Leipzig University
Faculty of Mathematics and Computer Science
Computer Science Institute

Augustusplatz 10
04103 Leipzig
Germany

Proceedings of the International Workshop on Reactive Concepts in Knowledge Representation 2014

Technical Report 1 (2014)

Stefan Ellmauthaler
Computer Science Institute
Leipzig University
Germany

Jörg Pührer
Computer Science Institute
Leipzig University
Germany

ISSN 1430-3701

Preface

We are pleased to present the proceedings of the International Workshop on Reactive Concepts in Knowledge Representation (ReactKnow 2014), which took place on August 19$^{\text{th}}$, 2014 in Prague, co-located with the 21$^{\text{st}}$ European Conference on Artificial Intelligence (ECAI 2014).

In the field of Artificial Intelligence (AI), the subdomain of Knowledge Representation (KR) has the aim to represent, integrate, and exchange knowledge in order to provide reasoning about given and potentially incomplete information. While most traditional KR formalisms are concerned with knowledge bases that do not change over time or are only subject to occasional revisions, the advent of smart devices and recent advances in Internet technology - guided by the visions of a Semantic Web and the Internet of Things - has increased the interest in online applications that are able to directly *react on* a possibly infinite stream of external information such as sensor or network data. While current approaches for handling continuous stream data focus on rapid data processing, they lack complex reasoning capacities. Recent endeavours try to combine KR formalisms such as answer-set programming, Semantic Web ontologies, and multi-context systems with stream processing for providing knowledge-intense stream reasoning capabilities of various application areas such as urban computing, ambient assisted living, robotics, or dynamic scheduling. The goal of making sophisticated KR techniques accessible in the reactive setting poses many scientific challenges how to deal with emerging as well as expiring data in a seamless way.

ReactKnow is intended as an international forum for researchers in the AI and KR community to discuss and present advances in theory, formalisms, and applications to get closer to the vision of an artificial intelligence system which may react according to changing knowledge.

Eight papers have been accepted for presentation at ReactKnow 2014 and constitute the technical contributions to this proceedings. They cover a wide range of research topics including reasoning about action, multi-context systems, reasoning over big data, and complexity aspects. Many of these contributions introduce novel formalisms or refine existing ones. Each submission received three reviews in a blind review process.

We would like to thank the invited speaker Michael Fink for his opening talk titled "On Reactive Concepts for Multi-Context Systems and HEX-Programs". An abstract of the talk is contained in this volume.

We would like to take this opportunity to thank all the authors of submissions for ReactKnow 2014 for sending their papers and the authors of accepted papers for revising their contributions to be included in these proceedings. Moreover, we thank all PC members, reviewers, speakers, and participants for making this workshop happen. We are grateful for the help of João Leite in the review process and the support of the ECAI workshop chairs, Marina de Vos and Karl Tuyls.

August 2014

Stefan Ellmauthaler
Jörg Pührer
(Organisers ReactKnow 2014)

Programme Chairs

Stefan Ellmauthaler	Leipzig University, Germany
Jörg Pührer	Leipzig University, Germany

Programme Committee

Gerhard Brewka	Leipzig University, Germany
Michael Fink	Vienna University of Technology, Austria
Robert Kowalski	Imperial College London, UK
Joao Leite	CENTRIA, Universidade Nova de Lisboa, Portugal
Alessandra Mileo	DERI, Ireland
Philipp Obermeier	Potsdam University, Germany
Axel Polleres	Vienna University of Economics and Business, Austria
Sebastian Rudolph	Dresden Univeristy of Technology, Germany
Torsten Schaub	Potsdam University, Germany
Son Tran	NMSU, USA

Additional Reviewers

Minh Dao-Tran	Vienna University of Technology, Austria
Daria Stepanova	Vienna University of Technology, Austria

Table of Contents

Invited Talk:
On Reactive Concepts for Multi-Context Systems and HEX-Programs

Michael Fink

In this talk we will first briefly review the Multi-Context Systems (MCS) framework as proposed by Brewka and Eiter. It provides a declarative, rule-based approach to model the flow of information among different reasoning components, called contexts, by means of so-called bridge rules. More specifically, we will then consider a generalization of MCS called managed MCS (mMCS). While bridge rules were originally designed to add information to a context on the basis of beliefs held at other contexts, in an mMCS bridge rules can model arbitrary operations on a context knowledge base. Our motivation to review mMCS stems form the fact that this enhanced capability has recently triggered follow-up research that takes mMCS as the basis for modeling dynamic and reactive reasoning systems. In the second part of the talk we will turn to implementation aspects and sketch how the semantics (i.e., equilibria) can be computed for a class of mMCS (admitting centralized control) using HEX-programs. The letter essentially extend logic programs under the answer-set semantics with external atoms for incorporating external sources of information and computation. However, HEX-programs per se turn out to be inapt as a programming paradigm for modeling stateful computation with declarative control, as, e.g., required to implement dynamic, reactive extensions. For this purpose, and as a promising programming language for prototyping dynamic and reactive reasoning systems, we propose ACTHEX-programs that allow for action atoms in the head of rules which can actually effect changes to an external environment.

Reactive Reasoning with the Event Calculus

Alexander Artikis[1] and **Marek Sergot**[2] and **Georgios Paliouras**[3]

Abstract. Systems for symbolic event recognition accept as input a stream of time-stamped events from sensors and other computational devices, and seek to identify high-level composite events, collections of events that satisfy some pattern. RTEC is an Event Calculus dialect with novel implementation and 'windowing' techniques that allow for efficient event recognition, scalable to large data streams. RTEC can deal with applications where event data arrive with a (variable) delay from, and are revised by, the underlying sources. RTEC can update already recognised events and recognise new events when data arrive with a delay or following data revision. Our evaluation shows that RTEC can support real-time event recognition and is capable of meeting the performance requirements identified in a recent survey of event processing use cases. [4]

1 Introduction

Systems for symbolic event recognition ('event pattern matching') accept as input a stream of time-stamped simple, derived events (SDE)s. A SDE ('low-level event') is the result of applying a computational derivation process to some other event, such as an event coming from a sensor [21]. Using SDEs as input, event recognition systems identify composite events (CE)s of interest—collections of events that satisfy some pattern. The 'definition' of a CE ('high-level event') imposes temporal and, possibly, atemporal constraints on its subevents, i.e. SDEs or other CEs. Consider e.g. the recognition of attacks on computer network nodes given the TCP/IP messages.

Numerous recognition systems have been proposed in the literature [10]. Recognition systems with a logic-based representation of CE definitions, in particular, have recently been attracting attention [4]. They exhibit a formal, declarative semantics, in contrast to other types of recognition system that usually rely on an informal and/or procedural semantics. However, non-logic-based CE recognition systems have proven to be, overall, more efficient than logic-based ones. To address this issue, we present an efficient dialect of the Event Calculus [18], called 'Event Calculus for Run-Time reasoning' (RTEC). The Event Calculus is a logic programming formalism for representing and reasoning about events and their effects. RTEC includes novel implementation techniques for efficient CE recognition, scalable to large SDE and CE volumes. A set of interval manipulation constructs simplify CE definitions and improve reasoning efficiency. A simple indexing mechanism makes RTEC robust to SDEs that are irrelevant to the CEs we want to recognise and so RTEC can operate without SDE filtering modules. Finally, a 'windowing' mechanism supports real-time CE recognition. One main motivation for RTEC is that it should remain efficient and scalable in applications where SDEs arrive with a (variable) delay from, or are revised by, the underlying SDE detection system: RTEC can update the already recognised CEs, and recognise new CEs, when SDEs arrive with a delay or following revision. The code of RTEC is available at `<http://users.iit.demokritos.gr/~a.artikis/EC.html>`.

We evaluate RTEC on public space surveillance from video content. In this application, the SDEs are the 'short-term activities' detected on video frames—e.g. a person walking, running or being inactive. The aim then is to recognise 'long-term activities', i.e. short-term activity combinations, such as when a person leaves an object unattended, when two people are moving together, when they are having a meeting or fighting. The CE definitions are quite complex, allowing for a realistic evaluation of the efficiency of RTEC. This is in contrast to the majority of related approaches where rather simple CE definitions are used for empirical analysis. Our evaluation shows that RTEC supports real-time CE recognition and is capable of meeting the performance requirements of most of today's applications as estimated by a recent survey of event processing use cases [5].

The remainder of the paper is structured as follows. Sections 2 and 3 present the expressivity of RTEC and the way it performs reasoning. The experimental evaluation is given in Section 4. Section 5 summarises the presented work, puts the work in context, and outlines directions for further research.

2 Event Calculus

Our system for CE recognition is based on an Event Calculus dialect. The Event Calculus [18] is a logic programming formalism for representing and reasoning about events and their effects. For the dialect introduced here, called RTEC, the time model is linear and includes integer time-points. Variables start with an upper-case letter, while predicates and constants start with a lower-case letter. Where F is a *fluent*—a property that is allowed to have different values at different points in time—the term $F = V$ denotes that fluent F has value V. Boolean fluents are a special case in which the possible values are true and false. $\mathsf{holdsAt}(F = V, T)$ represents that fluent F has value V at a particular time-point T. $\mathsf{holdsFor}(F = V, I)$ represents that I is the list of the maximal intervals for which $F = V$ holds continuously. holdsAt and holdsFor are defined in such a way that, for any fluent F, $\mathsf{holdsAt}(F = V, T)$ if and only if T belongs to one of the maximal intervals of I for which $\mathsf{holdsFor}(F = V, I)$.

An *event description* in RTEC includes rules that define the event instances with the use of the happensAt predicate, the effects of events with the use of the initiatedAt and terminatedAt predicates, and the values of the fluents with the use of the holdsAt and holdsFor predicates, as well as other, possibly atemporal, constraints. Table 1 summarises the RTEC predicates available to the event description developer. The last three items in the table are interval manipulation

[1] University of Piraeus, Greece & NCSR Demokritos, Greece, email: a.artikis@unipi.gr
[2] Imperial College London, UK, email: m.sergot@imperial.ac.uk
[3] NCSR Demokritos, Greece, email: paliourg@iit.demokritos.gr
[4] A form of this paper has been submitted to IEEE TKDE.

predicates specific to RTEC.

Table 1: Main predicates of RTEC.

Predicate	**Meaning**
$\mathsf{happensAt}(E, T)$	Event E occurs at time T
$\mathsf{holdsAt}(F = V, T)$	The value of fluent F is V at time T
$\mathsf{holdsFor}(F = V, I)$	I is the list of the maximal intervals for which $F = V$ holds continuously
$\mathsf{initiatedAt}(F = V, T)$	At time T a period of time for which $F = V$ is initiated
$\mathsf{terminatedAt}(F = V, T)$	At time T a period of time for which $F = V$ is terminated
$\mathsf{relative\_complement\_all\_}(I', L, I)$	I is the list of maximal intervals produced by the relative complement of the list of maximal intervals I' with respect to every list of maximal intervals of list L
$\mathsf{union\_all}(L, I)$	I is the list of maximal intervals produced by the union of the lists of maximal intervals of list L
$\mathsf{intersect\_all}(L, I)$	I is the list of maximal intervals produced by the intersection of the lists of maximal intervals of list L

We represent instantaneous SDEs and CEs by means of happensAt, while durative SDEs and CEs are represented as fluents. The majority of CEs are durative and, therefore, in CE recognition the task generally is to compute the maximal intervals for which a fluent representing a CE has a particular value continuously.

2.1 Simple Fluents

Fluents in RTEC are *simple* or *statically determined*. We assume, without loss of generality, that these types are disjoint. For a simple fluent F, $F = V$ holds at a particular time-point T if $F = V$ has been *initiated* by an event that has occurred at some time-point earlier than T, and has not been *terminated* at some other time-point in the meantime. This is an implementation of the law of inertia. To compute the *intervals* I for which $F = V$, i.e. $\mathsf{holdsFor}(F = V, I)$, we find all time-points T_s at which $F = V$ is initiated, and then, for each T_s, we compute the first time-point T_f after T_s at which $F = V$ is 'broken'. The time-points at which $F = V$ is initiated are computed by means of domain-specific initiatedAt rules. The time-points at which $F = V$ is 'broken' are computed as follows:

$$\begin{aligned}&\mathsf{broken}(F = V,\ T_s,\ T) \leftarrow \\ &\quad \mathsf{terminatedAt}(F = V,\ T_f),\ \ T_s < T_f \leq T\end{aligned} \tag{1}$$

$$\begin{aligned}&\mathsf{broken}(F = V_1,\ T_s,\ T) \leftarrow \\ &\quad \mathsf{initiatedAt}(F = V_2,\ T_f),\ \ T_s < T_f \leq T,\ \ V_1 \neq V_2\end{aligned} \tag{2}$$

$\mathsf{broken}(F = V, T_s, T)$ represents that a maximal interval starting at T_s for which $F = V$ holds continuously is terminated at some time T_f such that $T_s < T_f \leq T$. Similar to initiatedAt, terminatedAt rules are domain-specific (examples are presented below). According to rule (2), if $F = V_2$ is initiated at T_f then effectively $F = V_1$ is terminated at time T_f, for all other possible values V_1 of F. Rule (2) ensures therefore that a fluent cannot have more than one value at any time. We do not insist that a fluent must have a value at every time-point. There is a difference between initiating a Boolean fluent $F = \mathsf{false}$ and terminating $F = \mathsf{true}$: the former implies, but is not implied by, the latter.

RTEC stores and indexes holdsFor intervals as they are computed for any given fluent-value $F = V$: thereafter intervals for $F = V$ are retrieved from the computer memory without the need for re-computation. Similarly, a holdsAt query for $F = V$ looks up F's value in the holdsFor cache.

In public space surveillance, it is often required to detect when a person leaves an object unattended. Typically, an object carried by a person is not tracked by the computer vision algorithms—only the person that carries it is tracked. The object will be tracked, i.e. it will 'appear', if and only if the person leaves it somewhere. Moreover, objects (as opposed to persons) can exhibit only inactive activity. Accordingly, we define a durative 'leaving an object' CE as follows:

$$\begin{aligned}&\mathsf{initiatedAt}(\mathit{leaving\_object}(P,\ \mathit{Obj}) = \mathsf{true},\ \ T) \leftarrow \\ &\quad \mathsf{happensAt}(\mathit{appear}(\mathit{Obj}),\ \ T), \\ &\quad \mathsf{holdsAt}(\mathit{inactive}(\mathit{Obj}) = \mathsf{true},\ \ T), \\ &\quad \mathsf{holdsAt}(\mathit{close}(P,\ \mathit{Obj}) = \mathsf{true},\ \ T), \\ &\quad \mathsf{holdsAt}(\mathit{person}(P) = \mathsf{true},\ \ T)\end{aligned} \tag{3}$$

$$\begin{aligned}&\mathsf{initiatedAt}(\mathit{leaving\_object}(P,\ \mathit{Obj}) = \mathsf{false},\ \ T) \leftarrow \\ &\quad \mathsf{happensAt}(\mathit{disappear}(\mathit{Obj}),\ \ T)\end{aligned} \tag{4}$$

In rule (3) $\mathit{leaving\_object}(P,\ \mathit{Obj}) = \mathsf{true}$ is initiated at time T if Obj 'appears' at T, it is inactive at T, and there is a person P 'close' to Obj at T. appear and $\mathit{inactive}$ are instantaneous SDE and durative SDE respectively. SDE are detected on video frames in this application. $\mathit{close}(A,\ B)$ is true when the distance between A and B does not exceed some threshold of pixel positions.

There is no explicit information about whether a tracked entity is a person or an inanimate object. We define the simple fluent $\mathit{person}(P)$ to have value true if P has been active, walking, running or moving abruptly since P first 'appeared'. The value of $\mathit{person}(P)$ has to be time-dependent because the identifier P of a tracked entity that 'disappears' (is no longer tracked) at some point may be used later to refer to another entity that 'appears' (becomes tracked), and that other entity may not necessarily be a person. This is a feature of the application and not something that is imposed by RTEC.

Unlike the specification of person, it is not clear from the data whether a tracked entity is an object. $\mathit{person}(P) = \mathsf{false}$ does not necessarily mean that P is an object; it may be that P is not tracked, or that P is a person that has never walked, run, been active or moved abruptly. Note finally that rule (3) incorporates a reasonable simplifying assumption, that a person entity will never exhibit 'inactive' activity at the moment it first 'appears' (is tracked). If an entity is 'inactive' at the moment it 'appears' it can be assumed to be an object, as in the first two conditions of rule (3).

Rule (4) expresses the conditions in which $\mathit{leaving\_object}$ ceases to be recognised. $\mathit{leaving\_object}(P,\ \mathit{Obj})$ becomes false when the object in question is picked up. An object that is picked up by someone is no longer tracked—it 'disappears'—terminating $\mathit{leaving\_object}$. ($\mathit{disappear}$ is an instantaneous SDE.) The maximal intervals during which $\mathit{leaving\_object}(P,\ \mathit{Obj}) = \mathsf{true}$ holds continuously are computed using the built-in RTEC predicate holdsFor from rules (3) and (4).

Consider another example from public space surveillance:

$$\begin{aligned}&\mathsf{initiatedAt}(\mathit{moving}(P_1, P_2) = \mathsf{true},\ \ T)\ \leftarrow \\ &\quad \mathsf{happensAt}(\mathsf{start}(\mathit{walking}(P_1) = \mathsf{true}),\ T), \\ &\quad \mathsf{holdsAt}(\mathit{walking}(P_2) = \mathsf{true},\ \ T), \\ &\quad \mathsf{holdsAt}(\mathit{close}(P_1, P_2) = \mathsf{true},\ \ T)\end{aligned} \tag{5}$$

$$\begin{aligned}&\mathsf{initiatedAt}(moving(P_1, P_2) = \mathsf{true},\ T) \leftarrow\\&\quad \mathsf{happensAt}(\mathsf{start}(walking(P_2) = \mathsf{true}),\ T),\\&\quad \mathsf{holdsAt}(walking(P_1) = \mathsf{true},\ T),\\&\quad \mathsf{holdsAt}(close(P_1, P_2) = \mathsf{true},\ T)\end{aligned} \tag{6}$$

$$\begin{aligned}&\mathsf{initiatedAt}(moving(P_1, P_2) = \mathsf{true},\ T) \leftarrow\\&\quad \mathsf{happensAt}(\mathsf{start}(close(P_1, P_2) = \mathsf{true}),\ T),\\&\quad \mathsf{holdsAt}(walking(P_1) = \mathsf{true},\ T),\\&\quad \mathsf{holdsAt}(walking(P_2) = \mathsf{true},\ T)\end{aligned} \tag{7}$$

$$\begin{aligned}&\mathsf{terminatedAt}(moving(P_1, P_2) = \mathsf{true},\ T) \leftarrow\\&\quad \mathsf{happensAt}(\mathsf{end}(walking(P_1) = \mathsf{true}),\ T)\end{aligned} \tag{8}$$

$$\begin{aligned}&\mathsf{terminatedAt}(moving(P_1, P_2) = \mathsf{true},\ T) \leftarrow\\&\quad \mathsf{happensAt}(\mathsf{end}(walking(P_2) = \mathsf{true}),\ T)\end{aligned} \tag{9}$$

$$\begin{aligned}&\mathsf{terminatedAt}(moving(P_1, P_2) = \mathsf{true},\ T) \leftarrow\\&\quad \mathsf{happensAt}(\mathsf{end}(close(P_1, P_2) = \mathsf{true}),\ T)\end{aligned} \tag{10}$$

$walking$ is a durative SDE detected on video frames. $\mathsf{start}(F = V)$ (resp. $\mathsf{end}(F = V)$) is a built-in RTEC event taking place at each starting (ending) point of each maximal interval for which $F = V$ holds continuously. The above formalisation states that P_1 is moving with P_2 when they are walking close to each other.

One of the main attractions of RTEC is that it makes available the power of logic programming to express complex temporal and atemporal constraints, as conditions in initiatedAt and terminatedAt rules for durative CEs, and happensAt rules for instantaneous CEs. E.g. standard event algebra operators, such as sequence, disjunction, parallelism, etc, may be expressed in a RTEC event description.

2.2 Statically Determined Fluents

In addition to the domain-independent definition of holdsFor, an event description may include domain-specific holdsFor rules, used to define the values of a fluent F in terms of the values of other fluents. We call such a fluent F *statically determined.* holdsFor rules of this kind make use of interval manipulation constructs—see the last three items of Table 1. Consider, e.g. $moving$ as in rules (5)–(10) but defined instead as a statically determined fluent:

$$\begin{aligned}&\mathsf{holdsFor}(moving(P_1, P_2) = \mathsf{true},\ I) \leftarrow\\&\quad \mathsf{holdsFor}(walking(P_1) = \mathsf{true},\ I_1),\\&\quad \mathsf{holdsFor}(walking(P_2) = \mathsf{true},\ I_2),\\&\quad \mathsf{holdsFor}(close(P_1, P_2) = \mathsf{true},\ I_3),\\&\quad \mathsf{intersect\_all}([I_1, I_2, I_3],\ I)\end{aligned} \tag{11}$$

The list I of maximal intervals during which P_1 is moving with P_2 is computed by determining the list I_1 of maximal intervals during which P_1 is walking, the list I_2 of maximal intervals during which P_2 is walking, the list I_3 of maximal intervals during which P_1 is close to P_2, and then calculating the list I representing the intersections of the maximal intervals in I_1, I_2 and I_3.

RTEC provides three interval manipulation constructs: union_all, intersect_all and relative_complement_all. $\mathsf{union\_all}(L, I)$ computes the list I of maximal intervals representing the union of maximal intervals of the lists of list L. For instance:

$$\mathsf{union\_all}([[(5,20),(26,30)],[(28,35)]],\ [(5,20),(26,35)])$$

A term of the form $(T_s,\ T_e)$ in RTEC represents the closed-open interval $[T_s,\ T_e)$. I in $\mathsf{union\_all}(L, I)$ is a list of maximal intervals that includes each time-point that is part of at least one list of L.

$\mathsf{intersect\_all}(L, I)$ computes the list I of maximal intervals such that I represents the intersection of maximal intervals of the lists of list L, as, e.g.:

$$\mathsf{intersect\_all}([[(26,31)],[(21,26),(30,40)]],\ [(30,31)])$$

I in $\mathsf{intersect\_all}(L, I)$ is a list of maximal intervals that includes each time-point that is part of all lists of L.

$\mathsf{relative\_complement\_all}(I', L, I)$ computes the list I of maximal intervals such that I represents the relative complements of the list of maximal intervals I' with respect to the maximal intervals of the lists of list L. Below is an example of relative_complement_all:

$$\begin{aligned}&\mathsf{relative\_complement\_all}([(5,20),(26,50)],\\&\quad [[(1,4),(18,22)],[(28,35)]],\ [(5,18),(26,28),(35,50)])\end{aligned}$$

I in $\mathsf{relative\_complement\_all}(I', L, I)$ is a list of maximal intervals that includes each time-point of I' that is not part of any list of L.

When defining a statically determined fluent F we will often want to say that, for all time-points T, $F = V$ holds at T if and only if W holds at T where W is some Boolean combination of fluent-value pairs. RTEC provides optional shorthands for writing such definitions concisely. For example, the definition

$$\begin{aligned}&G = V \ \mathsf{iff}\\&\quad (A = V_1 \ \mathsf{or}\ B = V_2),\\&\quad (A = V_1' \ \mathsf{or}\ B = V_2'),\\&\quad \mathsf{not}\ C = V_3\end{aligned} \tag{12}$$

is expanded into the following holdsFor rule:

$$\begin{aligned}&\mathsf{holdsFor}(G = V,\ I) \leftarrow\\&\quad \mathsf{holdsFor}(A = V_1,\ I_1),\ \mathsf{holdsFor}(B = V_2,\ I_2),\\&\quad \mathsf{union\_all}([I_1, I_2],\ I_3),\\&\quad \mathsf{holdsFor}(A = V_1',\ I_4),\ \mathsf{holdsFor}(B = V_2',\ I_5),\\&\quad \mathsf{union\_all}([I_4, I_5],\ I_6),\\&\quad \mathsf{intersect\_all}([I_3, I_6],\ I_7),\\&\quad \mathsf{holdsFor}(C = V_3,\ I_8),\\&\quad \mathsf{relative\_complement\_all}(I_7,\ [I_8],\ I)\end{aligned} \tag{13}$$

The required transformation takes place automatically when event descriptions are loaded into RTEC.

For a wide range of fluents, the use of interval manipulation constructs leads to a much more concise definition than the traditional style of Event Calculus representation, i.e. identifying the various conditions under which the fluent is initiated and terminated so that maximal intervals can then be computed using the domain-independent holdsFor. Compare, e.g. the statically determined and simple fluent representations of $moving$ in rules (11) and (5)–(10) respectively.

The interval manipulation constructs of RTEC can also lead to much more efficient computation. The complexity analysis may be found in [3].

2.3 Semantics

CE definitions are (locally) stratified logic programs [25]. We restrict attention to *hierarchical* definitions, those where it is possible to define a function *level* that maps all fluent-values $F = V$ and all events to the non-negative integers as follows. Events and statically determined fluent-values $F = V$ of level 0 are those whose happensAt and holdsFor definitions do not depend on any other events or fluents. In CE recognition, they represent the input SDEs. There are no fluent-values $F = V$ of simple fluents F in level 0. Events and simple fluent-values of level n are defined in terms of at least one event or fluent-value of level $n-1$ and a possibly empty set of events and fluent-values from levels lower than $n-1$. Statically determined fluent-values of level n are defined in terms of at least one fluent-value of level $n-1$ and a possibly empty set of fluent-values from levels lower than $n-1$. Note that fluent-values $F=V_i$ and $F=V_j$ for $V_i \neq V_j$ could be mapped to different levels. For simplicity however, and without loss of generality, a fluent F itself is either simple or statically determined but not both. The CE definitions of public space surveillance, i.e. the holdsFor definitions of statically determined fluents, initiatedAt and terminatedAt definitions of simple fluents and happensAt definitions of events, are available with the RTEC code.

3 Run-Time Recognition

CE recognition has to be efficient enough to support real-time decision-making, and scale to very large numbers of SDEs and CEs. SDEs may not necessarily arrive at the CE recognition system in a timely manner, i.e. there may be a (variable) delay between the time at which SDEs take place and the time at which they arrive at the CE recognition system. Moreover, SDEs may be revised, or even completely discarded in the future, as in the case where the parameters of a SDE were originally computed erroneously and are subsequently revised, or in the case of retraction of a SDE that was reported by mistake, and the mistake was realised later [1]. Note that SDE revision is not performed by the CE recognition system, but by the underlying SDE detection system.

RTEC performs CE recognition by computing and storing the maximal intervals of fluents and the time-points in which events occur. CE recognition takes place at specified query times $Q_1, Q_2, \ldots$. At each Q_i the SDEs that fall within a specified interval—the 'working memory' (*WM*) or 'window'—are taken into consideration. All SDEs that took place before or at $Q_i - WM$ are discarded. This is to make the cost of CE recognition dependent only on the *WM* size and not on the complete SDE history. The *WM* size, and the temporal distance between two consecutive query times — the 'step' ($Q_i - Q_{i-1}$) — are set by the user.

At Q_i, the maximal intervals computed by RTEC are those that can be derived from SDEs that occurred in the interval $(Q_i - WM, Q_i]$, as recorded at time Q_i. When *WM* is longer than the inter-query step, i.e., when $Q_i - WM < Q_{i-1} < Q_i$, it is possible that an SDE occurs in the interval $(Q_i - WM, Q_{i-1}]$ but arrives at RTEC only after Q_{i-1}; its effects are taken into account at query time Q_i. And similarly for SDEs that took place in $(Q_i - WM, Q_{i-1}]$ and were subsequently revised after Q_{i-1}. In the common case that SDEs arrive at RTEC with delays, or there is SDE revision, it is preferable therefore to make *WM* longer than the inter-query step. Note that information may still be lost. Any SDEs arriving or revised between Q_{i-1} and Q_i are discarded at Q_i if they took place before or at $Q_i - WM$. To reduce the possibility of losing information, one may increase the *WM* size. Doing so, however, decreases recognition efficiency.

Figure 1 illustrates windowing in RTEC. In this example we have $WM > Q_i - Q_{i-1}$. To avoid clutter, Figure 1 shows streams of only five SDEs. These are displayed below *WM*, with dots for instantaneous SDEs and lines for durative ones. For the sake of the example, we are interested in recognising just two CEs:

- CE_s, represented as a simple fluent (see Section 2.1). The starting and ending points, and the maximal intervals of CE_s are displayed above *WM* in Figure 1.
- CE_{std}, represented as a statically determined fluent (see Section 2.2). For the example, the maximal intervals of CE_{std} are defined to be the union of the maximal intervals of the two durative SDEs in Figure 1. The maximal intervals of CE_{std} are displayed above the CE_s intervals.

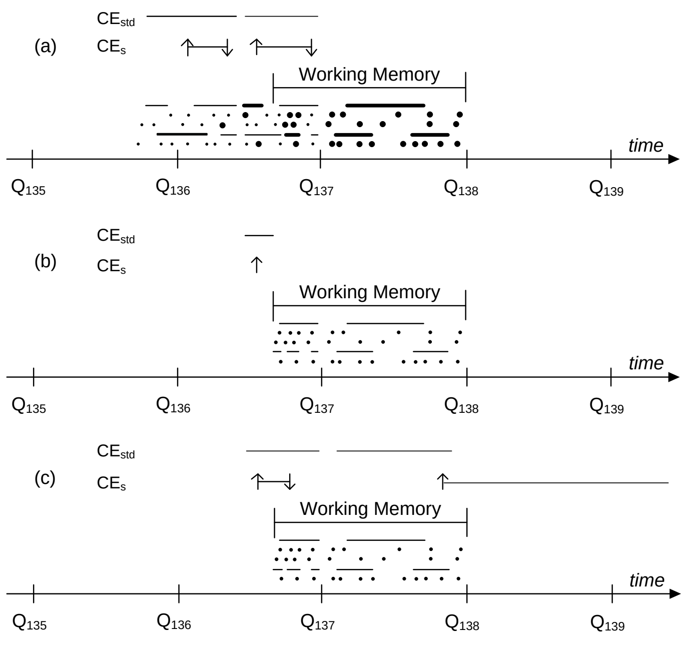


Figure 1: Windowing in RTEC.

For simplicity, we assume that both CE_s and CE_{std} are defined only in terms of SDE, i.e. they are not defined in terms of other CEs.

Figure 1 shows the steps that are followed in order to recognise CEs at an arbitrary query time, say Q_{138}. Figure 1(a) shows the state of RTEC as computation begins at Q_{138}. All SDEs that took place before or at $Q_{137} - WM$ were retracted at Q_{137}. The thick lines and dots represent the SDEs that arrived at RTEC between Q_{137} and Q_{138}; some of them took place before Q_{137}. Figure 1(a) also shows the maximal intervals for the CE fluents CE_s and CE_{std} that were computed and stored at Q_{137}.

The CE recognition process at Q_{138} considers the SDEs that took place in $(Q_{138} - WM, Q_{138}]$. All SDEs that took place before or at $Q_{138} - WM$ are discarded, as shown in Figure 1(b). For durative SDEs that started before $Q_{138} - WM$ and ended after that time, RTEC retracts the sub-interval up to and including $Q_{138} - WM$. Figure 1(b) shows the interval of a SDE that is partially retracted in this way.

Now consider CE intervals. At Q_i some of the maximal intervals computed at Q_{i-1} might have become invalid. This is because some SDEs occurring in $(Q_i - WM, Q_{i-1}]$ might have arrived or been revised after Q_{i-1}: their existence could not have been known at Q_{i-1}. Determining which CE intervals should be (partly) retracted in these circumstances can be computationally very expensive. See Section 5 for a discussion. We find it simpler, and more efficient, to discard all CE intervals in $(Q_i - WM, Q_i]$ and compute all intervals from scratch in that period. CE intervals that have ended before or at $Q_i - WM$ are discarded. Depending on the user requirements, these intervals may be stored in a database for retrospective inspection of the activities of a system.

In Figure 1(b), the earlier of the two maximal intervals computed for CE_{std} at Q_{137} is discarded at Q_{138} since its endpoint is before

$Q_{138}-WM$. The later of the two intervals overlaps $Q_{138}-WM$ (an interval 'overlaps' a time-point t if the interval starts before or at t and ends after or at that time) and is partly retracted at Q_{138}. Its starting point could not have been affected by SDEs arriving between $Q_{138}-WM$ and Q_{138} but its endpoint has to be recalculated. Accordingly, the sub-interval from $Q_{138}-WM$ is retracted at Q_{138}.

In this example, the maximal intervals of CE_{std} are determined by computing the union of the maximal intervals of the two durative SDEs shown in Figure 1. At Q_{138}, only the SDE intervals in $(Q_{138}-WM, Q_{138}]$ are considered. In the example, there are two maximal intervals for CE_{std} in this period as can be seen in Figure 1(c). The earlier of them has its startpoint at $Q_{138}-WM$. Since that abuts the existing, partially retracted sub-interval for CE_{std} whose endpoint is $Q_{138}-WM$, those two intervals are amalgamated into one continuous maximal interval as shown in Figure 1(c). In this way, the endpoint of the CE_{std} interval that overlapped $Q_{138}-WM$ at Q_{137} is recomputed to take account of SDEs available at Q_{138}. (In this particular example, it happens that the endpoint of this interval is the same as that computed at Q_{137}. That is merely a feature of this particular example. Had CE_{std} been defined e.g. as the *intersection* of the maximal intervals of the two durative SDE, then the intervals of CE_{std} would have changed in $(Q_{138}-WM, Q_{137}]$.)

Figure 1 also shows how the intervals of the simple fluent CE_s are computed at Q_{138}. Arrows facing upwards (downwards) denote the starting (ending) points of CE_s intervals. First, in analogy with the treatment of statically determined fluents, the earlier of the two CE_s intervals in Figure 1(a), and its start and endpoints, are retracted. They occur before $Q_{138}-WM$. The later of the two intervals overlaps $Q_{138}-WM$. The interval is retracted, and only its starting point is kept; its new endpoint, if any, will be recomputed at Q_{138}. See Figure 1(b). For simple fluents, it is simpler, and more efficient, to retract such intervals completely and reconstruct them later from their start and endpoints by means of the domain-independent holdsFor rules, rather than keeping the sub-interval that takes place before $Q_{138}-WM$, and possibly amalgamating it later with another interval, as we do for statically determined fluents.

The second step for CE_s at Q_{138} is to calculate its starting and ending points by evaluating the relevant initiatedAt and terminatedAt rules. For this, we only consider SDEs that took place in $(Q_{138}-WM, Q_{138}]$. Figure 1(c) shows the starting and ending points of CE_s in $(Q_{138}-WM, Q_{138}]$. The last ending point of CE_s that was computed at Q_{137} was invalidated in the light of the new SDEs that became available at Q_{138} (compare Figures 1(c)–(a)). Moreover, another ending point was computed at an earlier time.

