq 2)$	$\mathbf{\Sigma^P_i}$	$\mathbf{\Pi^P_i}$
PSPACE	**PSPACE**	**PSPACE**

Membership for $Q^{\exists}$: a non-deterministic Turing machine can guess a projected belief state $\mathsf{S}^j = \langle S_1, \ldots, S_n \rangle$ for all m observations in O in polynomial time. Then, iteratively for each of the consecutive observations obs_j, first the context problem can be solved polynomially or using an oracle (the guess of S^j and the oracle guess can be combined which explains that we stay on the same complexity level for higher context complexity). If the answer is 'yes', S^j is a projected equilibrium. We can check whether $b \in S_i$, compute the updated knowledge bases and continue the iteration until reaching the last observation. The argument is similar for the co-problem of $Q^{\forall}$. Hardness: holds by a reduction from deciding equilibrium existence for an MCS when $\mathcal{CC}(M, i{:}\mathrm{b})$ is polynomial and by a reduction from the context complexity problem for the other results.

Note that $Q^{\exists}$ and $Q^{\forall}$ are undecidable if we allow for infinite observations. The reason is that rMCSs are expressive enough (even with very simple context logics) to simulate a Turing machine such that deciding $Q^{\exists}$ or $Q^{\forall}$ for infinite runs solves the halting problem.

8 Discussion

In this paper we introduced reactive MCS, an extension of managed MCS for online reasoning, and showed how they allow us to handle typical problems arising in this area. Although we restricted our discussion to deterministic management functions, two sources of non-determinism can be spotted by the attentive reader. On the one hand, we allow for semantics that return multiple belief sets for the same knowledge base, and, on the other hand, non-determinism can be introduced through bridge rules.

The simplest example is guessing via positive support cycles, e.g., using bridge rules like $\mathsf{add}(\mathrm{a}) \leftarrow c{:}\mathrm{a}$ that allow (under the standard interpretation of add) for belief sets with and without formula a. Multiple equilibria may lead to an exponential number of runs. In practice, non-determinism will have to be restricted. An obvious option is to choose a context formalism able to express preferences so that the semantics only returns sufficiently good solutions. For preferences between equilibria that depend on the belief sets of multiple contexts, one cannot rely on intra-context preference resolution. Here, we refer the reader to preference functions as proposed by Ellmauthaler [8]. One might also adopt language constructs for expressing preferences in ASP such as optimization statements [10] or weak constraints [6]. Essentially, these assign a quality measure to an equilibrium. With such additional quality measures at hand, the best equilibrium can be chosen for the run.

As to related work, there is quite some literature on MCS by now, for an overview see [4]. Recently an interesting approach to belief change in MCS has been proposed [14]. Other related work concerns stream reasoning in ASP [9] and in databases: a continuous version of SPARQL [1] exists, and logical considerations about continuous query languages [15] were investigated. Kowalski's logic-based framework for computing [13] is an approach which utilizes first order logic and concepts of the situation- and event-calculus in response to observations. None of these approaches combines a solution to both knowledge integration and online reasoning, as we do.

For a related alternative approach using an operator for directly manipulating KBs without contributing to the current equilibrium, we refer to the work by Gonçalves, Knorr, and Leite [12].

REFERENCES

[1] D. F. Barbieri, D. Braga, S. Ceri, E. D. Valle, and M. Grossniklaus, 'C-SPARQL: a continuous query language for RDF data streams', *Int. J. of Semantic Computing*, **4**(1), 3–25, (2010).

[2] G. Brewka, 'Towards reactive multi-context systems', in *Proc. LPNMR'13*, pp. 1–10, (2013).

[3] G. Brewka and T. Eiter, 'Equilibria in heterogeneous nonmonotonic multi-context systems', in *AAAI'07*, pp. 385–390, (2007).

[4] G. Brewka, T. Eiter, and M. Fink, 'Nonmonotonic multi-context systems: A flexible approach for integrating heterogeneous knowledge sources', in *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning*, 233–258, Springer, (2011).

[5] G. Brewka, T. Eiter, M. Fink, and A. Weinzierl, 'Managed multi-context systems', in *IJCAI'11*, pp. 786–791, (2011).

[6] F. Buccafurri, N. Leone, and P. Rullo, 'Strong and weak constraints in disjunctive datalog.', in *Proc. LPNMR'97*, pp. 2–17, (1997).

[7] T. Eiter, M. Fink, P. Schüller, and A. Weinzierl, 'Finding explanations of inconsistency in multi-context systems', in *Proc. KR'10*, (2010).

[8] S. Ellmauthaler, 'Generalizing multi-context systems for reactive stream reasoning applications', in *Proc. ICCSW'13*, pp. 17–24, (2013).

[9] M. Gebser, T. Grote, R. Kaminski, P. Obermeier, O. Sabuncu, and T. Schaub, 'Stream reasoning with answer set programming: Preliminary report', in *Proc. KR'12*, (2012).

[10] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and S. Thiele, *A users guide to gringo, clasp, clingo, and iclingo*, Potassco Team, 2010.

[11] F. Giunchiglia and L. Serafini, 'Multilanguage hierarchical logics or: How we can do without modal logics', *Artif. Intell.*, **65**(1), 29–70, (1994).

[12] R. Gonçalves, M. Knorr, and J. Leite, 'Evolving multi-context systems', in *Proc. ECAI'14*, (2014). To appear.

[13] R. A. Kowalski and F. Sadri, 'Towards a logic-based unifying framework for computing', *CoRR*, **abs/1301.6905**, (2013).

[14] Y. Wang, Z. Zhuang, and K. Wang, 'Belief change in nonmonotonic multi-context systems', in *Proc. LPNMR'13*, pp. 543–555, (2013).

[15] C. Zaniolo, 'Logical foundations of continuous query languages for data streams', in *Proc. Datalog 2.0*, pp. 177–189, (2012).