Finally, in order to recognise CE_s at Q_{138} we use the domain-independent holdsFor to calculate the maximal intervals of CE_s given its starting and ending points. The later of the two CE_s intervals computed at Q_{137} became shorter when re-computed at Q_{138}. The second interval of CE_s at Q_{138} is open: given the SDEs available at Q_{138}, we say that CE_s holds *since* time t, where t is the last starting point of CE_s.

The discussion above showed that, when SDEs arrive with a variable delay, CE intervals computed at an earlier query time may be (partly) retracted at the current or a future query time. (And similarly if SDEs are revised.) Depending on the application requirements, RTEC may be set to report:

- CEs as soon as they are recognised, even if their intervals may be (partly) retracted in the future.
- CEs whose intervals may be partly, but not completely, retracted in the future, i.e. CEs whose intervals overlap $Q_{i+1}-WM$.
- CEs whose intervals will not be even partly retracted in the future, i.e. CEs whose intervals end before or at $Q_{i+1}-WM$.

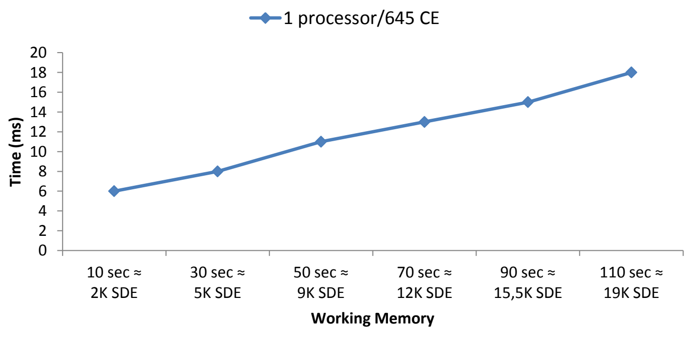
The example used for illustration shows how RTEC performs CE recognition. To support real-time reasoning, at each query time Q_i all SDEs that took place before or at Q_i-WM are discarded. To handle efficiently delayed SDEs and SDE revision, CE intervals within *WM* are computed from scratch. At Q_i, the computed maximal CE intervals are those that can be derived from SDEs that occurred in the interval $(Q_i-WM, Q_i]$, as recorded at time Q_i. For completeness, RTEC amalgamates the computed intervals to any intervals ending at Q_i-WM. More details about CE recognition in RTEC may be found at [3].

4 Experimental Results

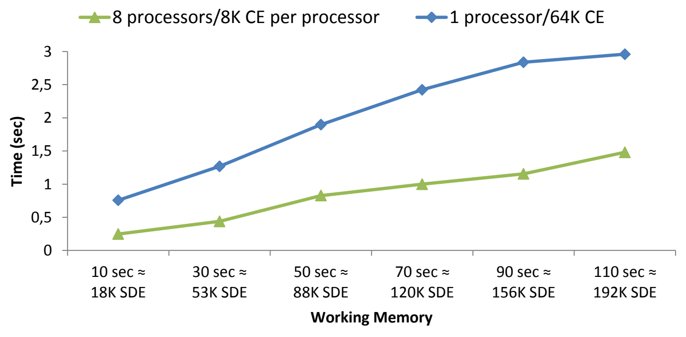
We present experimental results on the public space surveillance application. The experiments were performed on a computer with eight Intel i7 950@3.07GHz processors and 12GiB RAM, running Ubuntu Linux 12.04 and YAP Prolog 6.2.2. Each CE recognition time displayed in this section is the average of 30 runs. We use the CAVIAR benchmark dataset consisting of 28 surveillance videos of a public space `<http://groups.inf.ed.ac.uk/vision/CAVIAR/CAVIARDATA1>`. The videos are staged—actors walk around, sit down, meet one another, leave objects behind, etc. Each video has been manually annotated by the CAVIAR team in order to provide the ground truth for 'short-term activities', i.e. activities taking place in a short period of time detected on individual video frames. (The frame rate in CAVIAR is 40 ms.) The short-term activities of CAVIAR concern an entity (person or object) entering or exiting the surveillance area, walking, running, moving abruptly, being active or inactive. The CAVIAR team has also annotated the 28 videos with 'long-term activities': a person leaving an object unattended, two people meeting, moving together and fighting. Short-term activities can be viewed as SDEs while long-term activities can be viewed as CEs. Consequently, the input to RTEC in this case study includes the set of annotated short-term activities, and the output is a set of recognised long-term activities. The CE definitions and the datasets on which the experiments were performed are available with the RTEC code.

CE recognition for multiple pairs of entities. Figure 2(a) shows the results of experiments concerning all 45 pairs of the 10 entities tracked in the CAVIAR dataset. (In CAVIAR each CE concerns a pair of entities.) On average, 179 SDEs are detected per sec. We used a single processor for CE recognition concerning all 45 tracked pairs. That requires computing and storing the intervals of 645 CEs. We varied *WM* from 10 sec (≈2,000 SDEs) to 110 sec (≈19,000 SDEs). The inter-query step is set to 5 sec (≈1,000 SDEs). In all settings shown in Figure 2(a), RTEC performs real-time CE recognition.

Larger datasets. We constructed a larger dataset by taking ten copies of the original CAVIAR dataset with new identifiers for the tracked entities in each copy. The resulting dataset has 100 tracked entities, i.e. 4,950 entity pairs, while on average 1,800 SDEs take place per sec. According to the use case survey of the Event Processing Technical Society [5], in the resulting dataset there are more SDEs per sec than in most applications. First, we used a single processor for CE recognition. In this case, the intervals of approximately 64,000 CEs were computed and stored. Second, we used all eight processors of the computer in parallel. Consequently, each instance of RTEC running on a processor computed and stored the intervals of approximately 8,000 CEs. We emphasize that the input data was



CE recognition for all 10 CAVIAR tracked entities.



CE recognition for 100 tracked entities.

Figure 2: Event Recognition for Public Space Surveillance.

the same in all sets of experiments: each processor receives SDEs coming from *all* tracked entities—i.e. there was no SDE filtering to restrict the input relevant for each processor. We rely only on the indexing mechanism of RTEC to pick out relevant SDEs from the stream. RTEC employs a very simple indexing mechanism: it merely exploits YAP Prolog's standard indexing on the functor of the first argument of the head of a clause.

As in the previous set of experiments, the inter-query step is set to 5 sec, while the size of the *WM* varies from 10 to 110 sec. In this case, however, step includes approximately 9,000 SDEs, and *WM* varies from 18,000 to 192,000 SDEs. Figure 2(b) shows the average CE recognition times. In all cases RTEC performs real-time CE recognition. Figure 2(b) also shows that we can achieve significant performance gain by running RTEC in parallel on different processors. Such a gain is achieved without requiring SDE filtering.

5 Discussion

We presented RTEC, an Event Calculus dialect with novel implementation techniques that allow for efficient CE recognition, scalable to large numbers of SDEs and CEs. RTEC remains efficient and scalable in applications where SDEs arrive with a (variable) delay from, or are revised by, the SDE detection systems: it can update the already recognised CEs, and recognise new CEs, when SDEs are arrive with a delay or following revision.

RTEC has a formal, declarative semantics as opposed to most complex event processing languages, several data stream processing and event query languages, and most commercial production rule systems. Furthermore, RTEC has available the power of logic programming and thus supports atemporal reasoning and reasoning over background knowledge (as opposed to e.g. [2, 13, 19, 9]), has built-in axioms for complex temporal phenomena (as opposed to [26, 1]), explicitly represents CE intervals and thus avoids the related logical problems (as opposed to e.g. [22, 13, 9, 15]), and supports out-of-order SDE streams (as opposed to [14, 12, 9, 11, 20, 24]). Concerning the Event Calculus literature, a key feature of RTEC is that it includes a windowing technique. In contrast, no Event Calculus system (including e.g. [8, 7, 23, 24, 6]) 'forgets' or represents concisely the SDE history.

The 'Cached Event Calculus' [8] performs *update-time* reasoning: it computes and stores the consequences of a SDE as soon as it arrives. Query processing, therefore, amounts to retrieving the appropriate CE intervals from the computer memory. When a maximal interval of a fluent is asserted or retracted due to a delayed SDE, the assertion/retraction is propagated to the fluents whose validity may rely on such an interval. E.g. $propagateAssert([T_1, T_2], U)$ in the Cached Event Calculus checks whether there are new initiations as a result of asserting the interval $(T_1, T_2]$ of fluent U. In particular, $propagateAssert$ checks whether: (1) the asserted fluent U is a condition for the initiation of a fluent F at the occurrence of event E, (2) the occurrence time T of E belongs to $(T_1, T_2]$, and (3) there is not already a maximal interval for F with T as its starting point. If the above conditions are satisfied, $propagateAssert$ recursively calls $updateInit(E, T, F)$ in order to determine if F is now initiated at T, and if it is, to update the fluent interval database accordingly.

$propagateAssert$ also checks whether there are new terminations as a result of a fluent interval assertion, while $propagateRetract$ checks whether there are new initiations and terminations as a result of a fluent interval retraction. The cost of $propagateAssert$ and $propagateRetract$ is very high, especially in applications where the CE definitions include many rules with several fluents that depend on several other fluents. Furthermore, this type of reasoning is performed very frequently. RTEC avoids the costly checks every time a fluent interval is asserted/retracted due to delayed SDE arrival/revision. We found that in RTEC it is more efficient, and simpler, to discard at each query time Q_i, all intervals of fluents representing CEs in $(Q_i - WM, Q_i]$ and compute from scratch all such intervals given the SDEs available at Q_i and detected in $(Q_i - WM, Q_i]$.

For further work, we are developing techniques, based on abductive-inductive logic programming, for automated generation and refinement of CE definitions from very large datasets, with the aim of minimising the time-consuming and error-prone process of manual CE definition construction [16]. We are also porting RTEC into probabilistic logic programming frameworks, in order to deal with various types of uncertainty, such as imperfect CE definitions, incomplete and erroneous SDE streams [17].

ACKNOWLEDGEMENTS

We would like to thank the anonymous reviewers for their very helpful comments. This work has been partly funded by the EU FP7 project SPEEDD (619435).

REFERENCES

[1] D. Anicic, S. Rudolph, P. Fodor, and N. Stojanovic, 'Real-time complex event recognition and reasoning', *Applied Artificial Intelligence*, **26**(1–2), 6–57, (2012).

[2] A. Arasu, S. Babu, and J. Widom, 'The CQL continuous query language: semantic foundations and query execution', *The VLDB Journal*, **15**(2), 121–142, (2006).

[3] A. Artikis, M. Sergot, and G. Paliouras, 'Run-time composite event recognition', in *DEBS*, pp. 69–80. ACM, (2012).

[4] A. Artikis, A. Skarlatidis, F. Portet, and G. Paliouras, 'Logic-based event recognition', *Knowledge Engineering Review*, **27**(4), 469–506, (2012).

[5] P. Bizzaro. Results of the survey on event processing use cases. Event Processing Technical Society, March 2011. `http://www.slideshare.net/pedrobizarro/epts-survey-results`.

[6] I. Cervesato and A. Montanari, 'A calculus of macro-events: Progress report', in *TIME*, pp. 47–58, (2000).

[7] F. Chesani, P. Mello, M. Montali, and P. Torroni, 'A logic-based, reactive calculus of events', *Fundamenta Informaticae*, **105**(1-2), 135–161, (2010).

[8] L. Chittaro and A. Montanari, 'Efficient temporal reasoning in the cached event calculus', *Computational Intelligence*, **12**(3), 359–382, (1996).

[9] G. Cugola and A. Margara, 'TESLA: a formally defined event specification language', in *DEBS*, pp. 50–61, (2010).

[10] G. Cugola and A. Margara, 'Processing flows of information: From data stream to complex event processing', *ACM Computing Surveys*, **44**(3), 15, (2012).

[11] N. Dindar, P. M. Fischer, M. Soner, and N. Tatbul, 'Efficiently correlating complex events over live and archived data streams', in *DEBS*, pp. 243–254, (2011).

[12] L. Ding, S. Chen, E. A. Rundensteiner, J. Tatemura, W.-P. Hsiung, and K. Candan, 'Runtime semantic query optimization for event stream processing', in *ICDE*, pp. 676–685, (2008).

[13] C. Dousson and P. Le Maigat, 'Chronicle recognition improvement using temporal focusing and hierarchisation', in *IJCAI*, pp. 324–329, (2007).

[14] D. Gyllstrom, E. Wu, H.-J. Chae, Y. Diao, P. Stahlberg, and G. Anderson, 'SASE: Complex event processing over streams', in *CIDR*, (2007).

[15] A. Kakas, L. Michael, and R. Miller, 'Modular-$\mathcal{E}$: An elaboration tolerant approach to the ramification and qualification problems', in *LPNMR*, pp. 211–226, (2005).

[16] Nikos Katzouris, Alexander Artikis, and George Paliouras, 'Incremental learning of event definitions with inductive logic programming', *CoRR*, **abs/1402.5988**, (2014).

[17] A. Kimmig, B. Demoen, L. De Raedt, V. Santos Costa, and R. Rocha, 'On the implementation of the probabilistic logic programming language ProbLog', *Theory and Practice of Logic Programming*, **11**, 235–262, (2011).

[18] R. Kowalski and M. Sergot, 'A logic-based calculus of events', *New Generation Computing*, **4**(1), 67–96, (1986).

[19] J. Krämer and B. Seeger, 'Semantics and implementation of continuous sliding window queries over data streams', *ACM Transactions on Database Systems*, **34**(1), 1–49, (2009).

[20] M. Li, M. Mani, E. A. Rundensteiner, and T. Lin, 'Complex event pattern detection over streams with interval-based temporal semantics', in *DEBS*, pp. 291–302, (2011).

[21] D. Luckham and R. Schulte. Event processing glossary — version 1.1. Event Processing Technical Society, July 2008.

[22] K. Mahbub, G. Spanoudakis, and A. Zisman, 'A monitoring approach for runtime service discovery', *Automated Software Engineering*, **18**(2), 117–161, (2011).

[23] M. Montali, F. M. Maggi, F. Chesani, P. Mello, and W. M. P. van der Aalst, 'Monitoring business constraints with the Event Calculus', *ACM TIST*, **5**(1), (2014).

[24] A. Paschke and M. Bichler, 'Knowledge representation concepts for automated SLA management', *Decision Support Systems*, **46**(1), 187–205, (2008).

[25] T. Przymusinski, 'On the declarate semantics of stratified deductive databases and logic programs', in *Foundations of Deductive Databases and Logic Programming*, Morgan, (1987).

[26] V. Shet, J. Neumann, V. Ramesh, and L. Davis, 'Bilattice-based logical reasoning for human detection', in *CVPR*, (2007).

Towards Ideal Semantics for Analyzing Stream Reasoning[1]

Harald Beck and **Minh Dao-Tran** and **Thomas Eiter** and **Michael Fink** [2]

Abstract. The rise of smart applications has drawn interest to logical reasoning over data streams. Recently, different query languages and stream processing/reasoning engines were proposed in different communities. However, due to a lack of theoretical foundations, the expressivity and semantics of these diverse approaches are given only informally. Towards clear specifications and means for analytic study, a formal framework is needed to define their semantics in precise terms. To this end, we present a first step towards an ideal semantics that allows for exact descriptions and comparisons of stream reasoning systems.

1 Introduction

The emergence of sensors, networks, and mobile devices has generated a trend towards *pushing* rather than *pulling* of data in information processing. In the setting of *stream processing* [4] studied by the database community, input tuples dynamically arrive at the processing systems in form of possibly infinite streams. To deal with unboundedness of data, such systems typically apply *window operators* to obtain snapshots of recent data. The user then runs *continuous queries* which are either periodically driven by time or eagerly driven by the arrival of new input. The Continuous Query Language (CQL) [3] is a well-known stream processing language. It has a syntax close to SQL and a clear operational semantics.

Recently, the rise of *smart applications* such as smart cities, smart home, smart grid, etc., has raised interest in the topic of *stream reasoning* [16], i.e., logical reasoning on streaming data. To illustrate our contributions on this topic, we use an example from the public transport domain.

Example 1 To monitor a city's public transportation, the city traffic center receives sensor data at every stop regarding tram/bus appearances of the form $tr(X, P)$ and $bus(X, P)$ where X, P hold the tram/bus and stop identifiers, respectively. On top of this streaming data tuples (or atoms), one may ask different queries, e.g., to monitor the status of the public transport system. To keep things simple, we start with stream processing queries:

(q_1) At stop P, did a tram and a bus arrive within the last 5 min?
(q_2) At stop P, did a tram and a bus arrive *at the same time* within the last 5 min?

Consider the scenario of Fig. 1 which depicts arrival times of trams and buses. The answer to query (q_2) is yes for stop p_2 and all time points from 2 to 7. Query (q_1) also succeeds for p_1 from 11 to 13.

As for stream reasoning, later we will additionally consider a more involved query, where we are interested in whether a bus always arrived within three minutes after the last two arrivals of trams. ∎

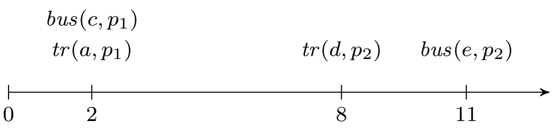


Figure 1. Traffic scenario with arrivals of trams and buses

Different communities have contributed to different aspects of this topic. (i) The Semantic Web community extends SPARQL to allow querying on streams of RDF triples. Engines such as CQELS [14] and C-SPARQL [5] also follow the snapshot semantics approach of CQL. (ii) In Knowledge Representation and Reasoning (KRR), first attempts towards expressive stream reasoning have been carried out by considering continuous data in Answer Set Programming (ASP) [9, 11] or extending Datalog to sequential logic programs [17]. However, the state of the art in either field has several shortcomings.

Approaches in (i) face difficulties with extensions of the formalism to incorporate the Closed World Assumption, nonmonotonicity, or non-determinism. Such features are important to deal with missing of incomplete data, which can temporarily happen due to unstable network connections or hardware failure. In this case, engines like C-SPARQL and CQELS remain idle, while some output based on default reasoning might be useful. Moreover, e.g., in the use case of dynamic planning on live data, multiple plans shall be generated based on previous choices and the availability of new data. This is not possible with current deterministic approaches.

On the other hand, advanced reasoning has extensively been investigated in (ii) but traditionally only on static data. First attempts towards stream reasoning reveal many problems to solve. The plain approach of [9] periodically calls the dlvhex solver [10] but is not capable of incremental reasoning and thus fails under heavy load of data. StreamLog [17] is an extension of Datalog towards stream reasoning. It always computes a single model and does not consider windows. Time-decaying logic programs [11] attempt to implement time-based windows in reactive ASP [13] but the relation to other stream processing/reasoning approaches has not yet been explored.

Moreover, as observed in [8], conceptually identical queries may produce different results in different engines. While such deviations may occur due to differences (i.e., flaws) in implementations of a common semantics, they might also arise from (correct implementations of) different semantics. For a user it is important to know the exact capabilities and the semantic behavior of a given approach. However, there is a lack of theoretical underpinning or a formal framework for stream reasoning that allows to capture different (intended) semantics in precise terms. Investigations of specific languages, as well as comparisons between different approaches, are confined to experimental analysis [15], or informal examination on specific examples. A

[1] Supported by the Austrian Science Fund (FWF) project 26471.
[2] Institut für Informationssysteme, Technische Universität Wien. email: {beck,dao,eiter,fink}@kr.tuwien.ac.at

systematic investigation, however, requires a formalism to rigorously describe the expressivity and the properties of a language.

Contributions. We present a first step towards a *formal framework for stream reasoning* that (i) provides a common ground to express concepts from different stream processing/reasoning formalisms and engines; (ii) allows systematic analysis and comparison between existing stream processing/reasoning semantics; and (iii) also provides a basis for extension towards more expressive stream reasoning. Moreover, we present (iv) exemplary formalizations based on a running example, and (v) compare our approach to existing work.

Thereby, we aim at capturing idealized stream reasoning semantics where no information is dropped and semantics are characterized as providing an abstract view over the entire stream. Second, we idealize with respect to implementations and do not consider processing time, delays or outages in the semantics itself. Moreover, we allow for a high degree of expressivity regarding time reference: We distinguish notions of truth of a formula (i) at specific time points, (ii) some time point within a window, or (iii) all time points in a window. Moreover, we allow (iv) for nested window operators, which provide a means to reason over streams within the language itself (a formal counterpart to repeated runs of continuous queries).

2 Streams

In this section, we introduce a logic-oriented view of streams and formally define generalized versions of prominent window functions.

2.1 Streaming Data

A stream is usually seen as a sequence, set or bag of tuples with a timestamp. Here, we view streams as functions from a discrete time domain to sets of logical atoms and assume no fixed schema for tuples.

We build upon mutually disjoint sets of predicates $\mathcal{P}$, constants $\mathcal{C}$, variables $\mathcal{V}$ and time variables $\mathcal{U}$. The set $\mathcal{T}$ of terms is given by $\mathcal{C} \cup \mathcal{V}$ and the set $\mathcal{A}$ of atoms is defined as $\{p(t_1, \dots, t_n) \mid p \in \mathcal{P},\ t_1, \dots, t_n \in \mathcal{T}\}$. The set $\mathcal{G}$ of *ground atoms* contains all atoms $p(t_1, \dots, t_n) \in \mathcal{A}$ such that $\{t_1, \dots, t_n\} \subseteq \mathcal{C}$. If $i, j \in \mathbb{N}$, the set $[i, j] = \{k \in \mathbb{N} \mid i \le k \le j\}$ is called an *interval*.

Definition 1 (Stream) *Let T be an interval and $v \colon \mathbb{N} \to 2^{\mathcal{G}}$ an* interpretation function *such that $v(t) = \emptyset$ for all $t \in \mathbb{N} \setminus T$. Then, the pair $S = (T, v)$ is called a* stream*, and T is called the* timeline *of S.*

The elements of a timeline are called *time points* or *timestamps*. A stream $S' = (T', v')$ is a *substream* or *window* of stream $S = (T, v)$, denoted $S' \subseteq S$, if $T' \subseteq T$ and $v'(t') \subseteq v(t')$ for all $t' \in T'$. The *cardinality* of S, denoted $\#S$, is defined by $\Sigma_{t \in T} |v(t)|$. The *restriction of S to $T' \subseteq T$*, denoted $S|_{T'}$, is the stream $(T', v|_{T'})$, where $v|_{T'}$ is the usual domain restriction of function v.

Example 2 (cont'd) The input for the scenario in Example 1 can be modeled as a stream $S = (T, v)$ where $T = [0, 13]$ and

$$\begin{array}{lll} v(2) = \{tr(a, p_1), bus(c, p_1)\} & & v(11) = \{bus(e, p_2)\} \\ v(8) = \{tr(d, p_2)\} & & v(t) \ = \emptyset \text{ otherwise.} \end{array}$$

The interpretation v can be equally represented as the following set: $\{2 \mapsto \{tr(a, p_1), bus(c, p_1)\}, 8 \mapsto \{tr(d, p_2)\}, 11 \mapsto \{bus(e, p_2)\}\}$ ∎

2.2 Windows

An essential aspect of stream reasoning is to limit the considered data to so-called *windows*, i.e., recent substreams, in order to limit the amount of data and forget outdated information.

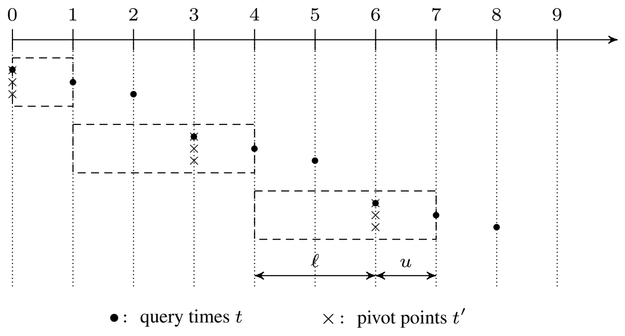


Figure 2. Time-based window $w_3^{2,1}$ with range $(2, 1)$ and step size 3

Definition 2 (Window function) *A* window function *maps from a stream $S = (T, v)$ and a time point $t \in T$ to a window $S' \subseteq S$.*

The usual time-based window of size ℓ [3] contains only the tuples of the last ℓ time units. We give a generalized definition where the window can also include the tuples of the future u time points. Based on query time t and a step size d, we derive a *pivot* point t' from which an interval $[t_\ell, t_u]$ is selected by looking backward (resp., forward) ℓ (resp., u) time units from t', i.e., $t_\ell + \ell = t'$ and $t' + u = t_u$.

Definition 3 (Time-based window) *Let $S = (T, v)$ be a stream with timeline $T = [t_{min}, t_{max}]$, let $t \in T$, and let $d, \ell, u \in \mathbb{N}$ such that $d \le \ell + u$. The* time-based window with range (ℓ, u) and step size d of S at time t *is defined by*

$$w_d^{\ell,u}(S, t) = (T', v|_{T'}),$$

where $T' = [t_\ell, t_u]$, $t_\ell = \max\{t_{min}, t' - \ell\}$ with $t' = \lfloor \frac{t}{d} \rfloor \cdot d$, and $t_u = \min\{t' + u, t_{max}\}$.

For time-based windows that target only the past ℓ time points, we abbreviate $w_d^{\ell,0}$ with w_d^ℓ. For windows which target only the future, we write w_d^{+u} for $w_d^{0,u}$. If the step size d is omitted, we take $d = 1$. Thus, the standard sliding window with range ℓ is denoted by w^ℓ.

The CQL [3] syntax for w_d^ℓ is `[Range l Slide d]` and w^ℓ corresponds to `[Range l]`. Moreover, the window `[Now]` equals `[Range 0]` and thus corresponds to w^0. The entire past stream, selected by `[Range Unbounded]` in CQL, is obtained by w^t, where t is the query time. To consider the entire stream (including the future), we can use w^n, where $n = \max T$.

Furthermore, we obtain *tumbling windows* by setting $d = \ell + u$.

Example 3 (cont'd) To formulate the monitoring over the stream S of Example 2, one can use a time-based window w^5 with a range of 5 minutes (to the past) and step size of 1 minute, i.e., the granularity of T. The results of applying this window function at $t = 5, 11$ are

$$\begin{aligned} w^5(S, 5) &= ([0, 5], \{2 \mapsto \{tr(a, p_1), bus(c, p_1)\}\}), \text{ and} \\ w^5(S, 11) &= ([6, 11], \{8 \mapsto \{tr(d, p_2)\}, 11 \mapsto \{bus(e, p_2)\}\}). \end{aligned}$$

Moreover, consider a time-based (tumbling) window with range $(2, 1)$ and step size 3. For $t_1 = 5$, we have $t_1' = \lfloor \frac{5}{3} \rfloor \cdot 3 = 3$, thus $T_1' = [\max\{0, 3 - 2\}, \min\{3 + 1, 13\}] = [1, 4]$. For $t_2 = 11$, we get $t_2' = 9$ and $T_2' = [7, 10]$. The windows for $t = 5, 11$ are

$$\begin{aligned} w_3^{2,1}(S, 5) &= ([1, 4], \{2 \mapsto \{tr(a, p_1), bus(c, p_1)\}\}), \text{ and} \\ w_3^{2,1}(S, 11) &= ([7, 10], \{8 \mapsto \{tr(d, p_2))\}\}). \end{aligned}$$

Figure 2 illustrates the progression of this window with time. ∎

The goal of the standard tuple-based window with count n is to fetch the most recent n tuples. Again, we give a more general definition which may consider future tuples. That is, relative to a time point $t \in T = [t_{min}, t_{max}]$, we want to obtain the most recent ℓ tuples (of the past) and next u tuples in the future. Thus, we must return the stream restricted to the smallest interval $T' = [t_\ell, t_u] \subseteq T$, where $t_\ell \leq t \leq t_u$, such that S contains ℓ tuples in the interval $[t_\ell, t]$ and u tuples in the interval $[t+1, t_u]$. In general, we have to discard tuples arbitrarily at time points t_ℓ and t_u in order to receive *exactly* ℓ and u tuples, respectively. In extreme cases, where fewer than ℓ tuples exist in $[t_{min}, t]$, respectively fewer than u tuples in $[t+1, t_{max}]$, we return all tuples of the according intervals. Given $t \in T$ and the tuple counts $\ell, u \in \mathbb{N}$, we define the *tuple time bounds* t_ℓ and t_u as

$$t_\ell = \max\{t_{min}\} \cup \{t' \mid t_{min} \leq t' \leq t \,\wedge\, \#(S|_{[t',t]}) \geq \ell\}, \text{ and}$$
$$t_u = \min\{t_{\max}\} \cup \{t' \mid t+1 \leq t' \leq t_{\max} \,\wedge\, \#(S|_{[t+1,t']}) \geq u\}.$$

Definition 4 (Tuple-based window) *Let $S = (T, v)$ be a stream and $t \in T$. Moreover, let $\ell, u \in \mathbb{N}$, $T_\ell = [t_\ell, t]$ and $T_u = [t+1, t_u]$, where t_ℓ and t_u are the tuple time bounds. The* tuple-based window with counts (ℓ, u) of S at time t *is defined by*

$$w^{\#\ell,u}(S,t) = (T', v'|_{T'}), \text{ where } T' = [t_\ell, t_u], \text{ and}$$

$$v'(t') = \begin{cases} v(t') & \text{for all } t' \in T' \setminus \{t_\ell, t_u\} \\ v(t') & \text{if } t' = t_\ell \text{ and } \#(S|_{T_\ell}) \leq \ell \\ X_\ell & \text{if } t' = t_\ell \text{ and } \#(S|_{T_\ell}) > \ell \\ v(t') & \text{if } t' = t_u \text{ and } \#(S|_{T_u}) \leq u \\ X_u & \text{if } t' = t_u \text{ and } \#(S|_{T_u}) > u \end{cases}$$

where $X_q \subseteq v(t_q)$, $q \in \{\ell, u\}$, such that $\#(T_q, v'|_{T_q}) = q$.

Note that the tuple-based window is unique only if for both $q \in \{\ell, u\}$, $v'(t_q) = v(t_q)$, i.e., if all atoms at the endpoints of the selected interval are retained. There are two natural possibilities to enforce the uniqueness of a tuple-based window. First, if there is a total order over all atoms, one can give a deterministic definition of the sets X_q in Def. 4. Second, one may omit the requirement that *exactly* ℓ tuples of the past, resp. u tuples of the future are contained in the window, but instead demand the substream obtained by the smallest interval $[t_\ell, t_u]$ containing *at least* ℓ past and u future tuples. Note that this approach would simplify the definition to $w^{\#\ell,u}(S,t) = (T', v|_{T'})$, requiring only to select $T' = [t_\ell, t_u]$. We abbreviate the usual tuple-based window operator $w^{\#\ell,0}$, which looks only into the past, by $w^{\#\ell}$. Similarly, $w^{\#+u}$ stands for $w^{\#0,u}$.

Example 4 (cont'd) To get the last 3 appearances of trams or buses from stream S in Example 2 at time point 11, we can apply a tuple-based window with counts $(3, 0)$. The application $w^{\#3}(S, 11)$ can lead to two possible windows (T', v_1') and (T', v_2'), where $T' = [2, 11]$, and

$$v_1' = \{2 \mapsto \{tr(a, p_1)\}, 8 \mapsto \{tr(d, p_2)\}, 11 \mapsto \{bus(e, p_2)\}\},$$
$$v_2' = \{2 \mapsto \{bus(c, p_1)\}, 8 \mapsto \{tr(d, p_2)\}, 11 \mapsto \{bus(e, p_2)\}\}.$$

The two interpretations differ at time point 2, where either $tr(a, p_1)$ or $bus(c, p_1)$ is picked to complete the collection of 3 tuples. ∎

The CQL syntax for the tuple-based window is `[Rows n]`, which corresponds to $w^{\#n}$. Note that in CQL a single stream contains tuples of a fixed schema. In the logic-oriented view, this would translate to having only one predicate. Thus, applying a tuple-based window on a stream in our sense would amount to counting tuples across different streams. To enable counting of different predicates in separation, we introduce a general form of partition-based windows.

The partition-based window CQL applies a tuple-based window function on substreams which are determined by a sequence of attributes. The syntax `[Partition By A1,...,Ak Rows N]` means that tuples are grouped into substreams by identical values $a_1, \ldots, a_k$ of attributes `A1`,..., `Ak`. From each substream, the `N` tuples with the largest timestamps are returned.

Here, we have no notion of attributes. Instead, we employ a general total *index function* $\mathrm{idx}: \mathcal{G} \rightarrow I$ from ground atoms to a finite *index set* $I \subseteq \mathbb{N}$, where for each $i \in I$ we obtain from a stream $S = (T, v)$ a substream $\mathrm{idx}_i(S) = (T, v_i)$ by taking $v_i(t) = \{a \in v(t) \mid \mathrm{idx}(a) = i\}$. Moreover, we allow for individual tuple counts $n(i) = (\ell_i, u_i)$ for each substream S_i.

Definition 5 (Partition-based window) *Let $S = (T, v)$ be a stream, $\mathrm{idx}: \mathcal{G} \rightarrow I \subseteq \mathbb{N}$, an index function, and for all $i \in I$ let $n(i) = (\ell_i, u_i) \in \mathbb{N} \times \mathbb{N}$ and $S_i = \mathrm{idx}_i(S)$. Moreover, let $t \in T$ and $w^{\#\ell_i,u_i}(S_i, t) = ([t_i^\ell, t_i^u], v_i')$ be the tuple-based window of counts (ℓ_i, u_i) of S_i at time t. Then, the* partition-based window of counts $\{(\ell_i, u_i)\}_{i \in I}$ of S at time t relative to idx *is defined by*

$$w^{\#n}_{\mathrm{idx}}(S,t) = (T', v'), \text{ where } T' = [\min_{i \in I} t_i^\ell, \max_{i \in I} t_i^u],$$

and $v'(t') = \bigcup_{i \in I} v_i'(t')$ for all $t' \in T'$.

Note that, in contrast to schema-based streaming approaches, we have multiple kinds of tuples (predicates) in one stream. Whereas other approaches may use tuple-based windows of different counts on separate streams, we can have separate tuple-counts on the corresponding substreams of a partition-based window on a single stream.

Example 5 (cont'd) Suppose we are interested in the arrival times of the last 2 trams, but we are not interested in buses. To this end, we construct a partition-based window $w^{\#n}_{\mathrm{idx}}$ as follows. We use index set $I = \{1, 2\}$, and $\mathrm{idx}(p(X, Y)) = 1$ iff $p = tr$. For the counts in the tuple-based windows of the substreams, we use $n(1) = (2, 0)$ and $n(2) = (0, 0)$. We obtain the substreams

$$S_1 = ([2, 13], \{2 \mapsto \{tr(a, p_1)\}, 8 \mapsto \{tr(d, p_2)\}\}), \text{ and}$$
$$S_2 = ([2, 13], \{2 \mapsto \{bus(c, p_1)\}, 11 \mapsto \{bus(e, p_2)\}\}),$$

and the respective tuple-based windows

$$w^{\#2}(S_1, 13) = ([2, 13], \{2 \mapsto \{tr(a, p_1)\}, 8 \mapsto \{tr(d, p_2)\}\}), \text{ and}$$
$$w^{\#0}(S_2, 13) = ([13, 13], \emptyset).$$

Consequently, we get $w^{\#n}_{\mathrm{idx}}(S, 13) = ([2, 13], v')$, where v' is

$$\{2 \mapsto \{tr(a, p_1)\}, 8 \mapsto \{tr(d, p_2)\}\}. \quad ∎$$

3 Reasoning over Streams

We are now going to utilize the above definitions of streams and windows to formalize a semantics for stream reasoning.

3.1 Stream Semantics

Towards rich expressiveness, we provide different means to relate logical truth to time. Similarly as in modal logic, we will use operators $\Box$ and $\Diamond$ to test whether a tuple (atom) or a formula holds all the time, respectively sometime in a window. Moreover, we use an *exact operator* @ to refer to specific time points. To obtain a window of the stream, we employ *window operators* $\boxplus_i$.

Definition 6 (Formulas $\mathcal{F}_k$) *The set $\mathcal{F}_k$ of* formulas (for k modalities) *is defined by the grammar*

$$\alpha ::= a \mid \neg\alpha \mid \alpha \wedge \alpha \mid \alpha \vee \alpha \mid \alpha \rightarrow \alpha \mid \Diamond\alpha \mid \Box\alpha \mid @_t\alpha \mid \boxplus_i\alpha$$

where a is any atom in $\mathcal{A}$, $i \in \{1, \dots k\}$, and $t \in \mathbb{N} \cup \mathcal{U}$.

We say a formula α is *ground*, if all its atoms are ground and for all occurrences of form $@_t\beta$ in α it holds that $t \in \mathbb{N}$. In the following semantics definition, we will consider the input stream (*urstream*) which remains unchanged, as well as dynamic substreams thereof which are obtained by (possibly nested) applications of window functions. To this end, we define a *stream choice* to be a function that returns a stream based on two input streams.Two straightforward stream choices are ch_i, for $i \in \{1, 2\}$, defined by $ch_i(S_1, S_2) = S_i$. Given a stream choice ch, we obtain for any window function w an *extended window function* $\hat{w}$ by $\hat{w}(S_1, S_2, t) = w(ch(S_1, S_2), t)$ for all $t \in \mathbb{N}$. We say $\hat{w}$ is the *extension of w (due to ch)*.

Definition 7 (Structure) *Let $S_M = (T, \upsilon)$ be a stream, $I \subseteq \mathbb{N}$ a finite index set and let $\hat{W}$ be a function that maps every $i \in I$ to an extended window function. The triple $M = \langle T, \upsilon, \hat{W}\rangle$ is called a* structure *and S_M is called the* urstream *of M.*

We now define when a ground formula holds in a structure.

Definition 8 (Entailment) *Let $M = \langle T, \upsilon, \hat{W}\rangle$ be a structure. For a substream $S = (T_S, \upsilon_S)$ of (T, υ), we define the* entailment *relation $\Vdash$ between (M, S, t) and formulas. Let $t \in T$, $a \in \mathcal{G}$, and $\alpha, \beta \in \mathcal{F}_k$ be ground formulas and let $\hat{w}_i = \hat{W}(i)$. Then,*

$$\begin{array}{lll}
M,S,t \Vdash a & \textit{iff} & a \in \upsilon_S(t)\,, \\
M,S,t \Vdash \neg\alpha & \textit{iff} & M,S,t \nVdash \alpha, \\
M,S,t \Vdash \alpha \wedge \beta & \textit{iff} & M,S,t \Vdash \alpha \textit{ and } M,S,t \Vdash \beta, \\
M,S,t \Vdash \alpha \vee \beta & \textit{iff} & M,S,t \Vdash \alpha \textit{ or } M,S,t \Vdash \beta, \\
M,S,t \Vdash \alpha \rightarrow \beta & \textit{iff} & M,S,t \nVdash \alpha \textit{ or } M,S,t \Vdash \beta, \\
M,S,t \Vdash \Diamond\alpha & \textit{iff} & M,S,t' \Vdash \alpha \textit{ for some } t' \in T_S, \\
M,S,t \Vdash \Box\alpha & \textit{iff} & M,S,t' \Vdash \alpha \textit{ for all } t' \in T_S\,, \\
M,S,t \Vdash @_{t'}\alpha & \textit{iff} & M,S,t' \Vdash \alpha \textit{ and } t' \in T_S\,, \\
M,S,t \Vdash \boxplus_i\alpha & \textit{iff} & M,S',t \Vdash \alpha \textit{ where } S' = \hat{w}_i(S_M, S, t).
\end{array}$$

If $M, S, t \Vdash \alpha$ holds, we say (M, S, t) *entails* α. Intuitively, M contains the urstream S_M which remains unchanged and S is the currently considered window. An application of a window operator $\boxplus_i$ utilizes the extended window $\hat{W}(i)$ which can take into account both the urstream S_M and the current window S to obtain a new view, as we will discuss later. The operators $\Diamond$ and $\Box$ are used to evaluate whether a formula holds at some time point, respectively at all time points in the timeline T_S of S. The operator $@_t$ allows to evaluate whether a formula holds at a specific time point t in T_S.

Example 6 (cont'd) Let $M = \langle T, \upsilon, \hat{W}\rangle$, where $S_M = (T, \upsilon)$ is the stream S from Example 2 and $\hat{W}(1) = \hat{w}^5$, i.e., the extension of w^5 of Example 3 due to ch_2. Consider the following formula:

$$\alpha = \boxplus_1(\Diamond tr(d, p_2) \wedge \Diamond bus(e, p_2))$$

We verify that $M, S_M, 11 \Vdash \alpha$ holds. First, the window operator $\boxplus_1$ selects the substream $S' = (T_{S'}, \upsilon')$, where $T_{S'} = [6, 11]$ and $\upsilon' = \upsilon|_{T'} = \{8 \mapsto \{tr(d, p_2)\}, 11 \mapsto \{bus(e, p_2)\}\}$. Next, to see that $(M, S', 11)$ entails $\Diamond tr(d, p_2) \wedge \Diamond bus(e, p_2)$, we have to find time points in the timeline $T_{S'}$ of the current window S', such that $tr(d, p_2)$ and $bus(e, p_2)$ hold, respectively. Indeed, for 8 and 11, we have $M, S_1, 8 \Vdash tr(d, p_2)$ and $M, S_1, 11 \Vdash bus(e, p_2)$. ■

DEFINITION		SCOPE
$\Theta(t)$	$= t$	time points $t \in \mathbb{N}$
$\Theta(u)$	$= \tau(u)$	time variables $u \in \mathcal{U}$
$\Theta(c)$	$= c$	constants $c \in \mathcal{C}$
$\Theta(v)$	$= \sigma(v)$	variables $v \in \mathcal{V}$
$\Theta(p(t_1, \dots, t_n)) =$ $p(\Theta(t_1), \dots, \Theta(t_n))$		predicates $p \in \mathcal{P}$ and terms $t_i \in \mathcal{T}$
$\Theta(\alpha\, \mathbf{b}\, \beta)) = \Theta(\alpha)\, \mathbf{b}\, \Theta(\beta)$		$\mathbf{b} \in \{\wedge, \vee, \rightarrow\}$
$\Theta(\mathbf{u}\alpha)$	$= \mathbf{u}\,\Theta(\alpha)$	$\mathbf{u} \in \{\neg, \Diamond, \Box\} \cup \{\boxplus_i\}_{i \in \mathbb{N}}$
$\Theta(@_u\alpha)$	$= @_t\,\Theta(\alpha)$	$@_u\alpha;\ t = \Theta(u)$
$\Theta(\alpha[u])$	$= \Theta(\alpha)[\Theta(u)]$	queries $\alpha[u]$

Table 1. Definition of substitution Θ based on query assignment (σ, τ)

3.2 Queries

We are now going to define the semantics of queries over streams.

Definition 9 (Query) *Let $S = (T, \upsilon)$ be a stream, $u \in T \cup \mathcal{U}$ and let α be a formula. Then $\alpha[u]$ denotes a* query (on S). *We say a query is* ground, *if α is ground and $u \in T$, else* non-ground.

For the evaluation of a ground query $\alpha[t]$ we will use $M, S_M, t \Vdash \alpha$. To define the semantics of non-ground queries, we need the notions of assignments and substitution. A *variable assignment σ* is a mapping $\mathcal{V} \rightarrow \mathcal{C}$ from variables to constants. A *time variable assignment τ* is a mapping $\mathcal{U} \rightarrow \mathbb{N}$ from time variables to time points. The pair (σ, τ) is called a *query assignment*. Table 1 defines the *substitution* Θ based on query assignment (σ, τ), where $\alpha, \beta \in \mathcal{F}_k$.

Let $q = \alpha[u]$ be a query on $S = (T, \upsilon)$. We say a substitution Θ *grounds* q, if $\Theta(q)$ is ground, i.e., if Θ maps all variables and time variables occurring in q. If, in addition, $\tau(x) \in T$ for every time variable $x \in \mathcal{U}$ occurring in q, we say Θ is *compatible* with q.

Definition 10 (Answer) *The* answer $?q$ *to a query $q = \alpha[t]$ on S is defined as follows. If q is ground, then $?q = yes$ if $M, S_M, t \Vdash q$ holds, and $?q = no$ otherwise. If q is non-ground, then*

$$?q = \{(\sigma, \tau) \mid \Theta \textit{ is compatible with } q \textit{ and } ?\Theta(q) = yes\}.$$

That is, the answer to a non-ground query is the set of query substitutions such that the obtained ground queries hold.

Example 7 (cont'd) We formalize the queries of Ex. 1 as follows:

$$\begin{aligned} q_1 &= \boxplus_1(\Diamond tr(X, P) \wedge \Diamond bus(Y, P))[u] \\ q_2 &= \boxplus_1\Diamond(tr(X, P) \wedge bus(Y, P))[u] \end{aligned}$$

The query $q = \boxplus_1\Diamond(tr(a, p_1) \wedge bus(c, p_1))[t]$ is ground iff $t \in \mathbb{N}$ and $?q = yes$ iff $t \in [2, 7]$. We evaluate q_1 on structure M of Ex. 6:

$$\begin{array}{l} M, S_M, t \Vdash \boxplus_1(\Diamond tr(a, p_1) \wedge \Diamond bus(c, p_1)) \;\text{ for all }\; t \in [2, 7] \\ M, S_M, t \Vdash \boxplus_1(\Diamond tr(d, p_2) \wedge \Diamond bus(e, p_2)) \;\text{ for all }\; t \in [11, 13] \end{array}$$

Thus, the following set of substitutions is the answer to q_1 in M:

$$\begin{aligned} ?q_1 = \{&(\{X \mapsto a, Y \mapsto c, P \mapsto p_1\}, \{u \mapsto t\}) \mid t \in [2, 7]\} \cup \\ \{&(\{X \mapsto d, Y \mapsto e, P \mapsto p_2\}, \{u \mapsto t\}) \mid t \in [11, 13]\} \end{aligned}$$ ■

Exact time reference. With the operator $@_t$ we can ask whether a formula holds at a specific time point t. In its non-ground version, we can utilize this operator for the selection of time points.

Example 8 (cont'd) Let $\alpha = tram(X, P) \wedge bus(Y, P)$. For each of the queries $@_U\alpha[13]$ and $\alpha[U]$, the time assignments for U in the answers will map to time points when a tram and a bus arrived simultaneously at the same stop. In both cases, the single answer is $(\{X \mapsto a, Y \mapsto c, P \mapsto p_1\}, \{U \mapsto 2\})$. Note that omitting $@_U$ in the first query would give an empty answer, since the subformula α does not hold at time point 13. ■

We observe that the operator @ allows to replay a historic query. At any time $t' > t$, we can ask $@_t\alpha[t']$ to simulate a previous query $\alpha[t]$.

Nested windows. Typically, window functions are used exclusively to restrict the processing of streams to a recent subset of the input. In our view, window functions provide a flexible means to reason over temporally local contexts within larger windows. For these nested windows we carry both M and S for the entailment relation.

Example 9 (cont'd) Consider the following additional query (q_3): At which stops P, for the last 2 two trams X, did a bus Y arrive within 3 minutes? To answer (q_3) at time point 13, we ask

$$q_3 = \boxplus_1 \Box (tr(X, P) \rightarrow \boxplus_2 \Diamond bus(Y, P))[13].$$

For $\boxplus_1$, we can use the extension $\hat{w}^{\#n}_{\text{idx}}$ of the partition-based window $w^{\#n}_{\text{idx}}$ of Example 5. Applying $\hat{W}(1)$ on the stream $S = (T, \upsilon)$ in the previous examples yields $S' = (T', \upsilon')$, where $T' = [2, 13]$ and $\upsilon' = \{2 \mapsto \{tr(a, p_1)\}, 8 \mapsto \{tr(d, p_2)\}\}$. That is, after applying this window, the current window S' no longer contains information on buses. Consequently, to check whether a bus came in both cases within 3 minutes, we must use the urstream S_M. Thus, the second extended window $\hat{W}(2) = \hat{w}^{+3}$ is the extension of the time-based window w^{+3}, which looks 3 minutes into the future, due to the stream choice ch_1. Hence, $\hat{w}^{+3}$ will create a window based on S_M and not on S'. The two time points in T' where a tram appears are 2 and 8, with P matching p_1 and p_2, respectively. Applying $\hat{W}(2)$ there yields the streams $S''_2 = (T''_2, \upsilon''_2)$ and $S''_8 = (T''_8, \upsilon''_8)$, where

$$T''_2 = [2, 5], \quad \upsilon''_2 = \{2 \mapsto \{tr(a, p_1), bus(c, p_1)\}\}, \text{ and}$$
$$T''_8 = [8, 11], \quad \upsilon''_8 = \{8 \mapsto \{tr(d, p_2)\}, 11 \mapsto \{bus(e, p_2)\}\}.$$

In both streams, we find a time point with an atom $bus(Y, p_j)$ with the same stop p_j as the tram. Thus, in both cases the subformula $\Diamond bus(Y, P)$ is satisfied and so the implication $tr(X, P) \rightarrow \boxplus_2 \Diamond bus(Y, P)$ holds at every point in time of the stream selected by $\boxplus_1$. Hence, the answer to the query is

$$?q_3 = \{\{(X \mapsto a, Y \mapsto c, P \mapsto p_1\}, \emptyset)\}, \\ \{(X \mapsto d, Y \mapsto e, P \mapsto p_2\}, \emptyset)\}\}.$$ ■

4 Discussion and Related Work

In this section we discuss the relationship of this ongoing work with existing approaches from different communities.

Modal logic. The presented formalism employs operators $\Diamond$ and $\Box$ as in modal logic [6]. Also, the definition of entailment uses a structure similar to Kripke models for multi-modal logics. However, instead of static accessibility relations, we use window functions which take into account not only the worlds (i.e., the time points) but also the interpretation function. To our best knowledge, window operators have been considered neither in modal logics nor temporal logics.

CQL. By extending SQL to deal with input streams, CQL queries are evaluated based on three sets of operators:

(i) *Stream-to-relation* operators apply window functions to the input stream to create a mapping from execution times to bags of valid tuples (w.r.t. the window) without timestamps. This mapping is called a relation.

(ii) *Relation-to-relation* operators allow for modification of relations similarly as in relational algebra, respectively SQL.

(iii) *Relation-to-stream* operators convert relations to streams by directly associating the timestamp of the execution with each tuple (RStream). The other operators IStream/DStream, which report inserted/deleted tuples, are derived from RStream.

The proposed semantics has means to capture these operators:

(i) The window operators $\boxplus_i$ keep the timestamps of the selected atoms, whereas the stream-to-relation operator discards them. The CQL query for tuple x thus corresponds to a query $\Diamond x$ of the present setting. A stream in CQL belongs to a fixed schema. As noted earlier, this corresponds to the special case with only one predicate. CQL's partition-based window is a generalization of the tuple-based window defined there. In turn, the presented partition-based window generalizes the one of CQL.

(ii) Some relational operators can essentially be captured by logical connectives, e.g., the join by conjunction. Some operators like projection will require an extension of the formalism towards rules. Moreover, we did not consider arithmetic operators and aggregation functions, which CQL inherits from SQL.

(iii) The answer to a non-ground query $\alpha[u]$ is a set of query assignments (σ, τ). To capture the RStream of CQL, we can group these assignments by the time variable u.

Example 10 Queries (q_1) and (q_2) from Example 1 can be expressed in CQL. We assume that both streams have the attributes X and P, corresponding to the first, respectively second argument of predicates tr and bus. For (q_1), we can use:

```
SELECT * FROM tr [RANGE 5], bus [RANGE 5]
WHERE tr.P = bus.P
```

On the other hand, (q_2) needs two CQL queries.

```
SELECT * AS tr_bus FROM tr [NOW], bus [NOW]
WHERE tr.P = bus.P
SELECT * FROM tr_bus [RANGE 5]
```

Here, the first query produces a new stream that contains only simultaneous tuples and the second one covers the range of 5 minutes. ■

Traditionally, stream reasoning approaches use *continuous queries*, i.e., repeated runs of queries with snapshot semantics to deal with changing information. In this work, we go a step further and enable reasoning over streams within the formalism itself by means of nested windows. One can only mimic this feature with CQL's snapshot semantics when timestamps are part of the schema and explicitly encoded. Likewise, queries to future time points can be emulated in this way, as the next example shows.

Example 11 (cont'd) In Example 9, we considered bus arrivals within 3 minutes after the last 2 trams. In CQL, such a query is not possible on the assumed schema. However, by adding a third attribute TS that carries the timestamps to the schema, the following CQL query yields the same results.

```
SELECT * FROM tr  [ROWS 2],
              bus [RANGE UNBOUNDED]
WHERE tr.P = bus.P AND bus.TS - tr.TS <= 3
```

Note that we need no partition-based window here, since trams and buses arrive from different input streams. Moreover, we must use the unbounded window for buses to cover nesting of windows in (q_3) because windows in CQL are applied at query time and not the time where a tram appearance is notified. ■

Furthermore, nested CQL queries and aggregation inherited from SQL are promising to mimic the behavior of operator $\Box$. With according rewriting, CQL eingines like STREAM [2] could be used to realize the proposed semantics.

SECRET. In [8] a model called SECRET is proposed to analyze the execution behavior of different stream processing engines (SPEs) from a practical point of view. The authors found that even the outcome of identical, simple queries vary significantly due to the different underlying processing models. There, the focus is on understanding, comparing and predicting the *behaviour of engines*. In contrast, we want to provide means that allow for a similar analytical study for the *semantics* of stream reasoning formalisms and engines. The two approaches are thus orthogonal and can be used together to compare stream reasoning engines based on different input feeding modes as well as different reasoning expressiveness.

Reactive ASP. The most recent work related to expressive stream reasoning with rules [11] is based on Reactive ASP [12]. This setting introduces logic programs that extend over time. Such programs have the following components. Two components P and Q are parametrized with a natural number t for time points. In addition, a basic component B encodes background knowledge that is not time-dependent. Moreover, sequences of pairs of arbitrary logic programs (E_i, F_j), called *online progression* are used. While P and E_i capture accumulated knowledge, Q and F_j are only valid at specific time points. Compared to reactive ASP, our semantics has no mechanism for accumulating programs, and we take only streams of atoms/facts, but no background theories. Therefore, a framework based on idealized semantics with extension to rules should be able to capture a fragment of reactive ASP where P and F_j are empty and E_i contains only facts. The foreseeable conversion can be as follows: convert rules in Q by applying an unbounded window on all body atoms of a rule, using $@_t$ to query the truth value of the atoms at time point t. Then, conclude the head to be true at t and feed facts from E_i to the input stream S.

StreamLog. Another logic-based approach towards stream reasoning is StreamLog [17]. It makes use of Datalog and introduces *temporal predicates* whose first arguments are timestamps. By introducing *sequential programs* which have syntactical restrictions on temporal rules, StreamLog defines *non-blocking negation* (for which Closed World Assumption can be safely applied) that can be used in recursive rules in a stream setting. Since sequential programs are locally stratified, they have efficiently computable perfect (i.e., unique) models. Similar to capturing a fragment of Reactive ASP, we can capture StreamLog by converting temporal atoms $p(t, x_1, \dots, x_n)$ to expressions $@_t p(x_1, \dots, x_n)$ and employing safety conditions to rules to simulate non-blocking negation. Moreover, we plan for having weaker notions of negation that might block rules but just for a bounded number of time points to the future.

ETALIS. The ETALIS system [1] aims at adding expressiveness to Complex Event Processing (CEP). It provides a rule-based language for pattern matching over event streams with *declarative monotonic semantics*. Simultaneous events are not allowed and windows are not regarded as first-class objects in the semantics, but they are available at the system implementation level. Tuple-based windows are also not directly supported. Furthermore, nesting of windows is not possible within the language, but it can be emulated with multiple rules as in CQL. On the other hand, ETALIS models complex events with time intervals and has operators to express temporal relationships between events.

5 Conclusion

We presented a first step towards a theoretical foundation for (idealistic) semantics of stream reasoning formalisms. Analytical tools to characterize, study and compare logical aspects of stream engines have been missing. To fill this gap, we provide a framework to reason over streaming data with a fine-grained control over relating the truth of tuples with their occurrences in time. It thus, e.g., allows to capture various kinds of window applications on data streams. We discussed the relationship of the proposed formalism with exsisting approaches, namely CQL, SECRET, Reactive ASP, StreamLog, and ETALIS.

Next steps include extensions of the framework to formally capture fragments of existing approaches. Towards more advanced reasoning features like recursion and non-monotonicity, we aim at a rule-based semantics on top of the presented core. Furthermore, considering intervals of time as references is an interesting research issue. To improve practicality (as a tool for formal and experimental analysis) one might also develop an operational characterization of the framework. In a longer perspective, along the same lines with [7], we aim at a formalism for stream reasoning in distributed settings across heterogeneous nodes having potentially different logical capabilities.

REFERENCES

[1] Darko Anicic, Sebastian Rudolph, Paul Fodor, and Nenad Stojanovic., 'Stream reasoning and complex event processing in ETALIS', *Semantic Web Journal*, (2012).

[2] Arvind Arasu, Brian Babcock, Shivnath Babu, Mayur Datar, Keith Ito, Itaru Nishizawa, Justin Rosenstein, and Jennifer Widom, 'Stream: The stanford stream data manager', in *SIGMOD Conference*, p. 665, (2003).

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

[4] Shivnath Babu and Jennifer Widom, 'Continuous queries over data streams', *SIGMOD Record*, **3**(30), 109–120, (2001).

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

[6] Patrick Blackburn, Maarten de Rijke, and Yde Venema, *Modal Logic*, Cambridge University Press, New York, NY, USA, 2001.

[7] Gerhard Brewka, 'Towards reactive multi-context systems', in *LPNMR*, pp. 1–10, (2013).

[8] Nihal Dindar, Nesime Tatbul, Renée J. Miller, Laura M. Haas, and Irina Botan, 'Modeling the execution semantics of stream processing engines with secret', *VLDB J.*, **22**(4), 421–446, (2013).

[9] Thang M. Do, Seng Wai Loke, and Fei Liu, 'Answer set programming for stream reasoning', in *AI*, pp. 104–109, (2011).

[10] Thomas Eiter, Giovambattista Ianni, Roman Schindlauer, and Hans Tompits, 'dlvhex: A prover for semantic-web reasoning under the answer-set semantics', in *Web Intelligence*, pp. 1073–1074, (2006).

[11] Martin Gebser, Torsten Grote, Roland Kaminski, Philipp Obermeier, Orkunt Sabuncu, and Torsten Schaub, 'Stream reasoning with answer set programming', in *KR*, (2012).

[12] Martin Gebser, Torsten Grote, Roland Kaminski, and Torsten Schaub, 'Reactive answer set programming', in *LPNMR*, pp. 54–66, (2011).

[13] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Max Ostrowski, Torsten Schaub, and Sven Thiele, 'Engineering an incremental asp solver', in *ICLP*, pp. 190–205, (2008).

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

[15] Danh Le Phuoc, Minh Dao-Tran, Minh-Duc Pham, Peter Boncz, Thomas Eiter, and Michael Fink, 'Linked stream data processing engines: Facts and figures', in *ISWC - ET*, (2012).

[16] 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).

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

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 (mMCSs) can be turned into a reactive formalism suitable for continuous reasoning in dynamic environments. We extend mMCSs 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, selective integration of knowledge, and controlling the reasoning effort spent by contexts, like setting contexts to an idle mode. We also investigate the complexity of some important related decision problems and discuss different design choices which are given to the knowledge engineer.

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 [13] 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 (MCSs) [5] were explicitly developed to handle the integration problem. In a nutshell, an MCS consists of reasoning units - called contexts for historical reasons [15] - 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 MCSs were later generalized to so called managed MCSs (mMCSs) [7]. The main goal of this paper is to demonstrate that this additional functionality makes managed MCSs 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 mMCSs 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 mMCSs. We then extend the framework to make it suitable for dynamic environments, in particular we show

[1] A former version [8] of this paper has been accepted for publication in the Procs. of ECAI-14. 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

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 some important decision problems.[3]

2 Background: Multi-Context Systems

We now give the necessary background on managed MCSs [7] which provides the basis for our paper. We present a slightly simplified variant of mMCSs 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 mMCSs. More specifically we make two restrictions: 1) we assume all contexts have a single logic rather than a logic suite as in [7]; 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. Bridge rules will be separated due to technical reasons (i.e., better presentation of the later introduced notion of a run).

An mMCS builds 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 $\mathsf{C} = \langle C_1, \dots, C_n\rangle$ be a tuple of contexts. A bridge rule for C_i over C ($1 \le i \le n$) is of the form*

$$\mathsf{op} \leftarrow a_1, \dots, a_j, \mathrm{not}\ a_{j+1}, \dots, \mathrm{not}\ a_m, \tag{1}$$

such that $\mathsf{op} \in ops_i$ and every a_ℓ ($1 \le \ell \le m$) is an atom of form $c{:}\mathrm{b}$, where $c \in \{1, \dots, n\}$, 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 $\mathrm{bd}(r) = \{a_1, \dots, a_j, \mathrm{not}\ a_{j+1}, \dots, \mathrm{not}\ a_m\}$ 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 $\mathsf{C} = \langle C_1, \dots, C_n\rangle$,*
2. *a tuple $\mathsf{BR} = \langle br_1, \dots, br_n\rangle$, where each br_i is a set of bridge rules for C_i over C,*
3. *a tuple of KBs $\mathsf{KB} = \langle kb_1, \dots, kb_n\rangle$ such that $kb_i \in KB_{L_i}$.*

[3] The paper is based on preliminary ideas described in the extended abstract [4] and in [12]. 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.

A belief state $\mathsf{S} = \langle S_1, \dots, S_n\rangle$ for M consists of belief sets $S_i \in BS_{L_i}$, $1 \le i \le n$. Given a bridge rule r, an atom $c{:}\mathrm{p} \in \mathrm{bd}(r)$ is satisfied by S if $\mathrm{p} \in S_c$ and a negated atom $\mathrm{not}\ c{:}\mathrm{p} \in \mathrm{bd}(r)$ 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 $\mathsf{S} = \langle S_1, \dots, S_n\rangle$ 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 $1 \le i \le n$,

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

Management functions allow us to model all sorts of modifications of a context's KB and thus make mMCSs 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 accomplish 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 $\Pi = \langle \Pi_1, \dots, \Pi_k\rangle$, an observation *Obs* for Π (Π-observation for short) consists of a sensor reading for each sensor, that is, $Obs = \langle \pi_1, \dots, \pi_k\rangle$ where for $1 \le i \le k$, $\pi_i \subseteq L_{\Pi_i}$.

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 $\Pi = \langle \Pi_1, \dots, \Pi_k\rangle$ is a tuple $M = \langle \mathsf{C}, \mathsf{BR}, \mathsf{KB}\rangle$, as in Def. 3 except that the atoms a_ℓ ($1 \le \ell \le m$) of bridge rules in BR for context C_i of form (1) can either be a* context atom *of form $c{:}\mathrm{b}$ as in Def. 2, or a* sensor atom *of form $\mathrm{o@}s$, where s is an index determining a sensor ($1 \le s \le k$) 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 $Obs = \langle \pi_1, \dots, \pi_k\rangle$ a Π-observation. A sensor atom $\mathrm{o@}s$ is* satisfied *by Obs if $o \in \pi_s$; a literal* $\mathrm{not}\ \mathrm{o@}s$ *is satisfied by Obs if $o \notin \pi_s$.*

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 $1 \leq i \leq n$,*

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

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 formulas and beliefs 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 logic $L_{kt} = \langle 2^{at_{kt}}, 2^{at_{kt}}, ACC_{id}\rangle$ of C_{kt} 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 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 $mng_{kt}(app, kb) =$*

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

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 the 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:

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

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

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

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 $mng_{ig}(app, kb) =$

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

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 $\Pi_{j_1}, \dots, \Pi_{j_m}$ 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

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

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-dependent 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

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

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:

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

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 $E_1, \dots, E_m$ we want the context to handle. The role of C_d is to check, based on the 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. To signal an alarm, information is added to the context KB via the operation $alarm$.

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

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.

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

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:

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

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 make sure information about idle contexts is available in the form used by detector contexts:

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

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 rMCSs 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.* [10]. 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 $RB_j(M, i{:}\text{b}) = \{b' \mid r \in br_j, h{:}b' \in \text{bd}(r) \vee \text{not } h{:}b' \in \text{bd}(r)\} \cup \{b \mid i = j\}$. A *projected belief state* for M and i:b is a tuple $S^{i:\text{b}}_{|M} = \langle S_1 \cap RB_1(M, i{:}\text{b}), \ldots, S_n \cap RB_n(M, i{:}\text{b}) \rangle$ 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:b for a fixed Π-observation *Obs* is the complexity of deciding whether for a given projected belief state S for M and i:b, there is some belief state $\mathsf{S}' = \langle S'_1, \ldots, S'_n \rangle$ for M with $\mathsf{S}'^{i:\text{b}}_{|M} = \mathsf{S}$ 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 $\mathcal{CC}(M, i{:}\text{b})$ 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 $\mathcal{CC}(M, i{:}\text{b})$).

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

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 $\mathcal{CC}(M, i{:}\text{b})$ 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 MCSs, an extension of managed MCSs 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 $\mathsf{add}(\text{a}) \leftarrow c{:}\text{a}$ 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. A simple yet practical solution is to focus on a single run, disregarding alternative equilibria. Here, one might ask which is the best full equilibrium to proceed with. In this respect, it makes sense to differentiate between non-deterministic contexts and non-determinism due to bridge rules. In the first case, it is reasonable to adopt the point of view of the answer-set programming (ASP) paradigm, i.e., the knowledge bases of a context can be seen as an encoding of a problem such that the resulting belief sets correspond to the problem solutions. Hence, as every belief set is a solution it does not matter which one to choose. Thus, if the problem to be solved is an optimisation problem that has better and worse solutions, this could be handled by choosing 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 [12]. One might also adopt language constructs for expressing preferences in ASP such as optimization statements [14] or weak constraints [9]. 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 MCSs by now, for an overview see [6]. Recently an interesting approach to belief change in MCSs has been proposed [18]. Other related work concerns stream reasoning in ASP [13] and in databases: a continuous version of SPARQL [3] exists, and logical considerations about continuous query languages [19] were investigated. Kowalski's logic-based framework for computing [17] is an approach which utilizes first order logic and concepts of the situation- and event-calculus in response to observations. Updates on knowledge-bases, based upon the outcome of a given semantics where also facilitated for other formalisms, like logic programming in general. There the iterative approaches of EPI [11] and EVOLP [1] are the most prominent. Note that none of these related approaches combines a solution to both knowledge integration and online reasoning, as we do.

The idea of updates to the knowledge-base was also formalised for database systems [2].

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 [16].

REFERENCES

[1] José Júlio Alferes, Antonio Brogi, João Alexandre Leite, and Luís Moniz Pereira, ‘Evolving logic programs’, in *8th European Conference on Logics in Artificial Intelligence (JELIA 2002)*, eds., Sergio Flesca, Sergio Greco, Nicola Leone, and Giovambattista Ianni, volume 2424 of *Lecture Notes in Computer Science*, pp. 50–61. Springer, (September 2002).

[2] Chitta Baral, Jorge Lobo, and Goce Trajcevski, ‘Formal characterizations of active databases: Part ii’, in *5th International Conference on Deductive and Object-Oriented Databases (DOOD 1997)*, eds., François Bry, Raghu Ramakrishnan, and Kotagiri Ramamohanarao, volume 1341 of *Lecture Notes in Computer Science*, pp. 247–264. Springer, (1997).

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

[4] G. Brewka, ‘Towards reactive multi-context systems’, in *12th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2013)*, pp. 1–10, (2013).

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

[6] 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).

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

[8] Gerhard Brewka, Stefan Ellmauthaler, and Jörg Pührer, ‘Multi-context systems for reactive reasoning in dynamic environments’, in *21st European Conference on Artificial Intelligence (ECAI 2014)*, (2014). To appear.

[9] F. Buccafurri, N. Leone, and P. Rullo, ‘Strong and weak constraints in disjunctive datalog.’, in *4th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 1997)*, pp. 2–17, (1997).

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

[11] Thomas Eiter, Michael Fink, Guiliana Sabbatini, and Hans Tompits, ‘A framework for declarative update specifications in logic programs’, in *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI 2001)*, ed., Bernhard Nebel, pp. 649–654. Morgan Kaufmann, (2001).

[12] S. Ellmauthaler, ‘Generalizing multi-context systems for reactive stream reasoning applications’, in *Proceedings of the 2013 Imperial College Computing Student Workshop (ICCSW 2013)*, pp. 17–24, (2013).

[13] M. Gebser, T. Grote, R. Kaminski, P. Obermeier, O. Sabuncu, and T. Schaub, ‘Stream reasoning with answer set programming: Preliminary report’, in *Proceedings of the 13th International Conference on the Principles of Knowledge Representation and Reasoning (KR 2012)*, (2012).

[14] 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.

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

[16] R. Gonçalves, M. Knorr, and J. Leite, ‘Evolving multi-context systems’, in *21st European Conference on Artificial Intelligence (ECAI 2014)*, (2014). To appear.

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

[18] Y. Wang, Z. Zhuang, and K. Wang, ‘Belief change in nonmonotonic multi-context systems’, in *12th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2013)*, pp. 543–555, (2013).

[19] C. Zaniolo, ‘Logical foundations of continuous query languages for data streams’, in *2nd International Workshop on Datalog in Academia and Industry (Datalog 2.0)*, pp. 177–189, (2012).

Asynchronous Multi-Context Systems[1]

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

Abstract. In this work, we present *asynchronous multi-context systems* (aMCSs), which provide a framework for loosely coupling different knowledge representation formalisms that allows for online reasoning in a dynamic environment. Systems of this kind may interact with the outside world via input and output streams and may therefore react to a continuous flow of external information. In contrast to recent proposals, contexts in an aMCS communicate with each other in an asynchronous way which fits the needs of many application domains and is beneficial for scalability. The federal semantics of aMCSs renders our framework an integration approach rather than a knowledge representation formalism itself. We illustrate the introduced concepts by means of an example scenario dealing with rescue services. In addition, we compare aMCSs to reactive multi-context systems and describe how to simulate the latter with our novel approach.

1 Introduction

Research in the field of knowledge representation (KR) has originated a plethora of different languages and formats. Based on these formal concepts a wealth of tools has emerged (e.g., ontologies, triple-stores, modal logics, temporal logics, nonmonotonic logics, logic programs under nonmonotonic answer set semantics, …). In a *"connected world"* it is desirable not to spread out information over different applications but to have it available for every application if need be. Expressing all the knowledge usually represented in specifically tailored languages in a universal language would be too hard to achieve from the point of view of complexity as well as the troubles arising from the translation of the representations. Instead, a framework seems desirable that integrates multiple existing formalisms in order to represent every piece of knowledge in the language that is most appropriate for it.

Another aspect that has received little attention in the development of many KR formalisms is that in a variety of applications, knowledge is provided in a constant flow of information and it is desired to reason over this knowledge in a continuous manner. Many formalisms are conceptually one-shot formalisms: given a knowledge base, the user triggers the computation of a result (e.g., the answer to a query). In this paper we aim at using KR formalisms in an *online* fashion as it has been done in recent works, e.g., on stream data processing and querying [11, 10], stream reasoning with answer set programming [7], and forgetting [9, 5].

To address the demand for an integration of heterogeneous knowledge representation formalisms together with the awareness of a continuous flow of knowledge over time, reactive multi-context systems (rMCSs) [4] and evolving multi-context systems (eMCSs) [8] where proposed. Both frameworks are based on the ideas of managed multi-context systems (mMCSs) [3] which combine multiple *contexts* which can be seen as representations of different formalisms. The semantics of rMCSs and eMCSs are based on the notion of an *equilibrium* which realises a tight semantical integration of the different context formalisms which is in many applications not necessary. Due to reasoning over all contexts, the whole computation is necessarily synchronous as the different contexts have to agree on common beliefs for establishing equilibria.

Many real world applications which utilise communication between different services use asynchronous communication protocols (e.g., web services) and compute as soon as they have appropriate information about the problem they have to address. Therefore, we introduce *asynchronous multi-context systems* (aMCSs), a framework for loosely coupled knowledge representation formalisms and services. It still provides the capabilities to express different knowledge representation languages and the translation of information from one formalism to another. In addition, aMCSs are also aware of continuous streams of information and provide ways to model the asynchronous exchange of information. To communicate with the environment, they utilise input and output streams.

We will illustrate aMCSs using the example of a task planner for medical rescue units. Here, we assume a scenario where persons are calling an emergency response team to report incidents and the employee needs to collect all relevant information about the case. Afterwards, the case needs to be classified and available resources (e.g., free ambulances, …) have to be assigned to the emergencies. In addition, current traffic data as well as the estimated time of arrival should be considered by another employee, the dispatcher. Our proposed aMCS that we understand as a recommender-system for the emergency response employee as well as to the dispatcher, incorporates different contexts like a medical ontology, a database with the current state of the ambulances, or a navigation system which is connected to a traffic density reporter. We want to stress that this might be one application where it would be a great gain for the overall system by allowing asynchronous computation and communication such that it is not necessary to wait for all other contexts (e.g., it would be unnecessary to wait for the recommendation of a plan of action for the dispatcher during the treatment of an emergency call).

The remainder of this paper is structured as follows. At first we will give a short background on concepts we need. In Section 3, we extend the basic ideas of MCS to propose our new notion of aMCS for modelling asynchronous interaction between coupled knowledge representation formalisms and formally characterise its behaviour over time. The subsequent section presents an example scenario, where asynchronous computation and a reactive response to different events is needed. Section 5 compares aMCSs to rMCSs and shows how the latter can be simulated by the former. Section 6 concludes this paper with a discussion including an outlook on future work.

[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: {ellmauthaler,puehrer}@informatik.uni-leipzig.de

2 Preliminaries

We base our approach on the underlying ideas of mMCSs [3] which extend heterogeneous multi-context systems (MCSs) [2] by a management layer. It allows for complex updates and revisions of knowledge bases and is realised by a management function that provides the updates for each equilibrium. Despite we build on mMCSs, they differ substantially in some aspects from the formalism we introduce in this work for reasons intrinsic to the asynchronous approach (cf. Section 5). Consequently, we only reuse basic notions from the original work and refer the interested reader to the paper of Brewka *et al.* [3] for full details on mMCS.

Like mMCS, aMCSs build on an abstract notion of a *logic suite* which can be seen as an abstraction of different formalisms for knowledge representation. A logic suite is a triple $\mathcal{LS} = \langle \mathcal{KB}, \mathcal{BS}, \mathcal{ACC} \rangle$, where $\mathcal{KB}$ is the set of admissible knowledge bases (KBs) of $\mathcal{LS}$. Each knowledge base is a set of formulas that we do not further specify. $\mathcal{BS}$ is the set of possible belief sets of $\mathcal{LS}$, whose elements are beliefs. A semantics for $\mathcal{LS}$ is a function $\mathrm{ACC} : KB \to 2^{\mathcal{BS}}$ assigning to each KB a set of acceptable belief sets. Using a semantics with potentially more than one acceptable belief set allows for modelling non-determinism, where each belief set corresponds to an alternative solution. Finally, $\mathcal{ACC}$ is a set of semantics for $\mathcal{LS}$. We denote $\mathcal{KB}$,$\mathcal{BS}$, respectively, $\mathcal{ACC}$ by $\mathcal{KB}_{\mathcal{LS}}$,$\mathcal{BS}_{\mathcal{LS}}$, respectively, $\mathcal{ACC}_{\mathcal{LS}}$.

The motivation behind having multiple semantics for one formalism is that in our framework, the semantics of a formalism can be changed over time. While it is probably rarely the case that one wants to switch between different families of semantics during a run, e.g., from the stable-model semantics to the well-founded semantics of logic programs other switches of semantics are quite natural to many applications: we use different semantics to express different reasoning modes or to express different queries, i.e., ACC_1 returns belief sets answering a query q_1, whereas ACC_2 answers query q_2; ACC_3, in turn, could represent the computation of all solutions to a problem, whereas at some point in time one could be interested in using ACC_4 that only computes a single solution. For instance one that is optimal with respect to some criterion.

3 Asynchronous Multi-Context Systems

An aMCS is built up by multiple contexts which are defined next and which are used for representing reasoning units. We assume a set $\mathcal{N}$ of names that will serve as labels for sensors, contexts, and output streams.

Definition 1 *A context is a pair $C = \langle \mathsf{n}, \mathcal{LS} \rangle$ where $\mathsf{n} \in \mathcal{N}$ is the name of the context and $\mathcal{LS}$ is a logic suite.*

For a given context $C = \langle \mathsf{n}, \mathcal{LS} \rangle$ we denote n and $\mathcal{LS}$ by n_C and $\mathcal{LS}_C$, respectively.

Definition 2 *An aMCS (of length n with m output streams) is a pair $M = \langle \mathsf{C}, \mathsf{O} \rangle$, where $\mathsf{C} = \langle C_1, \dots, C_n \rangle$ is an n-tuple of contexts and $\mathsf{O} = \langle \mathsf{o}_1, \dots, \mathsf{o}_m \rangle$ with $\mathsf{o}_j \in \mathcal{N}$ for each $1 \leq j \leq m$ is a tuple containing the names of the output streams of M.*

By $\mathcal{N}(M)$ we denote the set $\{\mathsf{n}_{C_1}, \dots, \mathsf{n}_{C_n}, \mathsf{o}_1, \dots, \mathsf{o}_m\}$ of names of contexts and output streams of M.

A context in an aMCS communicates with other contexts and the outside world by means of streams of data. In particular, we assume that every context has an input stream on which information can be written from both external sources (we call them sensors) and internal sources (i.e., other contexts). For the data in the communication streams we assume a communication language $\mathcal{IL}$ where every $\mathsf{i} \in \mathcal{IL}$ is an abstract *piece of information*. In our framework, the data in the input stream of a context and the data in output streams are modelled by information buffers that are defined in the following.

Definition 3 *A* data package *is a pair $d = \langle s, I \rangle$, where $s \in \mathcal{N}$ is either a context name or a sensor name, stating the* source *of d, and $I \subseteq \mathcal{IL}$ is a set of pieces of information. An* information buffer *is a sequence of data packages.*

As we assume that data is asynchronously passed to a context on its input stream, it is natural that not all information required for a computation is available at all times. Consequently, we need means to decide whether a computation should take place, depending on the current KB and the data currently available on the stream, or whether the context has to wait for more data. In our framework, this decision is made by a computation controller as defined next.

Definition 4 *Let $C = \langle \mathsf{n}, \mathcal{LS} \rangle$ be a context. A* computation controller *for C is a relation* cc *between a KB* $\mathrm{KB} \in \mathcal{KB}_{\mathcal{LS}}$ *and a finite information buffer.*

Thus, if $\langle \mathrm{KB}, \mathrm{ib} \rangle \in \mathrm{cc}$ then a computation should take place, whereas $\langle \mathrm{KB}, \mathrm{ib} \rangle \notin \mathrm{cc}$ means that further information is required before the next computation is triggered in the respective context.

In contrast to the original definition of multi-context systems [1] and extensions thereof, we do not make use of so-called *bridge rules* as a means to communicate: a bridge rule defines which information a context should obtain based on the results of all the contexts of a multi-context system. In the asynchronous approach, we do not have (synchronised) results of all contexts available in general. As a consequence we use another type of rules, called *output rules*, that define which information should be sent to another context or an output stream, based on a result of a single context.

Definition 5 *Let $C = \langle \mathsf{n}, \mathcal{LS} \rangle$ be a context. An* output rule *r for C is an expression of the form*

$$\langle \mathsf{n}, \mathsf{i} \rangle \leftarrow b_1, \dots, b_j, \mathrm{not}\ b_{j+1}, \dots, \mathrm{not}\ b_m, \tag{1}$$

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

We call n the *stakeholder* of r, $\langle \mathsf{n}, \mathsf{i} \rangle$ the head of r denoted by $hd(r)$ and $b_1, \dots, b_j, \mathrm{not}\ b_{j+1}, \dots, \mathrm{not}\ b_m$ the body $\mathrm{bd}(r)$ of r. Moreover, we say that r is active under S, denoted by $S \models \mathrm{bd}(r)$, if $\{b_1, \dots, b_j\} \subseteq S$ and $\{b_{j+1}, \dots, b_m\} \cap S = \emptyset$.

Intuitively, the stakeholder is a reference to the addressee of information i.

Definition 6 *Let $C = \langle \mathsf{n}, \mathcal{LS} \rangle$ be a context,* OR *a set of output rules for C, $S \in \mathcal{BS}_{\mathcal{LS}}$ a belief set, and $\mathsf{n}' \in \mathcal{N}$ a name. Then, the data package*

$$d_C(S, \mathrm{OR}, \mathsf{n}') = \langle \mathsf{n}, \{i \mid r \in \mathrm{OR}, hd(r) = \langle \mathsf{n}', \mathsf{i} \rangle, S \models \mathrm{bd}(r)\} \rangle$$

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

Compared to previous notions of multi-context systems, contexts in our setting only specify which formalisms they use but they do not contain knowledge bases, the concrete semantics to use, and communication specifications. The reason is that for aMCSs these may change over time. Instead, we wrap concepts that are subject to change during runtime in the following notion of a *configuration*.

Definition 7 *Let $C = \langle \mathsf{n}, \mathcal{LS}\rangle$ be a context. A* configuration *of C is a tuple $cf = \langle \mathrm{KB}, \mathrm{ACC}, \mathrm{ib}, cm\rangle$, where $\mathrm{KB} \in \mathcal{KB}_{\mathcal{LS}}$, $\mathrm{ACC} \in \mathcal{ACC}_{\mathcal{LS}}$, ib is a finite information buffer, and cm is a* context management *for C which is a triple $cm = \langle \mathrm{cc}, \mathrm{cu}, \mathrm{OR}\rangle$, where*

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

We write cc_{cm}, cu_{cm}, and OR_{cm} to refer to the components of a given context management $cm = \langle \mathrm{cc}, \mathrm{cu}, \mathrm{OR}\rangle$. The context management is the counterpart of a *management function* of an rMCS, that computes an update of the knowledge base of a context given the results of bridge rules of the context.

In Section 2 we already discussed why we want to change semantics over time. Allowing also for changes of output rules can be motivated with applications where it should be dynamically decided where to direct the output of a context. For example, if a particular subproblem can be solved by two contexts C_1 and C_2 and it is known that some class of instances can be better solved by C_1 and others by C_2. Then a third context that provides an instance can choose whether C_1 or C_2 should carry out the computation by adapting its output rules. Dynamically changing output rules and semantics could require adjustments of the other components of the context management. Thus, it makes sense that also compution controllers and context update functions are subject to change for the sake of flexibility.

Definition 8 *Let $M = \langle\langle C_1, \dots, C_n\rangle, \langle \mathsf{o}_1, \dots, \mathsf{o}_m\rangle\rangle$ be an aMCS. A* configuration *of M is a pair*

$$Cf = \langle\langle cf_1, \dots, cf_n\rangle, \langle \mathrm{ob}_1, \dots, \mathrm{ob}_m\rangle\rangle,$$

where

- *for all $1 \leq i \leq n$ $cf_i = \langle \mathrm{KB}, \mathrm{ACC}, \mathrm{ib}, cm\rangle$ is a configuration for C_i and for every output rule $r \in \mathrm{OR}_{cm}$ we have $\mathsf{n} \in \mathcal{N}(M)$ for $\langle \mathsf{n}, \mathsf{i}\rangle = hd(r)$, and*
- *$\mathrm{ob}_j = \dots, d_{l-1}, d_l$ is an information buffer with a final element d_l that corresponds to the data on the output stream named o_j for each $1 \leq j \leq m$ such that for each $h \leq l$ with $d_h = \langle \mathsf{n}, \mathsf{i}\rangle$ we have $\mathsf{n} = \mathsf{n}_{C_i}$ for some $1 \leq i \leq n$.*

Figure 1 depicts an aMCS M with three contexts and a configuration for M.

We next characterise the dynamic behaviour of an aMCS. For easier notation we stick to a discrete notion of time represented by integers.

Definition 9 *Let $M = \langle\langle C_1, \dots, C_n\rangle, \langle \mathsf{o}_1, \dots, \mathsf{o}_m\rangle\rangle$ be an aMCS. A run structure for M is a sequence*

$$R = \dots, Cf^t, Cf^{t+1}, Cf^{t+2}, \dots \quad ,$$

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

We will sometimes use cf_i^t to denote the configuration of a context i that appears at time t in a given run structure in the context of a given aMCS. Similarly, ob_j^t refers to the information buffer representing the data in the output stream named o_j. Moreover, we write KB_i^t, ACC_i^t, ib_i^t, and cm_i^t to refer to the components of $cf_i^t = \langle \mathrm{KB}, \mathrm{ACC}, \mathrm{ib}, cm\rangle$. We say that context C_i is *waiting* at time t if $\langle \mathrm{KB}_i^t, \mathrm{ib}_i^t\rangle \notin \mathrm{cc}_{cm_i^t}$.

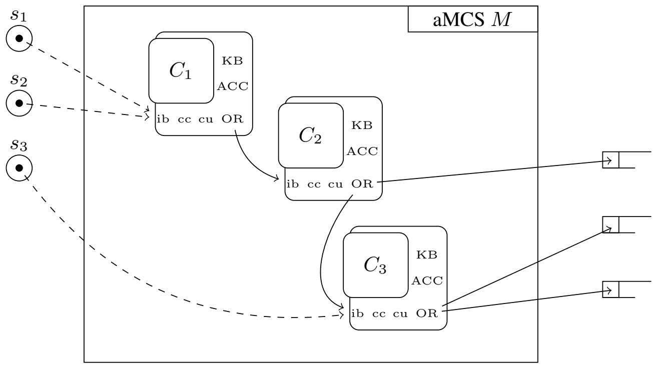


Figure 1. An aMCS with three contexts, three sensors on the left side, and three output streams on the right side. A solid line represents a flow of information from a context to its stakeholder streams, whereas a dashed line indicates sensor data written to the input buffer of a context.

From run structure to run In aMCSs we take into account that the computation of the semantics of a knowledge base needs time. Moreover, in a computation of our framework, different belief sets may become available at different times and verifying the non-existence of further belief sets can also take time after the final belief set has been computated. In order to model whether a context is busy with computing, we introduce a boolean variable $busy_i^t$ for each configuration cf_i^t in a run structure. Hence, context C_i is *busy* at time t iff $busy_i^t$ is true. While a context is busy, it does not read new information from its input stream until every belief set has been computed and it has concluded that no further belief set exists.

After the computation of a belief set, the output rules are applied in order to determine which data is passed on to stakeholder contexts or output streams. These are represented by *stakeholder buffers*: An information buffer b is the stakeholder buffer of C_i (for n) at time t if

- $\mathrm{b} = \mathrm{ib}_{i'}^t$ for some $1 \leq i' \leq n$ such that $\mathsf{n} = \mathsf{n}_{C_i}$ is stakeholder of some output rule in $\mathrm{OR}_{cm_i^t}$ or
- $\mathrm{b} = \mathrm{ob}_{j'}^t$ for some $1 \leq j' \leq m$ such that $\mathsf{n} = \mathsf{o}_{j'}$ is stakeholder of some output rule in $\mathrm{OR}_{cm_i^t}$.

In order to indicate that a computation has finished we assume a dedicated symbol $\mathrm{EOC} \in \mathcal{IL}$ that notifies a context's stakeholder buffers about the end of a computation.

Next, we formally characterise the behaviour of aMCSs followed by a summary of its intuition.

Definition 10 *Let M be an aMCS of length n with m output streams and R a run structure for M. R is a* run *for M if the following conditions hold for every $1 \leq i \leq n$ and every $1 \leq j \leq m$:*

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

- *C_i is busy at time $t+1$,*
- *$cf_i^{t+1} = \mathrm{cu}_{cm_i^t}(\mathrm{ib}_i^t, \mathrm{KB}_i^t)$*

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

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

- *we say that this computation ended at time t', if t' is the earliest time point with $t' \geq t$ such that $\langle \mathsf{n}_{C_i}, \mathrm{EOC}\rangle$ is added to every stakeholder buffer b of C_i at t'; the addition of*

$d_{C_i}(S, \mathrm{OR}_{cm_i^{t''}}, \mathsf{n})$ *to* b *is called an* end of computation notification.

– *for all $t' > t$ such that $cf_i^{t'}$ is defined, C_i is busy at t' unless the computation ended at some time t'' with $t < t'' < t'$.*

– *if the computation ended at time t' and $cf_i^{t'+1}$ is defined then C_i is not busy at $t' + 1$.*

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

– $d_{C_i}(S, \mathrm{OR}_{cm_i^{t''}}, \mathsf{n})$ *is added to every stakeholder buffer* b *of C_i for* n *at t''.*

We say that C_i computed *S at time t''. The addition of* $d_{C_i}(S, \mathrm{OR}_{cm_i^{t''}}, \mathsf{n})$ *to* b *is called a* belief set notification.

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

(v) *if cf_i^t and cf_i^{t+1} are defined, C_i is busy or waiting at time t, and $\mathrm{ib}_i^t = d_1, \ldots, d_l$ then we have $\mathrm{ib}_i^{t+1} = d_1, \ldots, d_l, \ldots, d_{l'}$ for some $l' \geq l$. Moreover, every data package $d_{l''}$ with $l < l'' \leq l'$ that was added at time $t + 1$ either results from an end of computation notification or a belief set notification or $\mathsf{n} \notin \mathcal{N}(M)$ (i.e.,* n *is a sensor name) for $d_{l''} = \langle \mathsf{n}, \mathsf{i} \rangle$.*

Condition (i) describes the transition from an idle phase to an ongoing computation. The end of such a compation is marked by an end of computation notification as introduced in Item (ii). Condition (iii) states that between the start and the end of a computation all belief sets are computed and stakeholders are notified. Items (iv) and (v) express how data is added to an output stream or to an input stream, respectively. Note that here, sensors and the flow of information from sensors to the input buffers of contexts are implicit. That is, data packages from a sensor may appear at the end of input buffers at all times and the only reference to a particular sensor is its name appearing in a data package.

Summarising the behaviour characterised by a run, whenever a context C is not busy, its context controller cc checks whether a new computation should take place, based on the knowledge base and the current input buffer of C. If yes, the current configuration of the context is replaced by a new one, computed by the context update function cu of C. Here, the new input buffer has to be a suffix of the old one and a new computation for the updated knowledge base starts. After an undefined period of time, belief sets are computed and based on the application of output rules of C, data packages are sent to stakeholder buffers. At some point in time, when all belief sets have been computed, an end of computation notification is sent to stakeholders, and the context is not busy anymore.

4 Scenario: Computer-Aided Emergency Team Management

Now we want to consider a scenario, where aMCSs may be used to describe the asynchronous information-exchange between different specialised reasoning systems. Our example deals with a recommender-system for the coordination and handling of ambulance assignments. The suggested aMCS supports decisions in various stages of an emergency case. It gives assistance during the rescue call, helps in assigning priorities and rescue units to a case, and assists in the necessary communication among all involved parties. The suggestions given by the system are based on different specialised systems which react to sensor readings. Moreover, the system can tolerate and incorporate overriding solutions proposed by the user that it considers non-optimal.

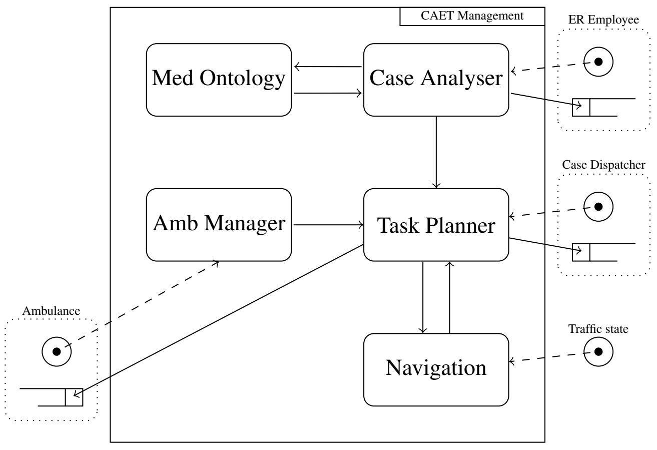


Figure 2. The Computer-Aided Emergency Team Management aMCS

Figure 2 depicts the example aMCS which models such a *Computer-Aided Emergency Team Management System* (CAET Management System). Note that interaction with a human (e.g., EM employee) is modelled as a pair containing an input stream and an output stream. The system consists of the following contexts:

Case Analyser (ca) This context implements a computer-aided call handling system which assists an emergency response employee (ER employee) during answering an emergency call. The system utilises reasoning methods to choose which questions need to be asked based on previous answers. In addition, it may check whether answers are inconsistent (e.g., amniotic sac bursts when the gender is male). For these purposes the case analyser context may also consult a medical ontology represented by another context. The communication with the ER employee is represented, on the one hand, as a sensor that reads the input of the employee and, on the other hand, by an output stream which prints the questions and results on a computer screen.
During the collection of all the important facts for this emergency case, the analyser computes the priority of the case and passes it to the task planner.

Med Ontology (mo) This medical ontology can be realised, e.g., by a description logic reasoner which handles requests from the case analyser and returns more specific knowledge about ongoing cases. This information may be used for the prioritisation of the importance of a case.

Task Planner (tp) This context keeps track of emergency cases. Based on the priority and age of a case and the availability and position of ambulances it suggests an efficient plan of action for the ambulances to the (human) case dispatcher (cd). The dispatcher may approve some of the suggestions or all of them. If the dispatcher has no faith in the given plan of action, she can also alter it at will. These decisions are reported back to the planning sys-

tem such that it can react to the alterations and provide further suggestions. Based on the final plan, the task planner informs the ambulance about their new mission.
The knowledge base of the context is an answer-set program for reasoning about a suggested plan. It gets the availability and position of the ambulances by the ambulance manager. In addition, the cases with their priority are provided by the case analyser. With this information, the task planner gives the locations of the ambulances together with the target locations of the cases to a navigation system which provides the distances (i.e., the estimated time of arrival (ETA)) of all the ambulances to all the locations.

Amb Manager (am) The ambulance manager is a database, which keeps track of the status and location of ambulance units. Each ambulance team reports its status (e.g., to be on duty, waiting for new mission, ...) to the database (modelled by the sensor "Ambulance" (amb)). Additionally, the car periodically sends GPS-coordinates to the database. These updates will be pushed to the task planner.

Navigation (na) This part of the aMCS gets traffic information (e.g., congestions, roadblocks, construction zones, ...) to predict the travel time for each route as accurate as possible. The task planner may push a query to the navigation system, which consists of a list of locations of ambulance units and a list of locations of target areas. Based on all the given information this context will return a ranking for each target area, representing the ETAs for each ambulance.

Now we want to have a closer look on the instantiation details of some aspects of our example. At first we investigate the cc relation of the case analyser. It allows for the computation of new belief sets whenever the ER employee pushes new information to the analyser. In addition, it will also approve of a new computation if the medical ontology supplies some requested information. Recall that the case analyser also assigns a priority to each case and that we want to allow the employee to set the priority manually. Let us suppose that such a manual override occurs and that the case analyser has an ongoing query to the medical ontology. Due to the manual priority assignment, the requested information from the ontology is no longer needed. Therefore, it would be desirable that cc does not allow for a recomputation if all conclusions of the ontology are only related to the manually prioritised case. With the same argumentation in mind, the context update function cu will also ignore this information on the input stream. This kind of behaviour may need knowledge about past queries which can be provided by an additional output rule for the case analyser which feeds the relevant information back to the context.

Next, we will have a look at the task planner that is based on answer-set programming. We will only present parts of the program, to show how the mechanics are intended to work. To represent the incoming information on the input stream, the following predicates can be used:

`case(caseid,loc,priority)` represents an active case (with its location and priority) which needs to be assigned to an ambulance.

`avail(amb,loc)` states the location of an available ambulance.

`eta(caseid,amb,value)` provides the estimated time of arrival for a unit at the location of the target area of the case.

`assign(amb,caseid)` represents the assignment of an ambulance to a case by the dispatcher.

These predicates will be added by the context update function to the knowledge base if corresponding information is put on the input stream of the context. Based on this knowledge, the other components of the answer-set program will compute the belief sets (e.g., via the stable model semantics). Note that an already assigned ambulance or case will not be handled as an available ambulance or an active case, respectively. In addition, cu can (and should) also manage forgetting of no longer needed knowledge. For our scenario it may be suitable to remove all `eta`, `avail` and `case` predicates when the cases or the unit is assigned. The `assign` predicate can be removed when the ambulance manager reports that the assigned ambulance is available again.

The set OR of output rules of the task planner could contain the following rules:[3]

$$\begin{aligned}
\langle \text{cd,assign}(A,C)\rangle &\leftarrow \quad \text{sugassignment}(A,C)\\
\langle \text{na,queryA}(L)\rangle &\leftarrow \quad \text{avail}(A), \text{not assign}(A,\_), \text{loc}(A,L)\\
\langle \text{na,queryC}(L)\rangle &\leftarrow \quad \text{case}(C,P), \text{loc}(A,L), \text{not assign}(A,\_)\\
\langle \text{amb,assigned}(A,C)\rangle &\leftarrow \quad \text{assign}(A,C)
\end{aligned}$$

The first rule informs the case dispatcher (cd) about a suggested assignment that has been computed by the answer-set program. Rules two and three prepare lists of ambulances and cases for querying the navigation context. Recall that the latter needs a list of ambulance locations (generated by rule two) and a list of target area locations (generated by rule three). Also keep in mind that for each belief set a data package with all information for one context or output stream is constructed. So the whole list of current target areas and free ambulance units will be passed to the navigation context at once. The last rule notifies the ambulance team that it has been assigned to a specific case.

Related to this example we want to mention privacy aspects as a real world policy which is especially important to applications in public services and health care. As the multi-context system is a heterogeneous system with different contexts, a completely free exchange of data may be against privacy policies. This issue can be addressed by the adequate design of output rules, which can also be altered with respect to additional information in the input stream (e.g., some context gains the permission to receive real names instead of anonymous data). So each context may decide by its own which parts of the belief sets are shared and exchanged with other contexts.

Another interesting aspect about aMCSs is the possibility to easily join two aMCSs together, outsource a subset of contexts in a new aMCS, or to view an aMCS as an abstract context for another aMCS in a modular way. This can be achieved due to the abstract communication by means of streams. With respect to our scenario there could be some aMCS which does the management of resources for hospitals (e.g., free beds with their capabilities). The task planner might communicate with this system to take the needed services for a case into account (e.g., intensive care unit) and informs the hospital via these streams about incoming patients. It would be easy to join both aMCSs together to one big system or to outsource some contexts as input sensors paired with an output stream. In addition, one may also combine different contexts or a whole aMCS to one abstract context to provide a dynamic granularity of information about the system and to group different reasoning tasks together.

5 Relation to Reactive Multi-Context Systems

In this section we want to address differences and communalities between aMCSs and rMCSs [4] as both are types of multi-context

[3] Keep in mind that in an actual implementation one may want to provide further information via communication.

systems that work in an online fashion and can react to external information. Runs of rMCSs are based on equilibria which are collections of belief sets—one for each context—on which, intuitively, all of the contexts have to agree. Thus, equilibria can be seen as a tight integration approach in which the semantics of the individual contexts are interdependent. However, the high level of integration also comes at the price that the different contexts must wait for each other for the computation of each equilibrium, i.e., they are synchronised. In aMCSs, on the other hand, the coupling of the semantics is much looser—communication between contexts only works via data packages that are sent to another context after a computation and not via a higher-level common semantics for multiple contexts. But as a benefit, each context can run at its own pace which is useful in settings where there is a context that requires much more time for evaluating its semantics than others.

A further difference is the role of non-determinism in the semantics of aMCSs and rMCSs. An equilibrium in an aMCS consists of a single belief set for each context. Hence, as aMCSs also use a multiple belief set semantics, there may also be multiple equilibria as a source of non-determinism at each step in a run. For aMCSs, all belief sets of a context are computed in a consecutive way (we assume that if only a single belief set is desired than the semantics of the respective context should be adapted accordingly by the knowledge engineer). Nevertheless, there is also a source of non-determinism in the case of aMCSs caused by the undefined duration of computations.

Regarding the computational complexity of the two frameworks, the computation of an equilibrium requires to guess an equilibrium candidate first before the semantics of the context is computed which is expensive regarding runtime when put to practice. In theory, this guess does not add extra complexity if the context semantics is already $\mathbf{NP}$-hard (as shown in [4]) because it can be combined with the guesses required in the contexts. However, this trick cannot be used in implementations that uses black boxes for computing context semantics. On the other hand, aMCSs do not add substantial computational requirements to the effort needed for computing context semantics. In particular, aMCSs are scalable as adding a further context has no direct influence on how the semantics of the other contexts are computed but can only influence the input they get.

Both, aMCSs and rMCSs are very general frameworks that allow for simulating Turing machines and thus for performing multi-purpose computations even if only very simple context formalisms are used (if the length of a run is not restricted). In this sense the approaches are equally expressive. Moreover, when allowing for arbitrary contexts one could trivially simulate the other by including it as a context. Despite the existence of these straightforward translations, we next sketch how we simulate an rMCS with an aMCS using a more direct translation, as this gives further insight into the differences of the two frameworks. Moreover, it demonstrates a way to implement rMCSs by means of aMCSs. For every context C_i of a given rMCS M_r, we introduce three contexts in the aMCS M_a that simulates M_r:

- a context C_i^{kb} that stores the current knowledge base of the context,
- a context $C_i^{kb'}$ in which a candidate for an updated knowledge base can be written and its semantics can be computed, and
- a management context C_i^m that implements the bridge rules, and the management function of the context.

There are three further contexts:

- C^{obs} receives sensor data and distributes it to every context C_i^m where C_i depends on the respective sensor. The context is also responsible for synchronisation: for each sensor, new sensor data is only passed on after an equilibrium has been computed.
- C^{guess} guesses equilibrium candidates for M and passes them to the management contexts C_i^m. Based on that and the information from C^{obs}, C_i^m computes an update kb'_i of the knowledge base in C_i^{kb} and stores kb'_i in $C_i^{kb'}$. The latter context then computes the semantics of kb'_i and passes it to the final context
- C^{check} that compares every belief set it receives with the equilibrium candidate (that it also receives from C^{guess}). If a matching belief set has been found for each context of M_r, the candidate is an actual equilibrium. In this case C^{check} sends the equilibrium to an output stream and notifies the other contexts about the success.

In case of a success, every context C_i^m replaces the knowledge base in C_i^{kb} by kb_i and a next iteration begins. In case no equilibrium was found but one of the $C_i^{kb'}$ contexts has finished its computation, C^{check} orders C^{guess} to guess another equilibrium candidate.

6 Related Work and Discussion

A concept similar to output-rules has been presented in the form of reactive bridge rules [6]. There the flow of information is represented by rules which add knowledge to the input streams of other contexts. Which information is communicated to other contexts is also determined by the local belief set of each context.

Note that evolving multi-context systems [8] follow a quite similar approach as rMCSs and hence the relation of aMCSs to rMCSs sketched in the previous section also applies in essence to this approach.

The system `clingo` [7] is a reactive answer-set programming solver. It utilises TCP/IP ports for incoming input streams and does also report the resulting answer sets via such a port. It provides means to compute different semantics and can keep learned structures and knowledge from previous solving steps. Although there are no output rules or input stream pre-processing as in aMCSs, the system features embedded imperative programming languages which may be helpful to model some of the presented concepts of this paper.

In general, the tasks performed by a context management can be realised by different formalisms (e.g., imperative scripting languages or declarative programming). Here, it seems likely that different languages can be the most appropriate management language, depending on the type of context formalism and the concrete problem domain. A feature that is not modelled in our proposal but that is potentially useful and we intend to consider in the future is to allow for aborting computations. Moreover, we want to study modelling patterns and best practices for aMCSs design for typical application settings and compare different inter-context topologies and communication strategies.

The next natural step towards an implementation is an analysis of how existing tools such as `clingo` could be used for a realisation. It is clear that such formalisms can be used as a context formalism. Moreover, we are interested in how reactive features of `clingo` (e.g., iterative computation, on-demand grounding, online-queries, ...) relate to aMCS concepts (e.g., cc, ib, ...) and whether the system can be described in terms of an aMCS.

REFERENCES

[1] Gerhard Brewka and Thomas Eiter, 'Equilibria in heterogeneous non-monotonic multi-context systems', in *AAAI'07*, pp. 385–390, (2007).
[2] Gerhard Brewka, Thomas Eiter, and Michael 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).

[3] Gerhard Brewka, Thomas Eiter, Michael Fink, and Antonius Weinzierl, ‘Managed multi-context systems’, in *IJCAI’11*, pp. 786–791, (2011).

[4] Gerhard Brewka, Stefan Ellmauthaler, and Jörg Pührer, ‘Multi-context systems for reactive reasoning in dynamic environments’, in *Proc. ECAI’14*, (2014). To appear.

[5] Fu-Leung Cheng, Thomas Eiter, Nathan Robinson, Abdul Sattar, and Kewen Wang, ‘Lpforget: A system of forgetting in answer set programming’, in *Proc. AUSAI’06*, eds., Abdul Sattar and Byeong Ho Kang, volume 4304 of *LNCS*, pp. 1101–1105. Springer, (2006).

[6] Stefan Ellmauthaler, ‘Generalizing multi-context systems for reactive stream reasoning applications’, in *Proc. ICCSW’13*, pp. 17–24, (2013).

[7] Martin Gebser, Torsten Grote, Roland Kaminski, Philipp Obermeier, Orkunt Sabuncu, and Torsten Schaub, ‘Stream reasoning with answer set programming: Preliminary report’, in *Proc. KR’12*, (2012).

[8] Ricardo Gonçalves, Matthias Knorr, and João Leite, ‘Evolving multi-context systems’, in *Proc. ECAI’14*, (2014). To appear.

[9] Jérôme Lang and Pierre Marquis, ‘Reasoning under inconsistency: A forgetting-based approach’, *Artif. Intell.*, **174**(12-13), 799–823, (2010).

[10] Danh Le-Phuoc, Josiane Xavier Parreira, and Manfred Hauswirth, ‘Linked stream data processing’, in *Proc. RW’12*, eds., Thomas Eiter and Thomas Krennwallner, volume 7487 of *LNCS*, pp. 245–289. Springer, (2012).

[11] Carlo Zaniolo, ‘Logical foundations of continuous query languages for data streams’, in *Proc. Datalog 2.0*, pp. 177–189, (2012).

Towards Efficient Evolving Multi-Context Systems (Preliminary Report)

Ricardo Gonçalves and **Matthias Knorr** and **João Leite** [1]

Abstract. Managed Multi-Context Systems (mMCSs) provide a general framework for integrating knowledge represented in heterogeneous KR formalisms. Recently, evolving Multi-Context Systems (eMCSs) have been introduced as an extension of mMCSs that add the ability to both react to, and reason in the presence of commonly temporary dynamic observations, and evolve by incorporating new knowledge. However, the general complexity of such an expressive formalism may simply be too high in cases where huge amounts of information have to be processed within a limited short amount of time, or even instantaneously. In this paper, we investigate under which conditions eMCSs may scale in such situations and we show that such polynomial eMCSs can be applied in a practical use case.

1 Introduction

Multi-Context Systems (MCSs) were introduced in [7], building on the work in [16, 27], to address the need for a general framework that integrates knowledge bases expressed in heterogeneous KR formalisms. Intuitively, instead of designing a unifying language (see e.g., [17, 26], and [23] with its reasoner NoHR [22]) to which other languages could be translated, in an MCS the different formalisms and knowledge bases are considered as modules, and means are provided to model the flow of information between them (cf. [1, 21, 24] and references therein for further motivation on hybrid languages and their connection to MCSs).

More specifically, an MCS consists of a set of contexts, each of which is a knowledge base in some KR formalism, such that each context can access information from the other contexts using so-called bridge rules. Such non-monotonic bridge rules add its head to the context's knowledge base provided the queries (to other contexts) in the body are successful. Managed Multi-Context Systems (mMCSs) were introduced in [8] to provide an extension of MCSs by allowing operations, other than simple addition, to be expressed in the heads of bridge rules. This allows mMCSs to properly deal with the problem of consistency management within contexts.

One recent challenge for KR languages is to shift from static application scenarios which assume a one-shot computation, usually triggered by a user query, to open and dynamic scenarios where there is a need to react and evolve in the presence of incoming information. Examples include EVOLP [2], Reactive ASP [14, 13], C-SPARQL [5], Ontology Streams [25] and ETALIS [3], to name only a few.

Whereas mMCSs are quite general and flexible to address the problem of integration of different KR formalisms, they are essentially static in the sense that the contexts do not evolve to incorporate the changes in the dynamic scenarios. In such scenarios, new knowledge and information is dynamically produced, often from several different sources – for example a stream of raw data produced by some sensors, new ontological axioms written by some user, newly found exceptions to some general rule, etc.

To address this issue, two recent frameworks, evolving Multi-Context Systems (eMCSs) [19] and reactive Multi-Context Systems (rMCSs) [6, 12, 9] have been proposed sharing the broad motivation of designing general and flexible frameworks inheriting from mMCSs the ability to integrate and manage knowledge represented in heterogeneous KR formalisms, and at the same time be able to incorporate knowledge obtained from dynamic observations.

Whereas some differences set eMCSs and rMCSs apart (see related work in Sec. 6), the definition of eMCSs is presented in a more general way. That, however, means that, as shown in [19], the worst-case complexity is in general high, which may be problematic in dynamic scenarios where the overall system needs to evolve and react interactively. This is all the more true for huge amounts of data – for example raw sensor data is likely to be constantly produced in large quantities – and systems that are capable of processing and reasoning with such data are required.

At the same time, eMCSs inherit from MCSs the property that models, i.e., equilibria, may be non-minimal, which potentially admits that certain pieces of information are considered true based solely on self-justification. As argued in [7], minimality may not always be desired, which can in principle be solved by indicating for each context whether it requires minimality or not. Yet, avoiding self-justifications for those contexts where minimality is desired has not been considered in eMCSs.

In this paper, we tackle these problems and, in particular, consider under which conditions reasoning with evolving Multi-Context Systems can be done in polynomial time. For that purpose, we base our work on a number of notions studied in the context of MCSs that solve these problems in this case [7]. Namely, we adapt the notions of minimal and grounded equilibria to eMCSs, and subsequently a well-founded semantics, which indeed paves the way to the desired result.

The remainder of this paper is structured as follows. After introducing the main concepts regarding mMCSs in Sect. 2, in Sect. 3 we recall with more detail the framework of eMCSs already introducing adjustments to achieve polynomial reasoning. Then, in Sect. 4 we present an example use case, before we adapt and generalize notions from MCSs in Sect. 5 as outlined. We conclude in Sect. 6 with discussing related work and possible future directions.

[1] CENTRIA & Departamento de Informática, Faculdade Ciências e Tecnologia, Universidade Nova de Lisboa, email: rjrg@fct.unl.pt

2 Preliminaries: Managed Multi-Context Systems

Following [7], a multi-context system (MCS) consists of a collection of components, each of which contains knowledge represented in some *logic*, defined as a triple $L = \langle \mathbf{KB}, \mathbf{BS}, \mathbf{ACC} \rangle$ where $\mathbf{KB}$ is the set of well-formed knowledge bases of L, $\mathbf{BS}$ is the set of possible belief sets, and $\mathbf{ACC} : \mathbf{KB} \to 2^{\mathbf{BS}}$ is a function describing the semantics of L by assigning to each knowledge base a set of acceptable belief sets. We assume that each element of $\mathbf{KB}$ and $\mathbf{BS}$ is a set, and define $F = \{s : s \in kb \wedge kb \in \mathbf{KB}\}$.

In addition to the knowledge base in each component, *bridge rules* are used to interconnect the components, specifying what knowledge to assert in one component given certain beliefs held in the components of the MCS. Bridge rules in MCSs only allow adding information to the knowledge base of their corresponding context. In [8], an extension, called managed Multi-Context Systems (mMCSs), is introduced in order to allow other types of operations to be performed on a knowledge base. For that purpose, each context of an mMCS is associated with a *management base*, which is a set of operations that can be applied to the possible knowledge bases of that context. Given a management base OP and a logic L, let $OF = \{op(s) : op \in OP \wedge s \in F\}$ be the *set of operational formulas* that can be built from OP and F. Each context of an mMCS gives semantics to operations in its management base using a *management function* over a logic L and a management base OP, $mng : 2^{OF} \times \mathbf{KB} \to \mathbf{KB}$, i.e., $mng(op, kb)$ is the knowledge base that results from applying the operations in op to the knowledge base kb. Note that this is already a specific restriction in our case, as mng commonly returns a (non-empty) set of possible knowledge bases for mMCS (and eMCS). We also assume that $mng(\emptyset, kb) = kb$. Now, for a sequence of logics $L = \langle L_1, \ldots, L_n \rangle$ and a management base OP_i, an *L_i-bridge rule σ over L*, $1 \leq i \leq n$, is of the form $H(\sigma) \leftarrow B(\sigma)$ where $H(\sigma) \in OF_i$ and $B(\sigma)$ is a set of *bridge literals* of the forms $(r : b)$ and $\mathbf{not}\,(r : b)$, $1 \leq r \leq n$, with b a belief formula of L_r.

A *managed Multi-Context System* (mMCS) is a sequence $M = \langle C_1, \ldots, C_n \rangle$, where each C_i, $i \in \{1, \ldots, n\}$, called a *managed context*, is defined as $C_i = \langle L_i, kb_i, br_i, OP_i, mng_i \rangle$ where $L_i = \langle \mathbf{KB}_i, \mathbf{BS}_i, \mathbf{ACC}_i \rangle$ is a logic, $kb_i \in \mathbf{KB}_i$, br_i is a set of L_i-bridge rules, OP_i is a management base, and mng_i is a management function over L_i and OP_i. Note that, for the sake of readability, we consider a slightly restricted version of mMCSs where $\mathbf{ACC}_i$ is still a function and not a set of functions as for logic suites [8].

For an mMCS $M = \langle C_1, \ldots, C_n \rangle$, a *belief state of M* is a sequence $S = \langle S_1, \ldots, S_n \rangle$ such that each S_i is an element of $\mathbf{BS}_i$. For a bridge literal $(r : b)$, $S \models (r : b)$ if $b \in S_r$ and $S \models \mathbf{not}\,(r : b)$ if $b \notin S_r$; for a set of bridge literals B, $S \models B$ if $S \models L$ for every $L \in B$. We say that a bridge rule σ of a context C_i is *applicable given a belief state S of M* if S satisfies $B(\sigma)$. We can then define $app_i(S)$, the set of heads of bridge rules of C_i which are applicable in S, by setting $app_i(S) = \{H(\sigma) : \sigma \in br_i \wedge S \models B(\sigma)\}$.

Equilibria are belief states that simultaneously assign an acceptable belief set to each context in the mMCS such that the applicable operational formulas in bridge rule heads are taken into account. Formally, a belief state $S = \langle S_1, \ldots, S_n \rangle$ of an mMCS M is an *equilibrium* of M if, for every $1 \leq i \leq n$, $S_i \in \mathbf{ACC}_i(mng_i(app_i(S), kb_i))$.

3 Evolving Multi-Context Systems

In this section, we recall evolving Multi-Context Systems as introduced in [19] including some alterations that are in line with our intentions to achieve polynomial reasoning. As indicated in [19], we consider that some of the contexts in the MCS become so-called *observation contexts* whose knowledge bases will be constantly changing over time according to the observations made, similar, e.g., to streams of data from sensors.[2]

The changing observations then will also affect the other contexts by means of the bridge rules. As we will see, such effect can either be instantaneous and temporary, i.e., limited to the current time instant, similar to (static) mMCSs, where the body of a bridge rule is evaluated in a state that already includes the effects of the operation in its head, or persistent, but only affecting the next time instant. To achieve the latter, we extend the operational language with a unary meta-operation $next$ that can only be applied on top of operations.

Definition 1 *Given a management base OP and a logic L, we define eOF, the evolving operational language, as $eOF = OF \cup \{next(op(s)) : op(s) \in OF\}$.*

We can now define evolving Multi-Context Systems.

Definition 2 *An* evolving Multi-Context System (eMCS) *is a sequence $M_e = \langle C_1, \ldots, C_n \rangle$, where each* evolving context *C_i, $i \in \{1, \ldots, n\}$ is defined as $C_i = \langle L_i, kb_i, br_i, OP_i, mng_i \rangle$ where*

- *$L_i = \langle \mathbf{KB}_i, \mathbf{BS}_i, \mathbf{ACC}_i \rangle$ is a logic*
- *$kb_i \in \mathbf{KB}_i$*
- *br_i is a set of L_i-bridge rules s.t. $H(\sigma) \in eOF_i$*
- *OP_i is a management base*
- *mng_i is a management function over L_i and OP_i.*

As already outlined, evolving contexts can be divided into regular *reasoning contexts* and special *observation contexts* that are meant to process a stream of observations which ultimately enables the entire eMCS to react and evolve in the presence of incoming observations. To ease the reading and simplify notation, w.l.o.g., we assume that the first ℓ contexts, $0 \leq \ell \leq n$, in the sequence $\langle C_1, \ldots, C_n \rangle$ are observation contexts, and, whenever necessary, such an eMCS can be explicitly represented by $\langle C_1^o, \ldots, C_\ell^o, C_{\ell+1}, \ldots, C_n \rangle$.

As for mMCSs, a *belief state for M_e* is a sequence $S = \langle S_1, \ldots, S_n \rangle$ such that, for each $1 \leq i \leq n$, we have $S_i \in \mathbf{BS}_i$.

Recall that the heads of bridge rules in an eMCS are more expressive than in an mMCS, since they may be of two types: those that contain $next$ and those that do not. As already mentioned, the former are to be applied to the current knowledge base and not persist, whereas the latter are to be applied in the next time instant and persist. Therefore, we distinguish these two subsets.

Definition 3 *Let $M_e = \langle C_1, \ldots, C_n \rangle$ be an eMCS and S a belief state for M_e. Then, for each $1 \leq i \leq n$, consider the following sets:*

- *$app_i^{next}(S) = \{op(s) : next(op(s)) \in app_i(S)\}$*
- *$app_i^{now}(S) = \{op(s) : op(s) \in app_i(S)\}$*

Note that if we want an effect to be instantaneous and persistent, then this can also be achieved using two bridge rules with identical body, one with and one without $next$ in the head.

Similar to equilibria in mMCS, the (static) equilibrium is defined to incorporate instantaneous effects based on $app_i^{now}(S)$ alone.

[2] For simplicity of presentation, we consider discrete steps in time here.

Definition 4 *Let $M_e = \langle C_1, \dots, C_n \rangle$ be an eMCS. A belief state $S = \langle S_1, \dots, S_n \rangle$ for M_e is a static* equilibrium *of M_e iff, for each $1 \leq i \leq n$, we have $S_i \in \mathbf{ACC}_i(mng_i(app_i^{now}(S), kb_i))$.*

Note the minor change due to mng now only returning one kb.

To be able to assign meaning to an eMCS evolving over time, we introduce evolving belief states, which are sequences of belief states, each referring to a subsequent time instant.

Definition 5 *Let $M_e = \langle C_1, \dots, C_n \rangle$ be an eMCS. An* evolving belief state *of size s for M_e is a sequence $S_e = \langle S^1, \dots, S^s \rangle$ where each S^j, $1 \leq j \leq s$, is a belief state for M_e.*

To enable an eMCS to react to incoming observations and evolve, an observation sequence defined in the following has to be processed. The idea is that the knowledge bases of the observation contexts C_i^o change according to that sequence.

Definition 6 *Let $M_e = \langle C_1^o, \dots, C_\ell^o, C_{\ell+1}, \dots, C_n \rangle$ be an eMCS. An* observation sequence *for M_e is a sequence $Obs = \langle \mathcal{O}^1, \dots, \mathcal{O}^m \rangle$, such that, for each $1 \leq j \leq m$, $\mathcal{O}^j = \langle o_1^j, \dots, o_\ell^j \rangle$ is an* instant observation *with $o_i^j \in \mathbf{KB}_i$ for each $1 \leq i \leq \ell$.*

To be able to update the knowledge bases in the evolving contexts, we need one further notation. Given an evolving context C_i and $k \in \mathbf{KB}_i$, we denote by $C_i[k]$ the evolving context in which kb_i is replaced by k, i.e., $C_i[k] = \langle L_i, k, br_i, OP_i, mng_i \rangle$.

We can now define that certain evolving belief states are evolving equilibria of an eMCS $M_e = \langle C_1^o, \dots, C_\ell^o, C_{\ell+1}, \dots, C_n \rangle$ given an observation sequence $Obs = \langle \mathcal{O}^1, \dots, \mathcal{O}^m \rangle$ for M_e. The intuitive idea is that, given an evolving belief state $S_e = \langle S^1, \dots, S^s \rangle$ for M_e, in order to check if S_e is an evolving equilibrium, we need to consider a sequence of eMCSs, $M^1, \dots, M^s$ (each with ℓ observation contexts), representing a possible evolution of M_e according to the observations in Obs, such that S^j is a (static) equilibrium of M^j. The knowledge bases of the observation contexts in M^j are exactly their corresponding elements o_i^j in $\mathcal{O}^j$. For each of the other contexts C_i, $\ell + 1 \leq i \leq n$, its knowledge base in M^j is obtained from the one in M^{j-1} by applying the operations in $app_i^{next}(S^{j-1})$.

Definition 7 *Let $M_e = \langle C_1^o, \dots, C_\ell^o, C_{\ell+1}, \dots, C_n \rangle$ be an eMCS, $S_e = \langle S^1, \dots, S^s \rangle$ an evolving belief state of size s for M_e, and $Obs = \langle \mathcal{O}^1, \dots, \mathcal{O}^m \rangle$ an observation sequence for M_e such that $m \geq s$. Then, S_e is an* evolving equilibrium *of size s of M_e given Obs iff, for each $1 \leq j \leq s$, S^j is an equilibrium of $M^j = \langle C_1^o[o_1^j], \dots, C_\ell^o[o_\ell^j], C_{\ell+1}[k_{\ell+1}^j], \dots, C_n[k_n^j] \rangle$ where, for each $\ell + 1 \leq i \leq n$, k_i^j is defined inductively as follows:*

- $k_i^1 = kb_i$
- $k_i^{j+1} = mng_i(app_i^{next}(S^j), k_i^j)$

Note that $next$ in bridge rule heads of observation contexts are thus without any effect, in other words, observation contexts can indeed be understood as managed contexts whose knowledge base changes with each time instant.

The essential difference to [19] is that the k_i^{j+1} can be effectively computed (instead of picking one of several options), simply because mng always returns one knowledge base. The same applies in Def. 4.

As shown in [19], two consequences of the previous definitions are that any subsequence of an evolving equilibrium is also an evolving equilibrium, and mMCSs are a particular case of eMCSs.

4 Use Case Scenario

In this section, we illustrate eMCSs adapting a scenario on cargo shipment assessment taken from [32].

The customs service for any developed country assesses imported cargo for a variety of risk factors including terrorism, narcotics, food and consumer safety, pest infestation, tariff violations, and intellectual property rights.[3] Assessing this risk, even at a preliminary level, involves extensive knowledge about commodities, business entities, trade patterns, government policies and trade agreements. Some of this knowledge may be external to a given customs agency: for instance the broad classification of commodities according to the international Harmonized Tariff System (HTS), or international trade agreements. Other knowledge may be internal to a customs agency, such as lists of suspected violators or of importers who have a history of good compliance with regulations. While some of this knowledge is relatively stable, much of it changes rapidly. Changes are made not only at a specific level, such as knowledge about the expected arrival date of a shipment; but at a more general level as well. For instance, while the broad HTS code for tomatoes (0702) does not change, the full classification and tariffs for cherry tomatoes for import into the US changes seasonally.

Here, we consider an eMCS $M_e = \langle C_1^o, C_2^o, C_3, C_4 \rangle$ composed of two observation contexts C_1^o and C_2^o, and two reasoning contexts C_3 and C_4. The first observation context is used to capture the data of passing shipments, i.e., the country of their origination, the commodity they contain, their importers and producers. Thus, the knowledge base and belief set language of C_1^o is composed of all the ground atoms over $\mathsf{ShpmtCommod}/2$, $\mathsf{ShpmtDeclHTSCode}/2$, $\mathsf{ShpmtImporter}/2$, $\mathsf{ShpmtCountry}/2$, $\mathsf{ShpmtProducer}/2$, and also $\mathsf{GrapeTomato}/1$ and $\mathsf{CherryTomato}/1$. The second observation context C_2^o serves to insert administrative information and data from other institutions. Its knowledge base and belief set language is composed of all the ground atoms over $\mathsf{NewEUMember}/1$, $\mathsf{Misfiling}/1$, and $\mathsf{RandomInspection}/1$. Neither of the two observation contexts has any bridge rules.

The reasoning context C_3 is an ontological Description Logic (DL) context that contains a geographic classification, along with information about producers who are located in various countries. It also contains a classification of commodities based on their harmonized tariff information (HTS chapters, headings and codes, cf. `http://www.usitc.gov/tata/hts`). We refer to [11] and [8] for the standard definition of L_3; kb_3 is given as follows:

$\mathsf{Commodity} \equiv (\exists \mathsf{HTSCode}.\top)$
$\mathsf{EdibleVegetable} \equiv (\exists \mathsf{HTSChapter}.\,\{\text{‘07’}\})$
$\mathsf{CherryTomato} \equiv (\exists \mathsf{HTSCode}.\,\{\text{‘07020020’}\})$
$\mathsf{Tomato} \equiv (\exists \mathsf{HTSHeading}.\,\{\text{‘0702’}\})$
$\mathsf{GrapeTomato} \equiv (\exists \mathsf{HTSCode}.\,\{\text{‘07020010’}\})$
$\mathsf{CherryTomato} \sqsubseteq \mathsf{Tomato}$ $\quad$ $\mathsf{CherryTomato} \sqcap \mathsf{GrapeTomato} \sqsubseteq \bot$
$\mathsf{GrapeTomato} \sqsubseteq \mathsf{Tomato}$ $\quad$ $\mathsf{Tomato} \sqsubseteq \mathsf{EdibleVegetable}$
$\mathsf{EURegisteredProducer} \equiv (\exists \mathsf{RegisteredProducer}.\mathsf{EUCountry})$
$\mathsf{LowRiskEUCommodity} \equiv (\exists \mathsf{ExpeditableImporter}.\top) \sqcap$
$\qquad (\exists \mathsf{CommodCountry}.\mathsf{EUCountry})$
$\mathsf{EUCountry}(portugal)$ $\quad$ $\mathsf{RegisteredProducer}(p_1, portugal)$
$\mathsf{EUCountry}(slovakia)$ $\quad$ $\mathsf{RegisteredProducer}(p_2, slovakia)$

OP_3 contains a single add operation to add factual knowledge. The bridge rules br_3 are given as follows:

[3] The system described here is not intended to reflect the policies of any country or agency.

$add(\mathsf{CherryTomato}(\mathbf{x})) \leftarrow (1 : \mathsf{CherryTomato}(\mathbf{x}))$
$add(\mathsf{GrapeTomato}(\mathbf{x})) \leftarrow (1 : \mathsf{GrapeTomato}(\mathbf{x}))$
$next(add(\mathsf{EUCountry}(\mathbf{x}))) \leftarrow (2 : \mathsf{NewEUMember}(\mathbf{x}))$
$add(\mathsf{CommodCountry}(\mathbf{x},\mathbf{y})) \leftarrow (1 : \mathsf{ShpmtCommod}(\mathbf{z},\mathbf{x})),$
$\quad (1 : \mathsf{ShpmtCountry}(\mathbf{z},\mathbf{y}))$
$add(\mathsf{ExpeditableImporter}(\mathbf{x},\mathbf{y})) \leftarrow (1 : \mathsf{ShpmtCommod}(\mathbf{z},\mathbf{x})),$
$\quad (1 : \mathsf{ShpmtImporter}(\mathbf{z},\mathbf{y})), (4 : \mathsf{AdmissibleImporter}(\mathbf{y})),$
$\quad (4 : \mathsf{ApprovedImporterOf}(\mathbf{y},\mathbf{x}))$

Note that kb_3 can indeed be expressed in the DL $\mathcal{EL}^{++}$ [4] for which standard reasoning tasks, such as subsumption, can be computed in PTIME.

Finally, C_4 is a logic programming (LP) indicating information about importers, and about whether to inspect a shipment either to check for compliance of tariff information or for food safety issues. For L_4 we consider that $\mathbf{KB}_i$ the set of normal logic programs over a signature Σ, $\mathbf{BS}_i$ is the set of atoms over Σ, and $\mathbf{ACC}_i(kb)$ returns returns a singleton set containing only the set of true atoms in the unique well-founded model. The latter is a bit unconventional, since this way undefinedness under the well-founded semantics [15] is merged with false information. However, as long as no loops over negation occur in the LP context (in combination with its bridge rules), undefinedness does not occur, and the obvious benefit of this choice is that computing the well-founded model is PTIME-data-complete [10]. We consider $OP_4 = OP_3$, and kb_4 and br_4 are given as follows:

$\mathsf{AdmissibleImporter}(\mathbf{x}) \leftarrow \sim\mathsf{SuspectedBadGuy}(\mathbf{x}).$
$\mathsf{PartialInspection}(\mathbf{x}) \leftarrow \mathsf{RandomInspection}(\mathbf{x}).$
$\mathsf{FullInspection}(\mathbf{x}) \leftarrow \sim\mathsf{CompliantShpmt}(\mathbf{x}).$
$\mathsf{SuspectedBadGuy}(i_1).$

$next((\mathsf{SuspectedBadGuy}(\mathbf{x})) \leftarrow (2 : \mathsf{Misfiling}(\mathbf{x}))$
$add(\mathsf{ApprovedImporterOf}(i_2,\mathbf{x})) \leftarrow (3 : \mathsf{EdibleVegetable}(\mathbf{x}))$
$add(\mathsf{ApprovedImporterOf}(i_3,\mathbf{x})) \leftarrow (1 : \mathsf{GrapeTomato}(\mathbf{x}))$
$add(\mathsf{CompliantShpmt}(\mathbf{x})) \leftarrow (1 : \mathsf{ShpmtCommod}(\mathbf{x},\mathbf{y})),$
$\quad (3 : \mathsf{HTSCode}(\mathbf{y},\mathbf{z})), (1 : \mathsf{ShpmtDeclHTSCode}(\mathbf{x},\mathbf{z}))$
$add(\mathsf{RandomInspection}(\mathbf{x})) \leftarrow (1 : \mathsf{ShpmtCommod}(\mathbf{x},\mathbf{y})),$
$\quad (2 : \mathsf{Random}(\mathbf{y}))$
$add(\mathsf{PartialInspection}(\mathbf{x})) \leftarrow (1 : \mathsf{ShpmtCommod}(\mathbf{x},\mathbf{y})),$
$\quad \mathbf{not}\ (3 : \mathsf{LowRiskEUCommodity}(\mathbf{y}))$
$add(\mathsf{FullInspection}(\mathbf{x})) \leftarrow (1 : \mathsf{ShpmtCommod}(\mathbf{x},\mathbf{y})),$
$\quad (3 : \mathsf{Tomato}(\mathbf{y})), (1 : \mathsf{ShpmtCountry}(\mathbf{x}, \mathit{slovakia}))$

Now consider the observation sequence $Obs = \langle \mathcal{O}^1, \mathcal{O}^2, \mathcal{O}^3 \rangle$ where o_1^1 consists of the following atoms on s_1 (where s in s_1 stands for shipment, c for commodity, and i for importer):

$\mathsf{ShpmtCommod}(s_1, c_1)$	$\mathsf{ShpmtDeclHTSCode}(s_1, \text{‘07020010’})$
$\mathsf{ShpmtImporter}(s_1, i_1)$	$\mathsf{CherryTomato}(c_1)$

o_1^2 of the following atoms on s_2:

$\mathsf{ShpmtCommod}(s_2, c_2)$	$\mathsf{ShpmtDeclHTSCode}(s_2, \text{‘07020020’})$
$\mathsf{ShpmtImporter}(s_2, i_2)$	$\mathsf{ShpmtCountry}(s_2, \mathit{portugal})$
$\mathsf{CherryTomato}(c_2)$	

and o_1^3 of the following atoms on s_3:

$\mathsf{ShpmtCommod}(s_3, c_3)$	$\mathsf{ShpmtDeclHTSCode}(s_3, \text{‘07020010’})$
$\mathsf{ShpmtImporter}(s_3, i_3)$	$\mathsf{ShpmtCountry}(s_3, \mathit{portugal})$
$\mathsf{GrapeTomato}(c_3)$	$\mathsf{ShpmtProducer}(s_3, p_1)$

while $o_2^1 = o_2^3 = \emptyset$ and $o_2^2 = \{\mathsf{Misfiling}(i_3)\}$. Then, an evolving equilibrium of size 3 of M_e given Obs is the sequence $S_e = \langle S^1, S^2, S^3 \rangle$ such that, for each $1 \leq j \leq 3$, $S^j = \langle S_1^j, S_2^j, S_3^j, S_4^j \rangle$. Since it is not feasible to present the entire S_e, we just highlight some interesting parts related to the evolution of the system. E.g., we have that $\mathsf{FullInspection}(s_1) \in S_4^1$ since the HTS code does not correspond to the cargo; no inspection on s_2 in S_4^2 since the shipment is compliant, c_2 is a EU commodity, and s_2 was not picked for random inspection; and $\mathsf{PartialInspection}(s_3) \in S_4^3$, even though s_3 comes from a EU country, because i_3 has been identified at time instant 2 for misfiling, which has become permanent info available at time 3.

5 Grounded Equilibria and Well-founded Semantics

Even if we only consider MCSs M, which are static and where an implicit mng always returns precisely one knowledge base, such that reasoning in all contexts can be done in PTIME, then deciding whether M has an equilibrium is in NP [7, 8]. The same result necessarily also holds for eMCSs, which can also be obtained from the considerations on eMCSs [19].

A number of special notions were studied in the context of MCSs that tackle this problem [7]. In fact, the notion of minimal equilibria was introduced with the aim of avoiding potential self-justifications. Then, grounded equilibria as a special case for so-called reducible MCSs were presented for which the existence of minimal equilibria can be effectively checked. Subsequently, a well-founded semantics for such reducible MCSs was defined under which an approximation of all grounded equilibria can be computed more efficiently. In the following, we transfer these notions from static MCSs in [7] to dynamic eMCSs and discuss under which (non-trivial) conditions they can actually be applied.

Given an eMCS $M_e = \langle C_1, \ldots, C_n \rangle$, we say that a static equilibrium $S = \langle S_1, \ldots, S_n \rangle$ is *minimal* if there is no equilibrium $S' = \langle S_1', \ldots, S_n' \rangle$ such that $S_i' \subseteq S_i$ for all i with $1 \leq i \leq n$ and $S_j' \subsetneq S_j$ for some j with $1 \leq j \leq n$.

This notion of minimality ensures the avoidance of self-justifications in evolving equilibria. The problem with this notion in terms of computation is that such minimization in general adds an additional level in the polynomial hierarchy. Therefore, we now formalize conditions under which minimal equilibria can be effectively checked. The idea is that the grounded equilibrium will be assigned to an eMCS M_e if all the logics of all its contexts can be reduced to special monotonic ones using a so-called reduction function. In the case where the logics of all contexts in M_e turn out to be monotonic, the minimal equilibrium will be unique.

Formally, a logic $L = (\mathbf{KB}, \mathbf{BS}, \mathbf{ACC})$ is *monotonic* if

1. $\mathbf{ACC}(kb)$ is a singleton set for each $kb \in \mathbf{KB}$, and
2. $S \subseteq S'$ whenever $kb \subseteq kb'$, $\mathbf{ACC}(kb) = \{ S \}$, and $\mathbf{ACC}(kb') = \{ S' \}$.

Furthermore, $L = (\mathbf{KB}, \mathbf{BS}, \mathbf{ACC})$ is *reducible* if for some $\mathbf{KB}^* \subseteq \mathbf{KB}$ and some reduction function $red : \mathbf{KB} \times \mathbf{BS} \to \mathbf{KB}^*$,

1. the restriction of L to $\mathbf{KB}^*$ is monotonic,
2. for each $kb \in \mathbf{KB}$, and all $S, S' \in \mathbf{BS}$:
   - $red(kb, S) = kb$ whenever $kb \in \mathbf{KB}^*$,
   - $red(kb, S) \subseteq red(kb, S')$ whenever $S' \subseteq S$,
   - $S \in \mathbf{ACC}(kb)$ iff $\mathbf{ACC}(red(kb, S)) = \{ S \}$.

Then, an evolving context $C = (L, kb, br, OP, mng)$ is *reducible* if its logic L is reducible and, for all $op \in F_L^{OP}$ and all belief sets S, $red(mng(op, kb), S) = mng(op, red(kb, S))$.

An eMCS is *reducible* if all of its contexts are. Note that a context is reducible whenever its logic L is monotonic. In this case $\mathbf{KB}^*$ coincides with $\mathbf{KB}$ and red is the identity with respect to the first argument.

As pointed out in [7], reducibility is inspired by the reduct in (non-monotonic) answer set programming. The crucial and novel condition in our case is the one that essentially says that the reduction function red and the management function mng can be applied in an arbitrary order. This may restrict to some extent the sets of operations OP and mng, but in our use case scenario in Sect. 4, all contexts are indeed reducible.

A particular case of reducible eMCSs, definite eMCSs, does not require the reduction function and admits the polynomial computation of minimal evolving equilibria as we will see next. Namely, a reducible eMCS $M_e = \langle C_1, \dots, C_n \rangle$ is *definite* if

1. none of the bridge rules in any context contains $\mathbf{not}$,
2. for all i and all $S \in \mathbf{BS}_i$, $kb_i = red_i(kb_i, S)$.

In a definite eMCS, bridge rules are monotonic, and knowledge bases are already in reduced form. Inference is thus monotonic and a unique minimal equilibrium exists. We take this equilibrium to be the grounded equilibrium. Let M_e be a definite eMCS. A belief state S of M_e is the *grounded equilibrium of* M_e, denoted by $\mathbf{GE}(M_e)$, if S is the unique minimal (static) equilibrium of M_e. This notion gives rise to evolving grounded equilibria.

Definition 8 *Let $M_e = \langle C_1, \dots, C_n \rangle$ be a definite eMCS, $S_e = \langle S^1, \dots, S^s \rangle$ an evolving belief state of size s for M_e, and $Obs = \langle \mathcal{O}^1, \dots, \mathcal{O}^m \rangle$ an observation sequence for M_e such that $m \geq s$. Then, S_e is the* evolving grounded equilibrium *of size s of M_e given Obs iff, for each $1 \leq j \leq s$, S^j is a grounded equilibrium of M^j defined as in Definition 7.*

Grounded equilibria for definite eMCSs can indeed be efficiently computed following [7]. The only additional requirement is that all operations $op \in OP$ are *monotonic*, i.e., for kb, we have that $kb \subseteq mng(op(s), kb)$. Note that this is indeed a further restriction and not covered by reducible eMCSs. Now, for $1 \leq i \leq n$, let $kb_i^0 = kb_i$ and define, for each successor ordinal $\alpha + 1$,

$$kb_i^{\alpha+1} = mng(app_i^{now}(E^\alpha), kb_i^\alpha),$$

where $E^\alpha = (E_1^\alpha, \dots, E_n^\alpha)$ and $\mathbf{ACC}_i(kb_i^\alpha) = \{E_i^\alpha\}$. Furthermore, for each limit ordinal α, define $kb_i^\alpha = \bigcup_{\beta \leq \alpha} kb_i^\beta$, and let $kb_i^\infty = \bigcup_{\alpha > 0} kb_i^\alpha$. Then Proposition 1 [7] can be adapted:

Proposition 1 *Let $M_e = \langle C_1, \dots, C_n \rangle$ be a definite eMCS s.t. all OP_i are monotonic. A belief state $S = \langle S_1, \dots, S_n \rangle$ is the grounded equilibrium of M_e iff $\mathbf{ACC}_i(kb_i^\infty) = \{S_i\}$, for $1 \leq i \leq n$.*

As pointed out in [7], for many logics, $kb_i^\infty = kb_i^\omega$ holds, i.e., the iteration stops after finitely many steps. This is indeed the case for the use case scenario in Sect. 4.

For evolving belief states S_e of size s and an observation sequence Obs for M_e, this proposition yields that the evolving grounded equilibrium for definite eMCSs can be obtained by simply applying this iteration s times.

Grounded equilibria for general eMCSs are defined based on a reduct which generalizes the Gelfond-Lifschitz reduct to the multi-context case:

Definition 9 *Let $M_e = \langle C_1, \dots, C_n \rangle$ be a reducible eMCS and $S = \langle S_1, \dots, S_n \rangle$ a belief state of M_e. The S-*reduct of *M_e is defined as $M_e^S = \langle C_1^S, \dots, C_n^S \rangle$ where, for each $C_i = \langle L_i, kb_i, br_i, OP_i, mng_i \rangle$, we define $C_i^S = (L_i, red_i(kb_i, S_i), br_i^S, OP_i, mng_i)$. Here, br_i^S results from br_i by deleting all*

1. *rules with $\mathbf{not}\ (r : p)$ in the body such that $S \models (r : p)$, and*
2. *$\mathbf{not}$ literals from the bodies of remaining rules.*

For each reducible eMCS M_e and each belief set S, the S-reduct of M_e is definite. We can thus check whether S is a grounded equilibrium in the usual manner:

Definition 10 *Let M_e be a reducible eMCS such that all OP_i are monotonic. A belief state S of M_e is a* grounded equilibrium of M_e *if S is the grounded equilibrium of M_e^S, that is $S = \mathbf{GE}(M_e^S)$.*

The following result generalizes Proposition 2 from [7].

Proposition 2 *Every grounded equilibrium of a reducible eMCS M_e such that all OP_i are monotonic is a minimal equilibrium of M_e.*

This can again be generalized to evolving grounded equilibria.

Definition 11 *Let $M_e = \langle C_1, \dots, C_n \rangle$ be a normal, reducible eMCS such that all OP_i are monotonic, $S_e = \langle S^1, \dots, S^s \rangle$ an evolving belief state of size s for M_e, and $Obs = \langle \mathcal{O}^1, \dots, \mathcal{O}^m \rangle$ an observation sequence for M_e such that $m \geq s$. Then, S_e is the* evolving grounded equilibrium *of size s of M_e given Obs iff, for each $1 \leq j \leq s$, S^j is the grounded equilibrium of $(M^j)^{S^j}$ with M^j defined as in Definition 7.*

This computation is still not polynomial, since, intuitively, we have to guess and check the (evolving) equilibrium, which is why the well-founded semantics for reducible eMCSs M_e is introduced following [7]. Its definition is based on the operator $\gamma_{M_e}(S) = \mathbf{GE}(M_e^S)$, provided $\mathbf{BS}_i$ for each logic L_i in all the contexts of M_e has a least element S^*. Such eMCSs are called *normal*.

The following result can be straightforwardly adopted from [7].

Proposition 3 *Let $M_e = \langle C_1, \dots, C_n \rangle$ be a reducible eMCS such that all OP_i are monotonic. Then γ_{M_e} is antimonotone.*

As usual, applying γ_{M_e} twice yields a monotonic operator. Hence, by the Knaster-Tarski theorem, $(\gamma_{M_e})^2$ has a least fixpoint which determines the well-founded semantics.

Definition 12 *Let $M_e = \langle C_1, \dots, C_n \rangle$ be a normal, reducible eMCS such that all OP_i are monotonic. The well-founded semantics of M_e, denoted $\mathbf{WFS}(M)$, is the least fixpoint of $(\gamma_{M_e})^2$.*

Starting with the least belief state $S^* = \langle S_1^*, \dots, S_n^* \rangle$, this fixpoint can be iterated, and the following correspondence between $\mathbf{WFS}(M_e)$ and the grounded equilibria of M_e can be shown.

Proposition 4 *Let $M_e = \langle C_1, \dots, C_n \rangle$ be a normal, reducible eMCS such that all OP_i are monotonic, $\mathbf{WFS}(M_e) = \langle W_1, \dots W_n \rangle$, and $S = \langle S_1, \dots, S_n \rangle$ a grounded equilibrium of M_e. Then $W_i \subseteq S_i$ for $1 \leq i \leq n$.*

The well-founded semantics can thus be viewed as an approximation of the belief state representing what is accepted in all grounded equilibria, even though $\mathbf{WFS}(M_e)$ may itself not necessarily be an equilibrium. Yet, if all $\mathbf{ACC}_i$ deterministically return one element of $\mathbf{BS}_i$ and the eMCS is acyclic (i.e., no cyclic dependencies over bridge rules exist between beliefs in the eMCS see [19]), then the grounded equilibrium is unique and identical to the well-founded semantics. This is indeed the case for the use case in Sect. 4.

As before, the well-founded semantics can be generalized to evolving belief states.

Definition 13 *Let $M_e = \langle C_1, \ldots, C_n \rangle$ be a normal, reducible eMCS such that all OP_i are monotonic, and $Obs = \langle \mathcal{O}^1, \ldots, \mathcal{O}^m \rangle$ an observation sequence for M_e such that $m \geq s$. The* evolving well-founded semantics *of M_e, denoted $\mathbf{WFS}_e(M)$, is the evolving belief state $S_e = \langle S^1, \ldots, S^s \rangle$ of size s for M_e such that S^j is the well-founded semantics of M^j defined as in Definition 7.*

Finally, as intended, we can show that computing the evolving well-founded semantics of M_e can be done in polynomial time under the restrictions established so far. For analyzing the complexity in each time instant, we can utilize *output-projected* belief states [11]. The idea is to consider only those beliefs that appear in some bridge rule body. Formally, given an evolving context C_i within $M_e = \langle C_1, \ldots, C_n \rangle$, we can define OUT_i to be the set of all beliefs of C_i occurring in the body of some bridge rule in M_e. The *output-projection* of a belief state $S = \langle S_1, \ldots, S_n \rangle$ of M_e is the belief state $S' = \langle S'_1, \ldots, S'_n \rangle$, $S'_i = S_i \cap OUT_i$, for $1 \leq i \leq n$.

Following [11, 8], we can adapt the *context complexity* of C_i from [19] as the complexity of the following problem:

(CC) Decide, given $Op_i \subseteq OF_i$ and $S'_i \subseteq OUT_i$, if exist $kb'_i = mng_i(Op_i, kb_i)$ and $S_i \in \mathbf{ACC}_i(kb'_i)$ s.t. $S'_i = S_i \cap OUT_i$.

Problem (CC) can intuitively be divided into two subproblems: (MC) compute some $kb'_i = mng_i(Op_i, kb_i)$ and (EC) decide whether $S_i \in \mathbf{ACC}(kb'_i)$ exists s.t. $S'_i = S_i \cap OUT_i$. Here, (MC) is trivial for monotonic operations, so (EC) determines the complexity of (CC).

Theorem 1 *Let $M_e = \langle C_1, \ldots, C_n \rangle$ be a normal, reducible eMCS such that all OP_i are monotonic, $Obs = \langle \mathcal{O}^1, \ldots, \mathcal{O}^m \rangle$ an observation sequence for M_e, and (CC) is in PTIME for all C_i. Then, for $s \leq m$, computing $\mathbf{WFS}^s_e(M_e)$ is in PTIME.*

This, together with the observation that $\mathbf{WFS}_e(M_e)$ coincides with the unique grounded equilibrium, allows us to verify that computing the results in our use case scenario can be done in polynomial time.

6 Related and Future Work

In this paper we have studied how eMCSs can be revised in such a way that polynomial reasoning is possible, and we have discussed an example use case to which this result applies. We have also investigated the adaptation of notions concerning minimality of (evolving) equilibria, and we observe that the notion of reducible eMCSs is considerably restricted, but not to the same extent as the efficient computation of the well-founded semantics requires. An open question is whether a more refined computation eventually tailored to less restrictive operations than considered here can be used to achieve similar results.

As mentioned in the Introduction, eMCSs share the main ideas of reactive Multi-Context Systems sketched in [6, 12, 9] inasmuch as both aim at extending mMCSs to cope with dynamic observations. Three main differences distinguish them. First, whereas eMCSs rely on a sequence of observations, each independent from the previous ones, rMCSs encode such sequences within the same observation contexts, with its elements being explicitly timestamped. This means that with rMCSs it is perhaps easier to write bridge rules that refer, e.g., to specific sequences of observations, which in eMCSs would require explicit timestamps and storing the observations in some context, although at the cost that rMCSs need to deal with explicit time which adds an additional overhead. Second, since in rMCSs the contexts resulting from the application of the management operations are the ones that are used in the subsequent state, difficulties may arise in separating non-persistent and persistent effects, for example, allowing an observation to override some fact in some context while the observation holds, but without changing the context itself – such separation is easily encodable in eMCSs given the two kinds of bridge rules, i.e., with or without operator $next$. Finally, bridge rules with $next$ allow for the specification of transitions based on the current state, such as the one encoded by the rule $next(add(p)) \leftarrow not\, p$, which do not seem possible in rMCSs. Overall, these differences indicate that an interesting future direction would be to merge both approaches, exploring a combination of explicitly timestamped observations with the expressiveness provided by operator $next$.

Another framework that aims at modeling the dynamics of knowledge is that of evolving logic programs EVOLP [2] focusing on updates of generalized logic programs. It is possible to show that EVOLP can be seen as a particular case of eMCSs, using the operator $next$ to capture the operator $assert$ of EVOLP. We leave the details for an extended version. Closely related to EVOLP, hence to eMCS, are the two frameworks of reactive ASP, one implemented as a solver *clingo* [14] and one described in [6]. The system *oclingo* extends an ASP solver for handling external modules provided at runtime by a controller. The output of these external modules can be seen as the observations of EVOLP. Unlike the observations in EVOLP, which can be rules, external modules in *oclingo* are restricted to produce atoms so the evolving capabilities are very restricted. On the other hand, *clingo* permits committing to a specific answer-set at each state, a feature that is not part of EVOLP, nor of eMCS. Reactive ASP as described in [6] can be seen as a more straightforward generalization of EVOLP where operations other than $assert$ for self-updating a program are permitted. Given the above mentioned embedding of EVOLP in eMCS, and the fact that eMCSs permit several (evolution) operations in the head of bridge rules, it is also not difficult to show that Reactive ASP as described in [6] can be captured by eMCSs.

Also, as already outlined in [20], an important non-trivial topic is the study of the notion of minimal change within an evolving equilibrium. Whereas minimal change may be desirable to obtain more coherent evolving equilibria, there are also arguments against adopting a one-size-fits-all approach embedded in the semantics. Different contexts, i.e., KR formalisms, may require different notions of minimal change, or even require to avoid it – e.g., suppose we want to represent some variable that can non-deterministically takes one of two values at each time instant: minimal change could force a constant value.

Another important issue open for future work is a more fine-grained characterization of updating bridge rules (and knowledge bases) as studied in [18] in light of the encountered difficulties when updating rules [28, 29, 31] and the combination of updates over various formalisms [29, 30].

Also interesting is to study how to perform AGM style belief revision at the (semantic) level of the equilibria, as in Wang et al [33], though different since knowledge is not incorporated in the contexts.

ACKNOWLEDGEMENTS

We would like to thank the referees for their comments, which helped improve this paper considerably. Matthias Knorr and João Leite were partially supported by FCT under project "ERRO – Efficient Reasoning with Rules and Ontologies" (PTDC/EIA-CCO/121823/2010). Ricardo Gonçalves was supported by FCT grant SFRH/BPD/47245/2008 and Matthias Knorr was also partially supported by FCT grant SFRH/BPD/86970/2012.

REFERENCES

[1] M. Alberti, A. S. Gomes, R. Gonçalves, J. Leite, and M. Slota, 'Normative systems represented as hybrid knowledge bases', in *CLIMA*, eds., J. Leite, P. Torroni, T. Ågotnes, G. Boella, and L. van der Torre, volume 6814 of *LNCS*, pp. 330–346. Springer, (2011).

[2] J. Alferes, A. Brogi, J. Leite, and L. Pereira, 'Evolving logic programs', in *JELIA*, eds., S. Flesca, S. Greco, N. Leone, and G. Ianni, volume 2424 of *LNCS*, pp. 50–61. Springer, (2002).

[3] D. Anicic, S. Rudolph, P. Fodor, and N. Stojanovic, 'Stream reasoning and complex event processing in ETALIS', *Semantic Web*, **3**(4), 397–407, (2012).

[4] Franz Baader, Sebastian Brandt, and Carsten Lutz, 'Pushing the el envelope', in *IJCAI*, eds., Leslie Pack Kaelbling and Alessandro Saffiotti, pp. 364–369. Professional Book Center, (2005).

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

[6] G. Brewka, 'Towards reactive multi-context systems', in *LPNMR*, eds., P. Cabalar and T. C. Son, volume 8148 of *LNCS*, pp. 1–10. Springer, (2013).

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

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

[9] G. Brewka, S. Ellmauthaler, and J. Pührer, 'Multi-context systems for reactive reasoning in dynamic environments', in *ECAI*, eds., T. Schaub, G. Friedrich, and B. O'Sullivan. IOS Press, (2014). To appear.

[10] Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov, 'Complexity and expressive power of logic programming', *ACM Comput. Surv.*, **33**(3), 374–425, (2001).

[11] T. Eiter, M. Fink, P. Schüller, and A. Weinzierl, 'Finding explanations of inconsistency in multi-context systems', in *KR*, eds., F. Lin, U. Sattler, and M. Truszczynski. AAAI Press, (2010).

[12] S. Ellmauthaler, 'Generalizing multi-context systems for reactive stream reasoning applications', in *ICCSW*, eds., A. V. Jones and N. Ng, volume 35 of *OASICS*, pp. 19–26. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, (2013).

[13] M. Gebser, T. Grote, R. Kaminski, P. Obermeier, O. Sabuncu, and T. Schaub, 'Stream reasoning with answer set programming: Preliminary report', in *KR*, eds., G Brewka, T. Eiter, and S. A. McIlraith. AAAI Press, (2012).

[14] M. Gebser, T. Grote, R. Kaminski, and T. Schaub, 'Reactive answer set programming', in *LPNMR*, eds., J. P. Delgrande and W. Faber, volume 6645 of *LNCS*, pp. 54–66. Springer, (2011).

[15] Allen Van Gelder, Kenneth A. Ross, and John S. Schlipf, 'The well-founded semantics for general logic programs', *J. ACM*, **38**(3), 620–650, (1991).

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

[17] R. Gonçalves and J. Alferes, 'Parametrized logic programming', in *JELIA*, eds., T. Janhunen and I. Niemelä, volume 6341 of *LNCS*, pp. 182–194. Springer, (2010).

[18] R. Gonçalves, M. Knorr, and J. Leite, 'Evolving bridge rules in evolving multi-context systems', in *CLIMA XV*, eds., N. Bulling, L. van der Torre, S. Villata, W. Jamroga, and W. Vasconcelos, (2014). To appear.

[19] R. Gonçalves, M. Knorr, and J. Leite, 'Evolving multi-context systems', in *ECAI*, eds., T. Schaub, G. Friedrich, and B. O'Sullivan. IOS Press, (2014). To appear.

[20] R. Gonçalves, M. Knorr, and J. Leite, 'On minimal change in evolving multi-context systems (preliminary report)', in *ReactKnow 2014*, (2014). To appear.

[21] M. Homola, M. Knorr, J. Leite, and M. Slota, 'MKNF knowledge bases in multi-context systems', in *CLIMA*, eds., M. Fisher, L. van der Torre, M. Dastani, and G. Governatori, volume 7486 of *LNCS*, pp. 146–162. Springer, (2012).

[22] V. Ivanov, M. Knorr, and J. Leite, 'A query tool for $\mathcal{EL}$ with non-monotonic rules', in *ISWC*, eds., H. Alani, L. Kagal, A. Fokoue, P. T. Groth, C. Biemann, J. Parreira, L. Aroyo, N. F. Noy, C. Welty, and K. Janowicz, volume 8218 of *LNCS*, pp. 216–231. Springer, (2013).

[23] M. Knorr, J. Alferes, and P. Hitzler, 'Local closed world reasoning with description logics under the well-founded semantics', *Artif. Intell.*, **175**(9-10), 1528–1554, (2011).

[24] M. Knorr, M. Slota, J. Leite, and M. Homola, 'What if no hybrid reasoner is available? Hybrid MKNF in multi-context systems', *J. Log. Comput.*, (2013).

[25] F. Lécué and J. Pan, 'Predicting knowledge in an ontology stream', in *IJCAI*, ed., F. Rossi. IJCAI/AAAI, (2013).

[26] B. Motik and R. Rosati, 'Reconciling description logics and rules', *J. ACM*, **57**(5), (2010).

[27] F. Roelofsen and L. Serafini, 'Minimal and absent information in contexts', in *IJCAI*, eds., L. Kaelbling and A. Saffiotti, pp. 558–563. Professional Book Center, (2005).

[28] M. Slota and J. Leite, 'On semantic update operators for answer-set programs', in *ECAI*, eds., H. Coelho, R. Studer, and M. Wooldridge, volume 215 of *Frontiers in Artificial Intelligence and Applications*, pp. 957–962. IOS Press, (2010).

[29] M. Slota and J. Leite, 'Robust equivalence models for semantic updates of answer-set programs', in *KR*, eds., G. Brewka, T. Eiter, and S. A. McIlraith. AAAI Press, (2012).

[30] M. Slota and J. Leite, 'A unifying perspective on knowledge updates', in *JELIA*, eds., L. del Cerro, A. Herzig, and J. Mengin, volume 7519 of *LNCS*, pp. 372–384. Springer, (2012).

[31] M. Slota and J. Leite, 'The rise and fall of semantic rule updates based on SE-models', *TPLP*, (2014). To appear.

[32] Martin Slota, João Leite, and Terrance Swift, 'Splitting and updating hybrid knowledge bases', *TPLP*, **11**(4-5), 801–819, (2011).

[33] Y. Wang, Z. Zhuang, and K. Wang, 'Belief change in nonmonotonic multi-context systems', in *LPNMR*, eds., P. Cabalar and T. C. Son, volume 8148 of *LNCS*, pp. 543–555. Springer, (2013).

On Minimal Change in Evolving Multi-Context Systems (Preliminary Report)

Ricardo Gonçalves and **Matthias Knorr** and **João Leite** [1]

Abstract. Managed Multi-Context Systems (mMCSs) provide a general framework for integrating knowledge represented in heterogeneous KR formalisms. However, mMCSs are essentially static as they were not designed to run in a dynamic scenario. Some recent approaches, among them evolving Multi-Context Systems (eMCSs), extend mMCSs by allowing not only the ability to integrate knowledge represented in heterogeneous KR formalisms, but at the same time to both react to, and reason in the presence of commonly temporary dynamic observations, and evolve by incorporating new knowledge. The notion of minimal change is a central notion in dynamic scenarios, specially in those that admit several possible alternative evolutions. Since eMCSs combine heterogeneous KR formalisms, each of which may require different notions of minimal change, the study of minimal change in eMCSs is an interesting and highly non-trivial problem. In this paper, we study the notion of minimal change in eMCSs, and discuss some alternative minimal change criteria.

1 Introduction

Multi-Context Systems (MCSs) were introduced in [6], building on the work in [12, 23], to address the need for a general framework that integrates knowledge bases expressed in heterogeneous KR formalisms. Intuitively, instead of designing a unifying language (see e.g., [13, 22], and [19] with its reasoner NoHR [18]) to which other languages could be translated, in an MCS the different formalisms and knowledge bases are considered as modules, and means are provided to model the flow of information between them (cf. [1, 17, 20] and references therein for further motivation on hybrid languages and their connection to MCSs).

More specifically, an MCS consists of a set of contexts, each of which is a knowledge base in some KR formalism, such that each context can access information from the other contexts using so-called bridge rules. Such non-monotonic bridge rules add their heads to the context's knowledge base provided the queries (to other contexts) in their bodies are successful.

Managed Multi-Context Systems (mMCSs) were introduced in [7] to provide an extension of MCSs by allowing operations, other than simple addition, to be expressed in the heads of bridge rules. This allows mMCSs to properly deal with the problem of consistency management within contexts.

One recent challenge for KR languages is to shift from static application scenarios which assume a one-shot computation, usually triggered by a user query, to open and dynamic scenarios where there is a need to react and evolve in the presence of incoming information. Examples include EVOLP [2], Reactive ASP [11, 10], C-SPARQL [4], Ontology Streams [21] and ETALIS [3], to name only a few.

Whereas mMCSs are quite general and flexible to address the problem of integration of different KR formalisms, they are essentially static in the sense that the contexts do not evolve to incorporate the changes in the dynamic scenarios.

In such scenarios, new knowledge and information is dynamically produced, often from several different sources – for example a stream of raw data produced by some sensors, new ontological axioms written by some user, newly found exceptions to some general rule, etc.

To address this issue, two recent frameworks, evolving Multi-Context Systems (eMCSs) [15] and reactive Multi-Context Systems (rMCSs) [5, 9, 8] have been proposed sharing the broad motivation of designing general and flexible frameworks inheriting from mMCSs the ability to integrate and manage knowledge represented in heterogeneous KR formalisms, and at the same time be able to incorporate knowledge obtained from dynamic observations.

In such dynamic scenarios, where systems can have alternative evolutions, it is desirable to have some criteria of minimal change to be able to compare the possible alternatives. This problem is particularly interesting and non-trivial in dynamic frameworks based on MCSs, not only because of the heterogeneity of KR frameworks that may exist in an MCS – each of which may require different notions of minimal change –, but also because the evolution of such systems is based not only on the semantics, but also on the evolution of the knowledge base of each context.

In this paper, we study minimal change in eMCSs, by presenting some minimal change criteria to be applied to the possible evolving equilibria of an eMCS, and by discussing the relation between them.

The remainder of this paper is structured as follows. After presenting the main concepts regarding mMCSs, we introduce the framework of eMCSs. Then, we present and study some minimal change criteria in eMCSs. We conclude with discussing related work and possible future directions.

2 Multi-Context Systems

In this section, we introduce the framework of multi-context systems (MCSs). Following [6], a multi-context system (MCS) consists of a collection of components, each of which contains knowledge represented in some *logic*, defined as a triple $L = \langle \mathbf{KB}, \mathbf{BS}, \mathbf{ACC} \rangle$ where $\mathbf{KB}$ is the set of well-formed knowledge bases of L, $\mathbf{BS}$ is the set of possible belief sets, and $\mathbf{ACC} : \mathbf{KB} \to 2^{\mathbf{BS}}$ is a function describing the semantics of L by assigning to each knowledge base a set of acceptable belief sets. We assume that each element of $\mathbf{KB}$ and $\mathbf{BS}$ is a set, and we define $F = \{s : s \in kb \wedge kb \in \mathbf{KB}\}$.

In addition to the knowledge base in each component, *bridge rules*

[1] CENTRIA & Departamento de Informática, Faculdade Ciências e Tecnologia, Universidade Nova de Lisboa, email: rjrg@fct.unl.pt

are used to interconnect the components, specifying what knowledge to assert in one component given certain beliefs held in the components of the MCS. Formally, for a sequence of logics $L = \langle L_1, \ldots, L_n \rangle$, an *$L_i$-bridge rule σ over L*, $1 \leq i \leq n$, is of the form

$$H(\sigma) \leftarrow B(\sigma) \tag{1}$$

where $B(\sigma)$ is a set of *bridge literals* of the form $(r : b)$ and of the form $\mathbf{not}\,(r\!:\!b)$, $1 \leq r \leq n$, with b a belief formula of L_r, and, for each $kb \in \mathbf{KB}_i$, $kb \cup \{H(\sigma)\} \in \mathbf{KB}_i$.

Bridge rules in MCSs only allow adding information to the knowledge base of their corresponding context. Note that the condition $kb \cup \{H(\sigma)\} \in \mathbf{KB}_i$ precisely guarantees that the resulting set is still a knowledge base of the context. In [7], an extension, called managed Multi-Context Systems (mMCSs), is introduced in order to allow other types of operations to be performed on a knowledge base. For that purpose, each context of an mMCS is associated with a *management base*, which is a set of operations that can be applied to the possible knowledge bases of that context. Given a management base OP and a logic L, let $OF = \{op(s) : op \in OP \wedge s \in F\}$ be the *set of operational formulas* that can be built from OP and L. Each context of an mMCS gives semantics to operations in its management base using a *management function* over a logic L and a management base OP, $mng : 2^{OF} \times \mathbf{KB} \rightarrow (2^{\mathbf{KB}} \setminus \{\emptyset\})$, i.e., $mng(op, kb)$ is the (non-empty) set of possible knowledge bases that result from applying the operations in *op* to the knowledge base kb. We assume that $mng(\emptyset, kb) = \{kb\}$. Now, L_i-bridge rules for mMCSs are defined in the same way as for MCSs, except that $H(\sigma)$ is now an operational formula over OP_i and L_i.

Definition 1 *A* managed Multi-Context System *(mMCS) is a sequence $M = \langle C_1, \ldots, C_n \rangle$, where each C_i, $i \in \{1, \ldots, n\}$, called a* managed context*, is defined as $C_i = \langle L_i, kb_i, br_i, OP_i, mng_i \rangle$ where $L_i = \langle \mathbf{KB}_i, \mathbf{BS}_i, \mathbf{ACC}_i \rangle$ is a logic, $kb_i \in \mathbf{KB}_i$, br_i is a set of L_i-bridge rules, OP_i is a management base, mng_i is a management function over L_i and OP_i.*

Note that, for the sake of readability, we consider a slightly restricted version of mMCSs where each $\mathbf{ACC}_i$ is still a function and not a set of functions as for logic suites [7].

For an mMCS $M = \langle C_1, \ldots, C_n \rangle$, a *belief state of M* is a sequence $S = \langle S_1, \ldots, S_n \rangle$ such that each S_i is an element of $\mathbf{BS}_i$. For a bridge literal $(r : b)$, $S \models (r : b)$ if $b \in S_r$ and $S \models \mathbf{not}\,(r : b)$ if $b \notin S_r$; for a set of bridge literals B, $S \models B$ if $S \models L$ for every $L \in B$. We say that a bridge rule σ of a context C_i is *applicable given a belief state S of M* if S satisfies $B(\sigma)$. We can then define $app_i(S)$, the set of heads of bridge rules of C_i which are applicable in S, by setting $app_i(S) = \{H(\sigma) : \sigma \in br_i \wedge S \models B(\sigma)\}$.

Equilibria are belief states that simultaneously assign an acceptable belief set to each context in the mMCS such that the applicable operational formulas in bridge rule heads are taken into account.

Definition 2 *A belief state $S = \langle S_1, \ldots, S_n \rangle$ of an mMCS M is an* equilibrium *of M if, for every $1 \leq i \leq n$,*

$$S_i \in \mathbf{ACC}_i(kb) \quad \textit{for some} \quad kb \in mng_i(app_i(S), kb_i).$$

3 Evolving Multi-Context Systems

In this section, we recall evolving Multi-Context Systems, which generalize mMCSs to a dynamic scenario in which contexts are enabled to react to external observations and evolve. For that purpose, we consider that some of the contexts in the MCS become so-called *observation contexts* whose knowledge bases will be constantly changing over time according to the observations made, similar, e.g., to streams of data from sensors.[2]

The changing observations then will also affect the other contexts by means of the bridge rules. As we will see, such effect can either be instantaneous and temporary, i.e., limited to the current time instant, similar to (static) mMCSs, where the body of a bridge rule is evaluated in a state that already includes the effects of the operation in its head, or persistent, but only affecting the next time instant. To achieve the latter, we extend the operational language with a unary meta-operation $next$ that can only be applied on top of operations.

Definition 3 *Given a management base OP and a logic L, we define eOF, the evolving operational language, as*

$$eOF = OF \cup \{next(op(s)) : op(s) \in OF\}.$$

The idea of observation contexts is that each such context has a language describing the set of possible observations of that context, along with its current observation. The elements of the language of the observation contexts can then be used in the body of bridge rules to allow contexts to access the observations. Formally, an *observation context* is a tuple $O = \langle \Pi_O, \pi \rangle$ where Π_O is the *observation language* of O and $\pi \subseteq \Pi_O$ is its *current observation*.

We can now define evolving Multi-Context Systems.

Definition 4 *An* evolving Multi-Context Systems *(eMCS) is a sequence $M_e = \langle C_1, \ldots, C_n, O_1, \ldots, O_\ell \rangle$, such that, for each $i \in \{1, \ldots, \ell\}$, $O_i = \langle \Pi_{O_i}, \pi_i \rangle$ is an* observation context*, and, for each $i \in \{1, \ldots, n\}$, C_i is an* evolving context *defined as $C_i = \langle L_i, kb_i, br_i, OP_i, mng_i \rangle$ where*

- *$L_i = \langle \mathbf{KB}_i, \mathbf{BS}_i, \mathbf{ACC}_i \rangle$ is a logic*
- *$kb_i \in \mathbf{KB}_i$*
- *br_i is a set of bridge rules of the form*

$$H(\sigma) \leftarrow a_1, \ldots, a_k, \mathbf{not}\ a_{k+1}, \ldots, \mathbf{not}\ a_n \tag{2}$$

  such that $H(\sigma) \in eOF_i$, and each a_i, $i \in \{1, \ldots, n\}$, is either of the form $(r\!:\!b)$ with $r \in \{1, \ldots, n\}$ and b a belief formula of L_r, or of the form $(r@b)$ with $r \in \{1, \ldots, \ell\}$ and $b \in \Pi_{O_r}$
- *OP_i is a management base*
- *mng_i is a management function over L_i and OP_i.*

Given an eMCS $M_e = \langle C_1, \ldots, C_n, O_1, \ldots, O_\ell \rangle$ we denote by $\mathbf{KB}_{M_e}$ the set of *knowledge base configurations for M_e*, i.e., $\mathbf{KB}_{M_e} = \{\langle k_1, \ldots, k_n \rangle : k_i \in \mathbf{KB}_i \text{ for each } 1 \leq i \leq n\}$.

Definition 5 *Let $M_e = \langle C_1, \ldots, C_n, O_1, \ldots, O_\ell \rangle$ be an eMCS. A* belief state *for M_e is a sequence $S = \langle S_1, \ldots, S_n \rangle$ such that, for each $1 \leq i \leq n$, we have $S_i \in \mathbf{BS}_i$. We denote by $\mathbf{BS}_{M_e}$ the set of belief states for M_e.*

An *instant observation* for M_e is a sequence $\mathcal{O} = \langle o_1, \ldots, o_\ell \rangle$ such that, for each $1 \leq i \leq \ell$, we have that $o_i \subseteq \Pi_{O_i}$.

The notion $app_i(S)$ of the set of heads of bridge rules of C_i which are applicable in a belief state S, cannot be directly transferred from mMCSs to eMCSs since bridge rule bodies can now contain atoms of the form $(r@b)$, whose satisfaction depends on the current observation. Formally, given a belief state $S = \langle S_1, \ldots, S_n \rangle$ for M_e and

[2] For simplicity of presentation, we consider discrete steps in time here.

an instant observation $\mathcal{O} = \langle o_1, \ldots, o_\ell \rangle$ for M_e, we define the satisfaction of bridge literals of the form $(r : b)$ as $S, \mathcal{O} \models (r : b)$ if $b \in S_r$ and $S, \mathcal{O} \models \mathbf{not}\,(r : b)$ if $b \notin S_r$. The satisfaction of bridge literal of the form $(r@b)$ depends on the current observations, i.e., we have that $S, \mathcal{O} \models (r@b)$ if $b \in o_r$ and $S \models \mathbf{not}\,(r@b)$ if $b \notin o_r$. As before, for a set B of bridge literals, we have that $S, \mathcal{O} \models B$ if $S, \mathcal{O} \models L$ for every $L \in B$.

We say that a bridge rule σ of a context C_i is *applicable given a belief state S and an instant observation $\mathcal{O}$* if its body is satisfied by S and $\mathcal{O}$, i.e., $S, \mathcal{O} \models B(\sigma)$.

We denote by $app_i(S, \mathcal{O})$ the set of heads of bridges rules of the context C_i which are applicable given the belief state S and the instant observation $\mathcal{O}$. In the case of eMCSs, the set $app_i(S, \mathcal{O})$ may contain two types of formulas: those that contain $next$ and those that do not. As already mentioned before, the former are to be applied to the current knowledge base and not persist, whereas the latter are to be applied in the next time instant and persist.

Definition 6 *Let $M_e = \langle C_1, \ldots, C_n, O_1, \ldots, O_\ell \rangle$ be an eMCS, S a belief state for M_e, and $\mathcal{O}$ an instant observation for M_e. Then, for each $1 \leq i \leq n$, consider the following sets:*

- $app_i^{next}(S, \mathcal{O}) = \{op(s) : next(op(s)) \in app_i(S, \mathcal{O})\}$
- $app_i^{now}(S, \mathcal{O}) = \{op(s) : op(s) \in app_i(S, \mathcal{O})\}$

Note that if we want an effect to be instantaneous and persistent, then this can also be achieved using two bridge rules with identical body, one with and one without $next$ in the head.

Similar to equilibria in mMCS, the (static) equilibrium is defined to incorporate instantaneous effects based on $app_i^{now}(S, \mathcal{O})$ alone.

Definition 7 *Let $M_e = \langle C_1, \ldots, C_n, O_1, \ldots, O_\ell \rangle$ be an eMCS, and $\mathcal{O}$ an instant observation for M_e. A belief state $S = \langle S_1, \ldots, S_n \rangle$ for M_e is an* equilibrium of M_e given $\mathcal{O}$ *iff for each $1 \leq i \leq n$, there exists some $kb \in mng_i(app_i^{now}(S, \mathcal{O}), kb_i)$ such that $S_i \in \mathbf{ACC}_i(kb)$.*

To be able to assign meaning to an eMCS evolving over time, we introduce evolving belief states, which are sequences of belief states, each referring to a subsequent time instant.

Definition 8 *Let $M_e = \langle C_1, \ldots, C_n, O_1, \ldots, O_\ell \rangle$ be an eMCS. An* evolving belief state *of size s for M_e is a sequence $S_e = \langle S^1, \ldots, S^s \rangle$ where each S^j, $1 \leq j \leq s$, is a belief state for M_e.*

To enable an eMCS to react to incoming observations and evolve, a sequence of observations defined in the following has to be processed. The idea is that the knowledge bases of the observation contexts O_i change according to that sequence.

Definition 9 *Let $M_e = \langle C_1, \ldots, C_n, O_1, \ldots, O_\ell \rangle$ be an eMCS. A* sequence of observations *for M_e is a sequence $Obs = \langle \mathcal{O}^1, \ldots, \mathcal{O}^m \rangle$, such that, for each $1 \leq j \leq m$, $\mathcal{O}^j = \langle o_1^j, \ldots, o_\ell^j \rangle$ is an instant observation for M_e, i.e., $o_i^j \subseteq \Pi_{O_i}$ for each $1 \leq i \leq \ell$.*

To be able to update the knowledge bases and the sets of bridge rules of the evolving contexts, we need the following notation. Given an evolving context C_i, and a knowledge base $k \in \mathbf{KB}_i$, we denote by $C_i[k]$ the evolving context in which kb_i is replaced by k, i.e., $C_i[k] = \langle L_i, k, br_i, OP_i, mng_i \rangle$. For an observation context O_i, given a set $\pi \subseteq \Pi_{O_i}$ of observations for O_i, we denote by $O_i[\pi]$ the observation context in which its current observation is replaced by π, i.e., $O_i[\pi] = \langle \Pi_{O_i}, \pi \rangle$. Given $K = \langle k_1, \ldots, k_n \rangle \in \mathbf{KB}_{M_e}$ a knowledge base configuration for M_e, we denote by $M_e[K]$ the eMCS $\langle C_1[k_1], \ldots, C_n[k_n], O_1, \ldots, O_\ell \rangle$.

We can now define that certain evolving belief states are evolving equilibria of an $M_e = \langle C_1, \ldots, C_n, O_1, \ldots, O_\ell \rangle$ given a sequence of observations $Obs = \langle \mathcal{O}^1, \ldots, \mathcal{O}^m \rangle$ for M_e. The intuitive idea is that, given an evolving belief state $S_e = \langle S^1, \ldots, S^s \rangle$ for M_e, in order to check if S_e is an evolving equilibrium, we need to consider a sequence of eMCSs, $M^1, \ldots, M^s$ (each with ℓ observation contexts), representing a possible evolution of M_e according to the observations in Obs, such that each S^j is a (static) equilibrium of M^j. The current observation of each observation context O_i in M^j is exactly its corresponding element o_i^j in $\mathcal{O}^j$. For each evolving context C_i, its knowledge base in M^j is obtained from the one in M^{j-1} by applying the operations in $app_i^{next}(S^{j-1}, \mathcal{O}^{j-1})$.

Definition 10 *Let $M_e = \langle C_1, \ldots, C_n, O_1, \ldots, O_\ell \rangle$ be an eMCS, $S_e = \langle S^1, \ldots, S^s \rangle$ an evolving belief state of size s for M_e, and $Obs = \langle \mathcal{O}^1, \ldots, \mathcal{O}^m \rangle$ an observation sequence for M_e such that $m \geq s$. Then, S_e is an* evolving equilibrium *of size s of M_e given Obs iff, for each $1 \leq j \leq s$, S^j is an equilibrium of $M^j = \langle C_1[k_1^j], \ldots, C_n[k_n^j], O_1[o_1^j], \ldots, O_\ell[o_\ell^j] \rangle$ where, for each $1 \leq i \leq n$, k_i^j is defined inductively as follows:*

- $k_i^1 = kb_i$
- $k_i^{j+1} \in mng_i(app^{next}(S^j, \mathcal{O}_i^j), k_i^j)$.

We end this section by presenting a proposition about evolving equilibria that will be useful in the next section. In Def. 10, the number of considered time instances of observations m is greater or equal the size of the evolving belief state with the intuition that an equilibrium may also be defined for only a part of the observation sequence. An immediate consequence is that any subsequence of an evolving equilibrium is an evolving equilibrium.

Proposition 1 *Let $M_e = \langle C_1, \ldots, C_n, O_1, \ldots, O_\ell \rangle$ be an eMCS and $Obs = \langle \mathcal{O}^1, \ldots, \mathcal{O}^m \rangle$ an observation sequence for M_e. If $S_e = \langle S^1, \ldots, S^s \rangle$ is an evolving equilibrium of size s of M_e given Obs, then, for each $1 \leq j \leq s$, and every $j \leq k \leq m$, we have that $\langle S^1, \ldots, S^j \rangle$ is an evolving equilibrium of size j of M_e given the observation sequence $\langle \mathcal{O}^1, \ldots, \mathcal{O}^k \rangle$.*

4 Minimal change

In this section, we discuss some alternatives for the notion of minimal change in eMCSs. What makes this problem interesting is that in an eMCS there are different parameters that we may want to minimize in a transition from one time instant to the next one. In the following discussion we focus on two we deem most relevant: the operations that can be applied to the knowledge bases, and the distance between consecutive belief states.

We start by studying minimal change at the level of the operations. In the following discussion we consider fixed an eMCS $M_e = \langle C_1, \ldots, C_n, O_1, \ldots, O_\ell \rangle$.

Recall from the definition of evolving equilibrium that, in the transition between consecutive time instants, the knowledge base of each context C_i of M_e changes according to the operations in $app_i^{next}(S, \mathcal{O})$, and these depend on the belief state S and the instant observation $\mathcal{O}$. The first idea to compare elements of this set of operations is to, for a fixed instant observation $\mathcal{O}$, distinguish those equilibria of M_e which generate a minimal set of operations to be applied to the current knowledge bases to obtain the knowledge bases of the next time instant. Formally, given a knowledge base configuration $K \in \mathbf{KB}_{M_e}$ and an instant observation $\mathcal{O}$ for M_e, we can define the set:

$$\begin{aligned} MinEq(K,\mathcal{O}) = \{S : & S \text{ is an equilibrium of } M_e[K] \text{ given } \mathcal{O} \\ & \text{and there is no equilibrium } S' \text{ of } M_e[K] \text{ given } \mathcal{O} \\ & \text{such that, for all } i \in \{1,\ldots,n\} \text{ we have} \\ & app_i^{next}(S',\mathcal{O}) \subset app_i^{next}(S,\mathcal{O})\} \end{aligned}$$

This first idea of comparing equilibria based on inclusion of the sets of operations can, however, be too strict in most cases. Moreover, different operations usually have different costs, and it may well be that, instead of minimizing based on set inclusion, we want to minimize the total cost of the operations to be applied. For that, we need to assume that each context has a cost function over the set of operations, i.e., $cost_i : OP_i \to \mathbb{N}$, where $cost_i(op)$ represents the cost of performing operation *op*.

Let S be a belief state for M_e and $\mathcal{O}$ an instant observation for M_e. Then, for each $1 \leq i \leq n$, we define the cost of the operations to be applied to obtain the knowledge base of the next time instant as:

$$Cost_i(S,\mathcal{O}) = \sum_{op(s)\in app_i^{next}(S,\mathcal{O})} cost_i(op)$$

Summing up the cost for all evolving contexts, we obtain the global cost of S given $\mathcal{O}$:

$$Cost(S,\mathcal{O}) = \sum_{i=1}^{n} Cost_i(S,\mathcal{O})$$

Now that we have defined a cost function over belief states, we can define a minimization function over possible equilibria of eMCS $M_e[K]$ for a fixed knowledge base configuration $K \in \mathbf{KB}_{M_e}$. Formally, given $\mathcal{O}$ an instant observation for M_e, we define the set of equilibria of $M_e[K]$ given $\mathcal{O}$ which minimize the global cost of the operations to be applied to obtain the knowledge base configuration of the next time instant as:

$$\begin{aligned} MinCost(K,\mathcal{O}) = \{S : & S \text{ is an equilibrium of } M_e[K] \text{ given } \mathcal{O} \text{ and} \\ & \text{there is no equilibrium } S' \text{ of } M_e[K] \text{ given } \mathcal{O} \\ & \text{such that } Cost(S',\mathcal{O}) < Cost(S,\mathcal{O})\} \end{aligned}$$

Note that, instead of using a global cost, we could have also considered a more fine-grained criterion by comparing costs for each context individually, and define some order based on these comparisons. Also note that the particular case of taking $cost_i(op) = 1$ for every $i \in \{1,\ldots,n\}$ and every $op \in OP_i$, captures the scenario of minimizing the total number of operations to be applied.

The function $MinCost$ allows for the choice of those equilibria which are minimal with respect to the operations to be performed to the current knowledge base configuration in order to obtain the knowledge base configuration of the next time instant. Still, for each choice of an equilibrium S, we have to deal with the existence of several alternatives in the set $mng_i(app_i^{next}(S,\mathcal{O}), kb_i)$. Our aim now is to discuss how we can apply some notion of minimal change that allows us to compare the elements $mng_i(app_i^{next}(S,\mathcal{O}), kb_i)$. The intuitive idea is to compare the distance between the current equilibria and the possible equilibria resulting from the elements in $mng_i(app_i^{next}(S,\mathcal{O}), kb_i)$. Of course, given the possible heterogeneity of contexts in an eMCS, we cannot assume a global notion of distance between belief sets. Therefore, we assume that each evolving context has its own distance function between its beliefs sets. Formally, for each $1 \leq i \leq n$, we assume the existence of a distance function d_i, i.e., $d_i : \mathbf{BS}_i \times \mathbf{BS}_i \to \mathbb{R}$ satisfying for all $S_1, S_2, S_3 \in \mathbf{BS}_i$ the conditions:

1. $d_i(S_1, S_2) \geq 0$
2. $d_i(S_1, S_2) = 0$ iff $S_1 = S_2$
3. $d_i(S_1, S_2) = d_i(S_2, S_1)$
4. $d_i(S_1, S_3) \leq d_i(S_1, S_2) + d_i(S_2, S_3)$

There are some alternatives to extend the distance function of each context to a distance function between belief states. In the following we present two natural choices. One option is to consider the maximal distance between belief sets of each context. Another possibility is to consider the average of distances between belief sets of each context. Formally, given S^1 and S^2 two belief states of M_e we define two functions $\overline{d}_{\text{max}} : \mathbf{BS}_{M_e} \times \mathbf{BS}_{M_e} \to \mathbb{R}$ and $\overline{d}_{\text{avg}} : \mathbf{BS}_{M_e} \times \mathbf{BS}_{M_e} \to \mathbb{R}$ as follows:

$$\overline{d}_{\text{max}}(S^1, S^2) = Max\{d_i(S_i^1, S_i^2) \mid 1 \leq i \leq n\}$$

$$\overline{d}_{\text{avg}}(S^1, S^2) = \frac{\sum_{i=1}^{n} d_i(S_i^1, S_i^2)}{n}$$

We can easily prove that both $\overline{d}_{\text{max}}$ and $\overline{d}_{\text{avg}}$ are indeed distance functions between belief states.

Proposition 2 *The functions $\overline{d}_{max}$ and $\overline{d}_{avg}$ defined above are both distance functions, i.e., satisfy the axioms (1- 4).*

We now study how we can use one of these distance functions between belief states to compare the possible alternatives in the sets $mng_i(app_i^{next}(S,\mathcal{O}), kb_i)$, for each $1 \leq i \leq n$. Recall that the intuitive idea is to minimize the distance between the current belief state S and the possible equilibria that each element of $mng_i(app_i^{next}(S,\mathcal{O}), kb_i)$ can give rise to. We explore here two options, which differ on whether the minimization is global or local. The idea of global minimization is to choose only those knowledge base configurations $\langle k_1,\ldots,k_n\rangle \in \mathbf{KB}_{M_e}$ with $k_i \in mng_i(app_i^{next}(S,\mathcal{O}), kb_i)$, which guarantee minimal distance between the original belief state S and the possible equilibria of the obtained eMCS. The idea of local minimization is to consider all possible tuples $\langle k_1,\ldots,k_n\rangle$ with $k_i \in mng_i(app_i^{next}(S,\mathcal{O}), kb_i)$, and only apply minimization for each such choice, i.e., for each such knowledge base configuration we only allow equilibria with minimal distance from the original belief state.

We first consider the case of pruning those tuples $\langle k_1,\ldots,k_n\rangle$ with $k_i \in mng_i(app_i^{next}(S,\mathcal{O}), kb_i)$ which do not guarantee minimal change with respect to the original belief state. We start with an auxiliary function. Let S be a belief state for M_e, $K = \langle k_1,\ldots,k_n\rangle \in \mathbf{KB}_{M_e}$ a knowledge base configuration for M_e, and $\mathcal{O} = \langle o_1,\ldots,o_\ell\rangle$ an instant observation for M_e. Then we can define the set of knowledge base configurations that are obtained from K given the belief state S and the instant observation $\mathcal{O}$.

$$\begin{aligned} NextKB(S,\mathcal{O},\langle k_1,\ldots,k_n\rangle) = \{ & \langle k_1',\ldots,k_n'\rangle \in \mathbf{KB}_{M_e} : \\ & \text{for each } 1 \leq i \leq n, \text{ we have that} \\ & k_i' \in mng_i(app_i^{next}(S,\mathcal{O}), k_i)\} \end{aligned}$$

For each choice $\overline{d}$ of a distance function between belief states, we can define the set of knowledge base configurations which give rise to equilibria of M_e which minimize the distance to the original belief state. Let S be a belief state for M_e, $K = \langle k_1, \ldots, k_n\rangle \in \mathbf{KB}_{M_e}$ a knowledge base configuration for M_e, and $\mathcal{O}^j$ and $\mathcal{O}^{j+1}$ instant observations for M_e.

$$\begin{aligned} MinNext(S, \mathcal{O}^j, \mathcal{O}^{j+1}, K) = \{(S', K') : \\ & K' \in NextKB(S, \mathcal{O}^j, K) \text{ and} \\ & S' \in MinCost(M_e[K'], \mathcal{O}^{j+1}) \\ & \text{such that there is no} \\ & K'' \in NextKB(S, \mathcal{O}^j, K) \text{ and no} \\ & S'' \in MinCost(M_e[K''], \mathcal{O}^{j+1}) \\ & \text{with } \overline{d}(S, S'') < \overline{d}(S, S')\}. \end{aligned}$$

Again note that $MinNext$ applies minimization over all possible equilibria resulting from every element of $NextKB(S, \mathcal{O}^j, K)$. Using $MinNext$, we can now define a minimal change criterion to be applied to evolving equilibria of M_e.

Definition 11 *Let $M_e = \langle C_1, \ldots, C_n, O_1, \ldots, O_\ell\rangle$ be an eMCS, $Obs = \langle \mathcal{O}^1, \ldots, O^m\rangle$ an observation sequence for M_e, and let $S_e = \langle S^1, \ldots, S^s\rangle$ be an evolving equilibrium of M_e given Obs. We assume that $\langle K^1, \ldots, K^s\rangle$, with $K^j = \langle k_1^j, \ldots, k_n^j\rangle$, is the sequence of knowledge base configurations associated with S_e as in Definition 10. Then, S_e satisfies the* strong minimal change criterion *for M_e given Obs if, for each $1 \leq j \leq s$, the following conditions are satisfied:*

- $S^j \in MinCost(M_e[K^j], \mathcal{O}^j)$
- $(S^{j+1}, K^{j+1}) \in MinNext(S^j, \mathcal{O}^j, \mathcal{O}^{j+1}, K^j)$

We call this minimal change criterion the *strong* minimal change criterion because it applies minimization over all possible equilibria resulting from every possible knowledge base configuration in $NextKB(S, \mathcal{O}^j, K)$.

The following proposition states the desirable property that the existence of an equilibrium guarantees the existence of an equilibrium satisfying the strong minimal change criterion. We should note that this is not a trivial statement since we are combining minimization of two different elements: the cost of the operations and the distance between belief states. This proposition in fact follows from their careful combination in the definition of $MinNext$.

Proposition 3 *Let $Obs = \langle \mathcal{O}^1, \ldots, O^m\rangle$ be an observation sequence for M_e. If M_e has an evolving equilibrium of size s given Obs, then there is at least one evolving equilibrium of size s given Obs satisfying the strong minimal change criterion.*

Note that in the definition of the strong minimal change criterion, the knowledge base configurations $K \in NextKB(S^j, \mathcal{O}^j, K^j)$, for which the corresponding possible equilibria are not at a minimal distance from S^j, are not considered. However, there could be situations in which this minimization criterion is too strong. For example, it may well be that all possible knowledge base configurations in $NextKB(S^j, \mathcal{O}^j, K^j)$ are important, and we do not want to disregard any of them. In that case, we can relax the minimization condition by applying minimization individually for each knowledge base configuration in $NextKB(S^j, \mathcal{O}^j, K^j)$. The idea is that, for each fixed $K \in NextKB(S^j, \mathcal{O}^j, K^j)$ we choose only those equilibria of $M_e[K]$ which minimize the distance to S^j.

Formally, let S be a belief state for M_e, $K \in \mathbf{KB}_{M_e}$ a knowledge base configuration for M_e, and $\mathcal{O}$ an instant observation for M_e. For each distance function $\overline{d}$ between belief states, we can define the following set:

$$\begin{aligned} MinDist(S, \mathcal{O}, K) = \{ & S' : S' \in MinCost(M_e[K], \mathcal{O}) \text{ and} \\ & \text{there is no } S'' \in MinCost(M_e[K], \mathcal{O}) \\ & \text{such that } \overline{d}(S, S'') < \overline{d}(S, S')\} \end{aligned}$$

Using this more relaxed notion of minimization we can define an alternative weaker criterion for minimal change to be applied to evolving equilibria of an eMCS.

Definition 12 *Let $M_e = \langle C_1, \ldots, C_n, O_1, \ldots, O_\ell\rangle$ be an eMCS, and $Obs = \langle \mathcal{O}^1, \ldots, O^m\rangle$ be an observation sequence for M_e, and $S_e = \langle S^1, \ldots, S^s\rangle$ be an evolving equilibrium of M_e given Obs. We assume that $\langle K^1, \ldots, K^s\rangle$, with $K^j = \langle k_1^j, \ldots, k_n^j\rangle$, is the sequence of knowledge base configurations associated with S_e as in Definition 10. Then, S_e satisfies the* weak minimal change criterion *of M_e given Obs, if for each $1 \leq j \leq s$ the following conditions are satisfied:*

- $S^j \in MinCost(M_e[K^j], \mathcal{O}^j)$
- $S^{j+1} \in MinDist(S^j, K^{j+1}, \mathcal{O}^{j+1})$

In this case we can also prove that the existence of an evolving equilibrium implies the existence of an equilibrium satisfying the weak minimal change criterion. Again we note that the careful combination of the two minimizations – cost and distance – in the definition of $MinDist$, is fundamental to obtain the following result.

Proposition 4 *Let $Obs = \langle \mathcal{O}^1, \ldots, O^m\rangle$ be an observation sequence for M_e. If M_e has an evolving equilibrium of size s given Obs, then M_e has at least one evolving equilibrium of size s given Obs satisfying the weak minimal change criterion.*

We can easily prove that, in fact, the strong minimal change criterion is stronger than the weak minimal change criterion.

Proposition 5 *Let $M_e = \langle C_1, \ldots, C_n, O_1, \ldots, O_\ell\rangle$ be an eMCS, and $Obs = \langle \mathcal{O}^1, \ldots, O^m\rangle$ be an observation sequence for M_e, and $S_e = \langle S^1, \ldots, S^s\rangle$ be an evolving equilibrium of M_e given Obs. If S_e satisfies the strong minimal change criterion of M_e given Obs, then S_e satisfies the weak minimal change criterion of M_e given Obs.*

We end this section by briefly discussing a global alternative to the minimization of costs of operations. Recall that both $MinNext$ and $MinDist$ combine minimization of distances between belief states, and minimization of costs of operations. The minimization on costs is done at each time instant. Another possibility is, instead of minimizing the cost at each time instant j, to minimize the global cost of an evolving equilibrium.

We first extend the cost function over belief states to a cost function over evolving belief states. Let $S_e = \langle S^1, \ldots, S^s\rangle$ be an evolving belief state for M_e, and $Obs = \langle \mathcal{O}^1, \ldots, \mathcal{O}^m\rangle$ an observation sequence for M_e such that $m \geq s$. Then we can define the cost of S_e given Obs:

$$Cost(S_e, Obs) = \sum_{j=1}^{s} Cost(S^j, \mathcal{O}^j)$$

Let $Obs = \langle \mathcal{O}^1, \dots, \mathcal{O}^m \rangle$ be an observation sequence for M_e and let $s \leq m$. We now define the set of evolving equilibria of size s of M_e given Obs which minimize the total cost of the operations generated by it.

$MinCost(M_e, Obs, s) = \{S_e :$

S_e is an evolving equilibrium of size s of M_e given Obs and there is no evolving equilibrium S'_e of size s of M_e given Obs such that $Cost(S'_e, Obs) < Cost(S_e, Obs)\}$

Since the minimization on costs of operations is orthogonal to the minimization on distances between belief states, it would be straightforward to use $MinCost(M_e, Obs, s)$ to define the global cost versions of $MinNext$ and $MinDist$, and use them to define the respective strong and weak minimal change criteria.

5 Related and Future Work

In this paper we have studied the notion of minimal change in the context of the dynamic framework of evolving Multi-Context Systems (eMCSs) [15]. We have presented and discussed some alternative definitions of minimal change criteria for evolving equilibria of a eMCS.

Closely related to eMCSs is the framework of reactive Multi-Context Systems (rMCSs) [5, 9, 8] inasmuch as both aim at extending mMCSs to cope with dynamic observations. Three main differences distinguish them.

First, whereas eMCSs rely on a sequence of observations, each independent from the previous ones, rMCSs encode such sequences within the same observation contexts, with its elements being explicitly timestamped. This means that with rMCSs it is perhaps easier to write bridge rules that refer, e.g., to specific sequences of observations, which in eMCSs would require explicit timestamps and storing the observations in some context, although at the cost that rMCSs need to deal with explicit time which adds an additional overhead. Second, since in rMCSs the contexts that result from the application of the management operations are the ones that are used in the subsequent state, difficulties may arise in separating non-persistent and persistent effects, for example, allowing an observation to override some fact in some context while the observation holds, but without changing the context itself – such separation is easily encodable in eMCSs given the two kinds of bridge rules, i.e., with or without the $next$ operator. Finally, the bridge rules with the $next$ operator allow the specification of transitions based on the current state, such as for example one encoded by the rule $next(add(p)) \leftarrow not\, p$, which do not seem possible in rMCSs. Overall, these differences seem to indicate that an interesting future direction would be to merge both approaches, exploring a combination of explicitly timestamped observations with the expressiveness of the $next$ operator.

Also interesting is to study how to perform AGM style belief revision at the (semantic) level of the equilibria, as in Wang et al [28], though necessarily different since knowledge is not incorporated in the contexts.

Another framework that aims at modeling the dynamics of knowledge is that of evolving logic programs EVOLP [2] focusing on updates of generalized logic programs. Closely related to EVOLP, hence to eMCS, are the two frameworks of reactive ASP, one implemented as a solver *clingo* [11] and one described in [5]. The system *clingo* extends an ASP solver for handling external modules provided at runtime by a controller. The output of these external modules can be seen as the observations of EVOLP. Unlike the observations in EVOLP, which can be rules, external modules in *clingo* are restricted to produce atoms so the evolving capabilities are very restricted. On the other hand, *clingo* permits committing to a specific answer-set at each state, a feature that is not part of EVOLP, nor of eMCSs. Reactive ASP as described in [5] can be seen as a more straightforward generalization of EVOLP where operations other than $assert$ for self-updating a program are permitted. Whereas EVOLP employs an update predicate that is similar in spirit to the $next$ predicate of eMCSs, it does not deal with distributed heterogeneous knowledge, neither do both versions of Reactive ASP. Moreover, no notion of minimal change is studied for these frameworks.

The dynamics of eMCSs is one kind of dynamics, but surely not the only one. Studying the dynamics of the bridge rules is also a relevant, non-trivial topic, to a great extent orthogonal to the current development, which nevertheless requires investigation.

Another important issue open for future work is a more fine-grained characterization of updating bridge rules (and knowledge bases) as studied in [14] in light of the encountered difficulties when updating rules [24, 25, 27] and the combination of updates over various formalisms [25, 26].

Also, as already outlined in [16], we can consider the generalization of the notions of minimal and grounded equilibria [6] to eMCSs to avoid, e.g., self-supporting cycles introduced by bridge rules, or the use of preferences to deal with several evolving equilibria an eMCS can have for the same observation sequence.

Also interesting is to apply the ideas in this paper to study the dynamics of frameworks closely related to MCSs, such as those in [20, 13].

ACKNOWLEDGEMENTS

We would like to thank the referees for their comments, which helped improve this paper considerably. Matthias Knorr and João Leite were partially supported by FCT under project "ERRO – Efficient Reasoning with Rules and Ontologies" (PTDC/EIA-CCO/121823/2010). Ricardo Gonçalves was supported by FCT grant SFRH/BPD/47245/2008 and Matthias Knorr was also partially supported by FCT grant SFRH/BPD/86970/2012.

REFERENCES

[1] M. Alberti, A. S. Gomes, R. Gonçalves, J. Leite, and M. Slota, 'Normative systems represented as hybrid knowledge bases', in *CLIMA*, eds., J. Leite, P. Torroni, T. Ågotnes, G. Boella, and L. van der Torre, volume 6814 of *LNCS*, pp. 330–346. Springer, (2011).

[2] J. Alferes, A. Brogi, J. Leite, and L. Pereira, 'Evolving logic programs', in *JELIA*, eds., S. Flesca, S. Greco, N. Leone, and G. Ianni, volume 2424 of *LNCS*, pp. 50–61. Springer, (2002).

[3] D. Anicic, S. Rudolph, P. Fodor, and N. Stojanovic, 'Stream reasoning and complex event processing in ETALIS', *Semantic Web*, **3**(4), 397–407, (2012).

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

[5] G. Brewka, 'Towards reactive multi-context systems', in *LPNMR*, eds., P. Cabalar and T. C. Son, volume 8148 of *LNCS*, pp. 1–10. Springer, (2013).

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

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

[8] G. Brewka, S. Ellmauthaler, and J. Pührer, 'Multi-context systems for reactive reasoning in dynamic environments', in *ECAI*, eds., T. Schaub, G. Friedrich, and B. O'Sullivan. IOS Press, (2014). To appear.

[9] S. Ellmauthaler, ‘Generalizing multi-context systems for reactive stream reasoning applications’, in *ICCSW*, eds., A. V. Jones and N. Ng, volume 35 of *OASICS*, pp. 19–26. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, (2013).

[10] M. Gebser, T. Grote, R. Kaminski, P. Obermeier, O. Sabuncu, and T. Schaub, ‘Stream reasoning with answer set programming: Preliminary report’, in *KR*, eds., G Brewka, T. Eiter, and S. A. McIlraith. AAAI Press, (2012).

[11] M. Gebser, T. Grote, R. Kaminski, and T. Schaub, ‘Reactive answer set programming’, in *LPNMR*, eds., J. P. Delgrande and W. Faber, volume 6645 of *LNCS*, pp. 54–66. Springer, (2011).

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

[13] R. Gonçalves and J. Alferes, ‘Parametrized logic programming’, in *JELIA*, eds., T. Janhunen and I. Niemelä, volume 6341 of *LNCS*, pp. 182–194. Springer, (2010).

[14] R. Gonçalves, M. Knorr, and J. Leite, ‘Evolving bridge rules in evolving multi-context systems’, in *CLIMA XV*, eds., N. Bulling, L. van der Torre, S. Villata, W. Jamroga, and W. Vasconcelos, (2014). To appear.

[15] R. Gonçalves, M. Knorr, and J. Leite, ‘Evolving multi-context systems’, in *ECAI*, eds., T. Schaub, G. Friedrich, and B. O’Sullivan. IOS Press, (2014). To appear.

[16] R. Gonçalves, M. Knorr, and J. Leite, ‘Towards efficient evolving multi-context systems (preliminary report)’, in *ReactKnow 2014*, (2014). To appear.

[17] M. Homola, M. Knorr, J. Leite, and M. Slota, ‘MKNF knowledge bases in multi-context systems’, in *CLIMA*, eds., M. Fisher, L. van der Torre, M. Dastani, and G. Governatori, volume 7486 of *LNCS*, pp. 146–162. Springer, (2012).

[18] V. Ivanov, M. Knorr, and J. Leite, ‘A query tool for $\mathcal{EL}$ with non-monotonic rules’, in *ISWC*, eds., H. Alani, L. Kagal, A. Fokoue, P. T. Groth, C. Biemann, J. Parreira, L. Aroyo, N. F. Noy, C. Welty, and K. Janowicz, volume 8218 of *LNCS*, pp. 216–231. Springer, (2013).

[19] M. Knorr, J. Alferes, and P. Hitzler, ‘Local closed world reasoning with description logics under the well-founded semantics’, *Artif. Intell.*, **175**(9-10), 1528–1554, (2011).

[20] M. Knorr, M. Slota, J. Leite, and M. Homola, ‘What if no hybrid reasoner is available? Hybrid MKNF in multi-context systems’, *J. Log. Comput.*, (2013).

[21] F. Lécué and J. Pan, ‘Predicting knowledge in an ontology stream’, in *IJCAI*, ed., F. Rossi. IJCAI/AAAI, (2013).

[22] B. Motik and R. Rosati, ‘Reconciling description logics and rules’, *J. ACM*, **57**(5), (2010).

[23] F. Roelofsen and L. Serafini, ‘Minimal and absent information in contexts’, in *IJCAI*, eds., L. Kaelbling and A. Saffiotti, pp. 558–563. Professional Book Center, (2005).

[24] M. Slota and J. Leite, ‘On semantic update operators for answer-set programs’, in *ECAI*, eds., H. Coelho, R. Studer, and M. Wooldridge, volume 215 of *Frontiers in Artificial Intelligence and Applications*, pp. 957–962. IOS Press, (2010).

[25] M. Slota and J. Leite, ‘Robust equivalence models for semantic updates of answer-set programs’, in *KR*, eds., G. Brewka, T. Eiter, and S. A. McIlraith. AAAI Press, (2012).

[26] M. Slota and J. Leite, ‘A unifying perspective on knowledge updates’, in *JELIA*, eds., L. del Cerro, A. Herzig, and J. Mengin, volume 7519 of *LNCS*, pp. 372–384. Springer, (2012).

[27] M. Slota and J. Leite, ‘The rise and fall of semantic rule updates based on SE-models’, *TPLP*, (2014). To appear.

[28] Y. Wang, Z. Zhuang, and K. Wang, ‘Belief change in nonmonotonic multi-context systems’, in *LPNMR*, eds., P. Cabalar and T. C. Son, volume 8148 of *LNCS*, pp. 543–555. Springer, (2013).

Towards a Simulation-Based Programming Paradigm for AI applications[1]

Jörg Pührer[2]

Abstract. We present initial ideas for a programming paradigm based on simulation that is targeted towards applications of artificial intelligence (AI). The approach aims at integrating techniques from different areas of AI and is based on the idea that simulated entities may freely exchange data and behavioural patterns. We define basic notions of a simulation-based programming paradigm and show how it can be used for implementing AI applications.

1 Introduction

Artificial intelligence (AI) is a wide field of research in which many different outstanding techniques have been developed and refined over the last decades [15]. Naturally, the question arises how to couple or integrate different subsets of these accomplishments. Besides many approaches to couple specific individual methods, a need for a wider integration of different AI techniques has been identified in the area of artificial general intelligence [16, 10]. Here, the goal is to build strong AI systems, i.e., reach human level intelligence. Arguably, integration of existing techniques is also desirable for less ambitious AI applications (that we aim for), consider for instance the realisation of intelligent opponents in computer games as a motivating example. As a side remark, note that current solutions for game AI rarely make use of techniques from reseach in AI but are often ad-hoc, based on hardcoded strategies, and incapable of learning.

Simulation has been used in different fields of AI (such as agent-based systems [12, 18] or evolutionary computation [4]) for achieving intelligent behaviour. The rationale is that many aspects of intelligent behaviour are complex and not well understood but can be observed to emerge when the environment in which they occur is simulated adequately. In this work, we propose to use a simulation environment for realising AI applications that offers an easy way to integrate existing methods from different areas of AI such as computational intelligence, symbolic AI, or statistical methods. In particular, we present the basic cornerstones of a simulation-based programming paradigm (SBP) and demonstrate how it can be used to model different use cases for intelligent systems. The basic idea of SBP is to simulate an environment of interacting *entities* driven by concurrent processes. Entities are not grouped in types or classes and contain data as well as *transition descriptions* that define possible behaviour. Both, the behaviour and data associated to entities are subject to change which allows for learning techniques.

In the proposed approach, different points of views, hypothetical reasoning, or different granularities of simulation can be addressed by using multiple *worlds* refering to (not necessarily) the same entities. For example, the beliefs of an agent which is modelled by an entity can be represented by a world that might differ from the data available in another world that represents an objective reality. This gives rise for epistemic reasoning capabilities, where e.g., an agent A thinks about what agent B thinks and acts upon these beliefs.

The remainder of the paper is organised as follows. Next, we introduce the basic notions of a simulation-based programming paradigm. Section 3 discusses how to model different scenarios of AI applications in the approach. We show how behaviour can be exchanged between entities and discuss how evolutionary processes can emerge. Moreover, we demonstrate the use of different worlds for hypothetical reasoning, expressing and exchanging different beliefs about facts and processes, and for using different granularities of simulation. In Section 4 we discuss interface considerations for transition descriptions. After that, Section 5 addresses the issue of maintaining consistency when data is updated by concurrent processes. Section 6 discusses the relation to existing techniques including differences to agent-based approaches and object-oriented programming. The paper is concluded in Section 7 with a short summary and an outlook on future work.

2 Simulation-Based Programming

In this section we explain the architecture of the proposed simulation-based programming paradigm on an abstract level.

An SBP system deals with different *worlds*, each of which can be seen as a different point of view. The meaning of these worlds is not pre-defined by SBP, e.g., the programmer can decide to take an objectivistic setting and consider one world the designated real one or treat all worlds alike. Different worlds allow for example to model the beliefs of an agent as in an agent-based approach. Other applications are hypothetical reasoning or realising different granularities of abstraction for efficiency, e.g., parts of the simulation that are currently in focus can be manipulated by a world that offers a more precise simulation whereas parts out of focus are handled by another world that implements an approximation (see Section 3).

A world contains a set of named *entities* which are the primary artifacts of SBP. Entities may have two sorts of named attributes: *data entries* which correspond to arbitrary data (including references to other entities) and *transition descriptions* which define the behaviour of the entities over time. The name of an entity has to be unique with respect to a world and serves as a means to reference the entity, however the same entity may appear in different worlds with potentially different attributes and attribute values. Transition descriptions can be seen as the main source code elements in the approach and they are, similar to the data entries, subject to change during runtime. This

[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: puehrer@informatik.uni-leipzig.de

allows for a dynamic setting in which the behaviour of entities can change over time, e.g., new behaviour can be learned, acquired from other entities, or shaped by evolutionary processes. We do not propose a particular language or programming paradigm for specifying transition descriptions. It might, on the contrary, be beneficial to allow for different languages for different transition descriptions even within the same simulation. For instance, a transition description implementing sorting can be realised by some efficient standard algorithm in an imperative language, while another transition description that deals with a combinatorial problem with many side constraints uses a declarative knowledge representation approach like answer-set programming (ASP) [8, 13] in which the problem can be easily modelled. Declarative languages are also quite useful in settings where the transition description should be modified at runtime (as mentioned above) as they often allow for easy changes. That is, because problem descriptions in these languages are typically concise and many declarative languages offer high elaboration tolerance [9], i.e., little changes of the problem statement require only few adaptations of the source code that solves the problem.

We require transition descriptions—in whatever language they are written—to comply to a specific interface that allows us to execute them in asynchronous *processes*. In particular, the output of a transition contains a set of updates to be performed on worlds, entities, data entries, and transition descriptions. When a transition has finished, per entity, these changes are applied in an atomic transaction that should leave the entity in a consistent state (provided that the transition description is well designed).

As mentioned, transition descriptions are executed in processes. Each process is associated with some entity and runs a transition description of this entity in a loop. A process can however decide to terminate itself or other processes at any time, initiate other processes, and wait for their results before finishing their own iteration.

We assume an infinite set $\mathcal{N}$ of names and say that a concept c is named if it has an associated name $n_c \in \mathcal{N}$. We frequently use the data structure of a *map*, which is a set M of pairs $\langle n, v\rangle$ such that v is a named object, $n = n_v$, and $\langle n, v_1\rangle, \langle n, v_2\rangle \in M$ implies $v_1 = v_2$. With slight abuse of notation we write $v \in M$ for $\langle n_v, v\rangle \in M$. In the following we describe how the concepts discussed above are related more formally. To this end, we assume the availability of a set Σ of semantics for transition functions that will be explained later on.

Definition 1

- *A* transition description *is a pair $t = \langle \mathrm{sc}, \sigma\rangle$, where* sc *is a piece of source code, and $\sigma \in \Sigma$ is a semantics.*
- *A* process *is a tuple $p = \langle t, \mathtt{t}_b\rangle$, where t is a transition description and $\mathtt{t}_b$ is a timestamp marking the begin of the current transition.*
- *An* entity *is a tuple $e = \langle D, T, P\rangle$, where D is a map of named data, T is a map of named transition descriptions, and P is a map of named processes. Entries of D,T, and P are called* properties *of e.*
- *A* world *is a map of named entities.*
- *An* SBP configuration *is a map of named worlds.*

We assume a pre-specified set Υ of *updates* which are descriptions of what changes should be made to an SBP configuration together with a fixed *update function* υ that maps an SBP configuration, an entity name, the name of a world, and a set of updates, to a new SBP configuration.

Definition 2 *A result structure for a process p is a tuple $r = \langle U, b_c\rangle$, where $U \subseteq \Upsilon$ is a set of updates and b_c is one of the boolean values true or false that decides whether process p should continue with another transition.*

A semantics *$\sigma \in \Sigma$ is a function that maps a piece of source code, an SBP configuration, the name of a world, the name of an entity, and a timestamp to a result structure.*

Dynamic Behaviour For presentational reasons we forgo giving a precise definition of the runtime semantics of an SBP system that would require heavy notation but describe the behaviour of the system on a semi-formal level using the concepts introduced above.

For running an SBP system we need an initial collection of worlds. Thus, let c^0 be an SBP configuration.[3] We assume a discrete notion of time where a run of the system starts at time 0, and that $c^{\mathtt{t}}$ denotes the SBP configuration of each point in time $\mathtt{t}$ during a run.

At every time $\mathtt{t}$ during the run of an SBP system the following conditions hold or changes are performed:

- for each process p such that $p = \langle t, \mathtt{t}_b\rangle \in P$ for some entity $e = \langle D, T, P\rangle$ in some world w of $c^{\mathtt{t}}$ there are three options:
  - (i) p continues, i.e., $p \in P'$ for entity $e' = \langle D', T', P'\rangle$ with $n_{e'} = n_e$ in world $w' \in c^{\mathtt{t}+1}$ with $n_{w'} = n_w$;
  - (ii) p can be cancelled, i.e., $p \notin P'$ for P' as in Item (i);
  - (iii) p has finished its computation, i.e., the result of p, namely
  $$r_p = \sigma(\mathrm{sc}, c_0, n_w, n_e, \mathtt{t}) = \langle U, b_c\rangle$$
  is computed for $t = \langle \mathrm{sc}, \sigma\rangle$. The updates that were computed by p are added to a set $U^{\mathtt{t}}_{n_e,n_w}$ of updates for entity e in world w, i.e., $U \subseteq U^{\mathtt{t}}_{n_e,n_w}$.
  The original process p is deleted similar as in Case (ii). However, if $b_c = \mathit{true}$ then, at time point $\mathtt{t}{+}1$ after $\mathtt{t}$, a new iteration of the process starts, i.e., if there is an entity $e' = \langle D', T', P'\rangle$ with $n_{e'} = n_e$ in world $w' \in c^{\mathtt{t}+1}$ with $n_{w'} = n_w$ and there is a transition description $t' \in T'$ such that $n_{t'} = n_t$, then P' contains a new process $p' = \langle t', \mathtt{t}+1\rangle$ with $n_{p'} = n_p$.

  Why and when a process continues, is cancelled, or finishes is not further specified at this point because, intuitively, in an SBP system this depends on decisions within the process itself, other processes, and the available computational resources.
- starting with $c'_1 = c^{\mathtt{t}}$, iteratively, for every item e in some world w of $c^{\mathtt{t}}$ where the set $U^{\mathtt{t}}_{n_e,n_w}$ of collected updates is non-empty, a new SBP configuration is computed by the update function:
  $$c'_{i+1} = \upsilon(c'_i, n_e, n_w, U^{\mathtt{t}}_{n_e,n_w})$$
  The SBP configuration c'_n computed in the last iteration becomes the new configuration $c^{\mathtt{t}+1}$ of the system for the next point in time. We do not make assumptions about the order in which updates are applied for now and discuss the related topic of consistency handling in Section 5.

Using names (for worlds, entities, and properties) allows us to speak about concepts that change over time. For example, if e_0 is an entity $e_0 = \langle D, T, P\rangle$ in some world at time 0 and some data is added to D for time 1 then, technically, this results in another entity $e_1 = \langle D', T, P\rangle$. As our intention is to consider e_1 an updated version of e_0 we use the same names for both, i.e., $n_{e_0} = n_{e_1}$. In the subsequent work we will sometimes refer to concepts by their names.

[3] In practice, c^0 could by convention consist of a single world with a process running a transition description for initialisation, similar to a main function.

Along these lines, we introduce the following path-like notation using the .-operator for referring to SBP concepts in a run. Assuming a sequence of SBP configurations $c^0, c^1, \dots$ in a run as above, we refer to

- the world $w_i \in c^{\mathrm{t}}$ by $n^{\mathrm{t}}_{w_i}$,
- the entity $e = \langle D, T, P\rangle \in w_i$ by $n^{\mathrm{t}}_{w_i}.n_e$,
- the property x in D, T, or P by $n^{\mathrm{t}}_{w_i}.n_e.n_x$ (we assume that no entity has multiple data entries, transition descriptions, or processes of the same name).

If clear from the context, we drop the name of the world or entity and apply the timestamp directly to entities or properties, or also drop the timestamp if not needed or clear.

3 Modelling AI Applications in SBP

The concepts introduced in the previous section provide an abstract computation framework for SBP. Next, we demonstrate how to use it for modelling AI scenarios. Note that in this section we will use high-level pseudo code for expressing the source code of transition descriptions and emphasise that in an implementation we suggest to use different high-level programming languages tailored to the specific needs of the task handled by the transition description. We discuss interface considerations for these embeddings of other formalisms in Section 4.

Following the basic idea, i.e., simulating an intended scenario on the level of the programming language, entities in SBP are meant to reflect real world entities. In contrast to objects as in object-oriented programming (cf. Section 6), entities are not grouped in a hierarchy of classes. Classes are a valuable tool in settings that require clear structures and rigorously defined behaviour. However, in the scenarios we target, the nature of entities may change over time and the focus is on emerging rather than predictable behaviour. For example, in a real-world simulation, a town may become a city and a caterpillar a butterfly, etc., or, in fictional settings (think of a computer game) a stone could turn into a creature or vice versa. We want to directly support metamorphoses of this kind, letting entities transform completely over time regarding their data as well as their behaviour (represented by transition descriptions). Instead of using predefined classes, type membership is expressed by means of properties in SBP, e.g., each entity n_e may have a data entry n_e.types that contains a list of types that n_e currently belongs to.

Example 1 *We deal with a scenario of a two-dimensional area, represented by a single SBP world w, where each entity $n_w.n_e$ may have a property n_e.loc with values of form $\langle X, Y\rangle$ determining the location of n_e to be at coordinates $\langle X, Y\rangle$. The area is full of chickens running around, each of which is represented by an entity. In the beginning, every chicken* ch *has a transition description* ch.mvRand *that allows the chicken to move around with the pseudo code:*

```
wait(randomValue(1..5000))
dir = randomValue(1..4)
switch{dir}
 case 1: return {'mv_up'}
 case 2: return {'mv_right'}
 case 3: return {'mv_down'}
 case 4: return {'mv_left'}
```

The transition first waits for a random amount of time and chooses a random direction for the move, represented by the updates mv_up, mv_right, $\dots \subseteq \Upsilon$. *The semantics of* mvRand *always returns $\langle U, true\rangle$, where U contains the update (the direction to move) and $true$ indicates that after the end of the transition there should be a new one. When the update function υ is called with one of the* mv *updates it changes the value of* ch.loc*, e.g., if* ch^{t}.loc *has value* $\langle 3, 5\rangle$ *and the update is* mv_left *then* $\mathrm{ch}^{\mathrm{t+1}}$.loc *has value* $\langle 2, 5\rangle$.

Besides randomly walking chickens, the area is sparsely strewn with corn. Corn does not move but it is eaten by chicken. Hence, each chicken ch *has another transition description* eat *with the code:*

```
if there is some entity en in myworld with
 en.loc = my.loc and
 en.types contains 'corn'
then
 return {'eatCorn(en)'}
```

Here, we assume that using the keyword `my` *we can refer to properties of the entity to which the transition description belongs (*ch *in this case). Furthermore,* `myworld` *refers to the world in which this entity appears. Also here, every iteration of* eat *automatically starts another one. For an update* eatCorn(en)*, the update function*

- *deletes the location entry* en.loc *of the corn and*
- *notifies the corn entity that it was eaten by setting the data entry* en.eatenBy *to* ch *(we will need the information which chicken ate the corn later) and adding a process to the corn entity with the transition description* en.beenEaten *that is specified by the following pseudo code:*

  ```
  return {'delete_me'}
  ```

  that causes the corn to delete itself from the area. Unlike for the other transition descriptions, a process with en.beenEaten *lasts for only a single iteration.*

Assume we have an initial SBP configuration $c^0 = \langle w\rangle$, where every chicken entity in w has an active process named move *with transition description* mvRand *and a process with transition description* eat*. Then, a run simulates chickens that run around randomly and eat corn on their way.*

While Example 1 illustrates how data is changed over time and new processes can be started by means of updates, the next example enriches the scenario with functionality for learning new behaviour.

Example 2 *We extend the scenario of Example 1 to a fairy tale the setting by assuming that among all the corn entities, there is one dedicated corn named* cornOfWisdom *that has the power to make chickens smarter if they eat it.*

This cornOfWisdom *has a transition description* cornOfWisdom.mvSmart*:*

```
wait(randomValue(1..1000))
en is an entity in myworld where
 en.types contains 'corn' and
 there is no other entity en'
               in myworld where
   en'.types contains 'corn'  and
   distance(en'.loc,my.loc) <
           distance(en.loc,my.loc)
let my.loc=(myX,myY)
let en.loc=(otherX,otherY)
distX = otherX - myX
distY = otherY - myY
```

```
if |distX| > |distY| then
  if distX > 0 then
   return {'mv_right'}
  else
   return {'mv_left'}
 else
  if distY > 0 then
   return {'mv_down'}
  else
   return {'mv_up'}
```

Intuitively, this transition causes an entity to move towards the closest corn rather than walking randomly as in mvRand*. Another difference is that* mvSmart *processes have shorter iterations on average as the range of the random amount of time to wait is smaller. The* cornOfWisdom *does not have active processes for this transition definition itself but can pass it on to everyone who eats it. This is defined in the transition* cornOfWisdom.beenEaten *that differs from the* beenEaten *transition description of other corn:*

```
 ch = my.eatenBy
 return {'delete_me',
       'copyTransition(mvSmart,ch)',
       'changeTransition(ch.move,mvSmart)'}
```

Besides issueing the delete_me *update as it is the case for normal corn, the update* copyTransition(mvSmart, ch) *copies the* mvSmart *transition description from the* cornOfWisdom *to the chicken by which it was eaten. The update* changeTransition(ch.move, mvSmart) *changes the* move *process of the chicken to use its new* mvSmart *transition description instead of* mvRand*. Thus, if a chicken happens to eat the* cornOfWisdom *it will subsequently have a better than random strategy to catch some corn.*

Having means to replace individual behavioural patterns, as in the example allows for modelling evolutionary processes in an easy way. For example, if the chicken scenario is modified in a way that chicken which do not eat corn regularly will die, a chicken that ate the cornOfWisdom has good chances to survive for a long period of time. Further processes could allow chickens to reproduce when they meet such that baby chicken may inherit which transition description to use for moving from one of the parents. Then, most likely, chicken using mvSmart will be predominant soon.

The next example illustrates the use of worlds for hypothetical reasoning.

Example 3 *Entity* barker *represents a waiter of an international restaurant in a tourist area trying to talk people on the street into having dinner in his restaurant. To this end,* barker *guesses what food they could like and makes offers accordingly. We assume an SBP configuration in which for every entity* h *that represents a human, there is a world* $\mathrm{w_h}$ *that represents the view of the world of this human. The following transition description* barker.watchPeople *allows* barker *to set the eating habits of passer-by in his world* w_{barker} *using country stereotypes.*

```
wait(randomValue(50))
let en be an entity in myworld where
  en.loc near my.loc
  en.types contains 'human'
  en.eatingHabits = unknown
 country = guess most likely
          home country of en
 prototype = myworld.country.inhPrototype
 return {'setEatingHabits(en,propotype)',
        'setPotentialCustomer(en)'}
```

For every country, $\mathrm{w_{barker}}$ *contains a reference* inhPrototype *to an entity representing a typical person from this country. The update* setEatingHabits(p1, p2) *copies transition descriptions and data properties that are related with food from person* p2 *to person* p1*. Moreover, the update* setPotentialCustomer(p) *lets* barker *consider entity* p *to be a potential customer. In order to choose what to offer a potential customer, the waiter thinks about what kind of food the person would choose (based on his stereotypes). This is modelled via the following transition* barker.makeOffer*:*

```
let cus be a potential customer in myworld
w' = copy of myworld
w'.cus.availableFood =
                  restaurant.availableFood
w'.cus.hungry = true
intermediate return {addWorld(w'),
          startProcess(w'.cus.startDinner)}
when process w'.cus.foodSelected is finished
 food = w'.cus.selectedFood
 return {praiseFood(food), deleteWorld(w')}
```

To allow for hypothetical reasoning by the waiter, a temporary copy $\mathrm{w'}$ *of the world* $\mathrm{w_{barker}}$ *is created. The sole purpose of this world is to simulate the customer dining. We use a temporary world since the simulation uses the same transition descriptions that drive the overall simulation. For example, if we would use* $\mathrm{w_{barker}}$ *instead, this would mean that* barker *thinks that the customer is actually having dinner. If we would use the world of the customer that would mean that the customer thinks she is having dinner and so on.*

After creating $\mathrm{w'}$*, the transition description defines that the food available to the version of the customer in* $\mathrm{w'}$ *is exactly the food that is on the menu of the restaurant and the customer is set to be hungry in the imagination of the waiter. Then,* $\mathrm{w'}$ *is added to the SBP configuration and a process for* $\mathrm{w'}$.cus *is started using the transition description* startDinner *that lets enitity* cus *start dining in* $\mathrm{w'}$*. Note that the keyword* `intermediate return` *in the pseudo code is a convenience notation that allows for manipulating the SBP configuration during a transition which is strictly speaking not allowed in the formal framework of Section 2. Nevertheless, the same behaviour could be accomplished in a conformant way by splitting* barker.makeOffer *into two separate transition descriptions that are used in an alternating scheme. As soon as the customer chooses some food in the simulation, transition* barker.makeOffer *is notified. It continues with reading which food has been chosen in the hypothetical setting. Finally, the update* praiseFood(food) *causes the waiter to make an offer for the chosen food in the subsequent computation, whereas* deleteWorld($\mathrm{w'}$) *deletes the temporary world.*

Note that copying worlds as done in Example 3 does not necessarily imply copying all of the resources in this world within an implementation of an SBP runtime engine (cf. the final discussion in Section 7). Moreover, it will sometimes be useful to adjust transition definitions in the copied world. For instance, when a transition definition deliberately slows down the pace of the simulation as it is done in Examples 1 and 2 using the `wait` statement, it would make sense to reduce the waiting time in a world for hypothetical reasoning. Another need for adapting a copied world is mentioned in Section 4 in the context of externally controlled processes.

Example 4 *We continue Example 3 by assuming an SBP configuration where a tourist, Ada, passes by the waiter. His process for transition* barker.watchPeople *classifies Ada by her looks to be an Englishwoman. After that, the process for* barker.makeOffer *starts hypothetical reasoning about Ada having dinner. Following the stereotypes of* barker *about English eating habits, the process reveals that Ada would go for blood pudding which he offers her subsequently. However, Ada is not interested in this dish as she is vegetarian. She explains her eating habits to the waiter, modelled by the following transition description* ada.explainEatingHabits*:*

```
let pers be current discussion partner
                              in myworld
if pers offers food containing meat then
 let w_pers be the world of pers
 return {'setEatingHabits(w_pers.me,
                         myworld.me)'}
```

Here, the update setEatingHabits *that we used also in the previous example, the transition will overwrite the food related properties of the entity representing Ada in the world of* barker *with her actual eating habits. If* barker *runs the dining simulation again for making another offer the result will match the real choices of Ada.*

The last two examples showed how worlds can be used to express different modalities like individual points of views or hypothetical scenarios. Next, we sketch a setting where different worlds represent the same situation at different granularities.

Example 5 *Consider a computer game in which the player controls a character in an environment over which different villages are distributed. Whenever the character is close to or in a village the inhabitants of the village should be simulated following their daily routines and interacting with the player. However, as the game environment is huge, simulating all inhabitants of each village at all times is too costly. The problem can be addressed by an SBP configuration that has two worlds,* $\mathrm{w(v)_{act}}$ *and* $\mathrm{w(v)_{apx}}$ *for each village* v*. Intuitively,* $\mathrm{w(v)_{act}}$ *simulates the village and its people in all details but has only active processes while the player is closeby. The world* $\mathrm{w(v)_{apx}}$ *approximates the behaviour of the whole village, e.g., increasing or shrinking of the population, economic output and input, or relations to neighbour villages, based on statistics and it has only active processes whenever the player is not around. Whenever the player enters a village, a process is started that synchronises the world* $\mathrm{w(v)_{act}}$ *with the current state of the village in* $\mathrm{w(v)_{apx}}$*, e.g., by deleting or adding new inhabitants or shops. Moreover, it starts processes in* $\mathrm{w(v)_{act}}$ *and cancels processes in* $\mathrm{w(v)_{apx}}$*. Another type of process is started when the player leaves again, that performs an opposite switch from* $\mathrm{w(v)_{apx}}$ *to* $\mathrm{w(v)_{apx}}$ *being active.*

While learning by simply copying transition descriptions from different other entities as shown earlier already allows for many different behaviour patterns to emerge, an SBP system can also be designed such that new transition descriptions are created at runtime. For example, by implementing mutation or crossing-over operators for decision descriptions, it is easy to realise genetic programming [6] principles in SBP. Another source for new transition descriptions is related to an important challenge in AI: learning behaviour by watching the environment. In an SBP framework it is easy to incorporate existing learning techniques [14, 1] by means of transition descriptions. Behaviour acquired by processes executing such transitions can then also be represented by means of transition descriptions and distributed to entities.

4 Interface Considerations for Transition Descriptions

As we want to allow for different formalisms to be used for transition descriptions it is important that they are able to interact smoothly. This is essentially already reached if their semantics respects the interface of Definition 2. As different transitions communicate by reading and writing from and to the SBP configuration their formalisms do not need to be aligned in a different way. It is certainly necessary, however, that they use the same format for property values.

The examples in the previous section already show some of the features that we think are useful in a language realising a transition description. For one, it is valuable to have generic keywords standing for the name of the entity to which the transition belongs to and its world, like `me` and `myworld` in the examples. This way, if the transition description is copied to another entity or the same entity in another world it dynamically works with the other entity or world.

We do not define an explicit user interface for SBP systems but suggest that interaction of the user or another external source with an SBP system by means of externally controlled processes: A transition description can use a dedicated 'external semantics' where the result structure returned for every transition is provided by the user or an external system. Following this approach, an SBP system acts as a reactive approach that is influenced by events from its environment. By having the decision which parts are controlled externally and which ones within the system on the level of transition descriptions allows for having parts of the behaviour of an entity partially controlled by the user and partially by the system. Moreover, as decisions descriptions can be replaced at runtime it is also possible to take control over aspects previously handled by the system and, conversely, formerly externally controlled transitions can be automatised. This way one can replace, e.g., a human player in a computer game by an AI player or vice versa. Naturally, this requires a proper modelling. For instance, in a simulation where worlds are copied for hypothetical reasoning like in the restaurant examples, a modeller would probably want to replace human controlled processes by automated ones in the copied world. Otherwise, the user would have to provide additional input for the hypothetical scenario.

In the context of the model-view-controller pattern, an SBP configuration represents the model and an SBP runtime engine corresponds to the controller. We suggest to handle the view outside of SBP, although it would be interesting to explore whether it is beneficial to also model graphical user interfaces inside SBP.

5 Consistency of Data

A key element of SBP is concurrent programming. Thus, a natural question is how problems regarding concurrent access on data and consistency of data are handled in the approach. Conceptionally, conflicting updates that occur at the same time instant do not cause a technical problem as the update function υ resolves an arbitrary set of updates to a valid follow-up SBP configuration. In practice, however, the functionality of this function has to be implemented and conflicts (e.g., deleting and changing a property of the same name at the same time) have to be addressed. Here, techniques for concurrency control in databases [2] could be useful. Moreover, it might be worthwhile to give the modeller means for specifying how to resolve individual conclicts by a dedicated language. Besides technically conflicting updates on data, another issue are semantical inconsistencies, i.e., data whose meaning with respect to the modelled problem domain is conflicting. As an example, consider an SBP configuration modelling a banana and two monkeys and assume that each monkey has a transition description that lets him grab the banana whenever it is laying on the ground. Now suppose that both monkeys detect the banana at slightly different times and their grabbing processes start. Then, after the first monkey has taken the banana, the process of the other monkey is still going on and, depending on the concrete modelling, it could happen that the system is in a state where each monkey is believed to have the banana. We argue that consistency problems of this kind should be tackled on the level of modelling rather than by the underlying computational framework as there are many types of issues that have to be addressed in different ways and also in the real world two monkeys could believe that they succeeded in getting a banana for a short period of time. One solution in the example could be that the successful grabbing process of the first monkey cancels that of the other or that the grabbing update is implemented in a conditional way such that grabbing takes only place if the banana is still in place at the time instant when the process has finished. Although one cannot expect that problems of this kind are handled automatically, a concurrent formalism should allow for addressing them in an easy way. A major point for future work on SBP is to explore best practices for avoiding inconsistencies in the first place by adequate modelling. Moreover, situations should be singled out in which inconsistency avoidance requires much modelling effort but the respective problem could be handled by adding features to the framework.

6 Influences and Relation to Existing Approaches

A goal of our approach is to integrate the use of different AI techniques in a dynamic framework. Here, a main mechanism of integration is using existing AI formalisms (e.g., ASP, planning techniques, etc.) for solving subproblems by means of transitions descriptions. This is similar in spirit to the use of different context formalisms in recent reactive forms of heterogenous multi-context systems [3, 5].

The idea of using multiple worlds for different points of view and modalities is loosely related to the possible world semantics of modal logics [7].

Evolutionary processes are intrinsic to many types of simulation. In, genetic algorithms [11] the fitness of individuals is typically rated by a dedicated fitness function, whereas the most obvious approach in SBP is simulating natural selection by competition in the simulated environment, as discussed in the context of the chicken scenario after Example 2. The evolution of behaviour is related to genetic programming [6] where computer programs are shaped by evolutionary processes. Besides processes for the evolution of data and behaviour also other techniques that are frequently used in meta-heuristics, like swarm intelligence methods can be modelled and mixed in SBP in a natural way.

Agent-based systems (ABS) [12, 18] share several aspects with SBP like the significance of emerging behaviour when entities are viewed as agents. This view, however, is not adequate for all types of entities in the SBP setting as an entity could also represent objects like stones, collections of entities, or intangible concepts like 'the right to vote' which should not be seen as agents. Moreover, agents interact via explicit acts of communication that can but need not be modelled in an SBP configuration. Thus, we see SBP conceptionally one level below ABS, i.e., SBP languages can be used for implementing ABSs rather than being ABSs themselves.

Focuses of integration efforts in artificial general intelligence are communication APIs [19] and design methodology [17].

The shift of paradigm from procedural to object-oriented programming (OOP) can be seen as a step towards structuring programming to be more like the real world: in OOP, a world of objects of defined types. In particular, objects are instances of classes that are organised in a hierachy of classes in which data structures and behaviour can be inherited from top to bottom. While classes are well-suited for applications that require a clear structuring of data, they also impose a rigid corset on their instances: the data and behaviour of objects is in essence limited to what is pre-defined in their class. Moreover, the type of object is defined on instantiation and does not change during runtime. In contrast, the behaviour and data of entities in SBP can be changed over time. Inheritance in SBP works on the individual level: entities can pass their transition descriptions and data entries to fellow entities. Thus, compared to OOP, inheritance is not organised in a hierarchical way. The underlying motivation is to follow the main idea of simulating real world objects, taking the stance that entities in nature are individuals that are not structured into distinct classes per se. Instead of the instantiation of classes for generating new objects, an important strategy for obtaining new entities in SBP is the prototype pattern: copying an entity that is closest to how the new entity should be like. As discussed in Section 3, other techniques for creating new objects are sexual reproduction or random mutation.

Another difference between typical object-oriented languages and the SBP approach is related to the control flow. Like procedural programming, OOP programs are executed in an imperative way. Typically, a run of a program in OOP starts with an entry method executed in a main thread from which child threads can be spawned in order to obtain concurrency. When a method calls another, it is per default executed in the same thread, i.e., the execution of the calling method is paused until the called method has finished. In SBP, there is no main thread and each transition runs in an independent process. Thereby, the approach exploits the trend to concurrent computing due to which simulation became feasible for many applications.

7 Conclusion

In this work we proposed an approach for using simulation as a programming paradigm. The cornerstones of the approach are

- typeless entities
- different worlds for different views on reality
- behaviour defined by heterogenous concurrent services
- exchange of behavioural patterns and individual inheritance

The main contribution of the paper is not a ready-to-use language but an initial idea for an architecture to combine these principles in a simulation-based programming paradigm. Clearly, there are many important aspects that need to be addressed when putting SBP in practice. Examples are the choice of data structures for entities, their interface when using different formalisms in transition definitions, and consistency of data as discussed in Section 5.

As a next step we want to explore the capabilities of different formalisms as a transition description language starting with ASP and identify different modelling patterns for important problems. A major goal is the development of a prototype SBP runtime engine which opens a wide field for further research: An important point is how to manage resources in SBP systems in which multiple worlds and entities share identical or slightly different data and processes. Efficiency requirements could necessitate mechanisms for sharing resources, e.g., by only keeping track of differences when a world or entity is cloned.

REFERENCES

[1] Brenna Argall, Sonia Chernova, Manuela M. Veloso, and Brett Browning, 'A survey of robot learning from demonstration', *Robotics and Autonomous Systems*, **57**(5), 469–483, (2009).

[2] Philip A. Bernstein, Vassos Hadzilacos, and Nathan Goodman, *Concurrency Control and Recovery in Database Systems*, Addison-Wesley, 1987.

[3] Gerhard Brewka, Stefan Ellmauthaler, and Jörg Pührer, 'Multi-context systems for reactive reasoning in dynamic environments', in *Proc. ECAI'14*, (2014). To appear.

[4] L.J. Fogel, A.J. Owens, and M.J. Walsh, *Artificial intelligence through simulated evolution*, Wiley, Chichester, WS, UK, 1966.

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

[6] John R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press, Cambridge, MA, USA, 1992.

[7] Saul Kripke, 'A completeness theorem in modal logic', *J. Symb. Log.*, **24**(1), 1–14, (1959).

[8] Victor W. Marek and Mirosław Truszczyński, 'Stable models and an alternative logic programming paradigm', in *In The Logic Programming Paradigm: a 25-Year Perspective*, eds., Krzysztof R. Apt, Victor W. Marek, Mirosław Truszczyński, and David S. Warren, 375–398, Springer, (1999).

[9] John McCarthy, 'Elaboration tolerance', in *Proceedings of the 4th Symposium on Logical Formalizations of Commonsense Reasoning* (*Common Sense 98*), pp. 198–217, (1998).

[10] Marvin Minsky, Push Singh, and Aaron Sloman, 'The st. thomas common sense symposium: Designing architectures for human-level intelligence', *AI Magazine*, **25**(2), 113–124, (2004).

[11] Melanie Mitchell, *An introduction to genetic algorithms*, MIT Press, 1998.

[12] Muaz Niazi and Amir Hussain, 'Agent-based computing from multi-agent systems to agent-based models: a visual survey', *Scientometrics*, **89**(2), 479–499, (2011).

[13] Ilkka Niemelä, 'Logic programs with stable model semantics as a constraint programming paradigm', *Annals of Mathematics and Artificial Intelligence*, **25**(3-4), 241–273, (1999).

[14] Santiago Ontañón, José L. Montaña, and Avelino J. Gonzalez, 'A dynamic-bayesian network framework for modeling and evaluating learning from observation', *Expert Syst. Appl.*, **41**(11), 5212–5226, (2014).

[15] Stuart J. Russell and Peter Norvig, *Artificial Intelligence - A Modern Approach (3. internat. ed.)*, Pearson Education, 2010.

[16] Kristinn R. Thórisson, 'Integrated a.i. systems', *Minds and Machines*, **17**(1), 11–25, (2007).

[17] Kristinn R. Thórisson, Hrvoje Benko, Denis Abramov, Andrew Arnold, Sameer Maskey, and Aruchunan Vaseekaran, 'Constructionist design methodology for interactive intelligences', *AI Magazine*, **25**(4), 77–90, (2004).

[18] Michael J. Wooldridge, *An Introduction to MultiAgent Systems (2. ed.)*, Wiley, 2009.

[19] Kai yuh Hsiao, Peter Gorniak, and Deb Roy, 'NetP: A network API for building heterogeneous modular intelligent systems', in *Proceedings of the AAAI 2005 Workshop on Modular Construction of Human-Like Intelligence*, (2005).

Towards Large-scale Inconsistency Measurement[1]

Matthias Thimm[2]

Abstract. We investigate the problem of inconsistency measurement on large knowledge bases by considering *stream-based inconsistency measurement*, i. e., we investigate inconsistency measures that cannot consider a knowledge base as a whole but process it within a stream. For that, we present, first, a novel inconsistency measure that is apt to be applied to the streaming case and, second, stream-based approximations for the new and some existing inconsistency measures. We conduct an extensive empirical analysis on the behavior of these inconsistency measures on large knowledge bases, in terms of runtime, accuracy, and scalability. We conclude that for two of these measures, the approximation of the new inconsistency measure and an approximation of the *contension* inconsistency measure, large-scale inconsistency measurement is feasible.

1 Introduction

Inconsistency measurement [2] is a subfield of Knowledge Representation and Reasoning (KR) that is concerned with the quantitative assessment of the severity of inconsistencies in knowledge bases. Consider the following two knowledge bases $\mathcal{K}_1$ and $\mathcal{K}_2$ formalized in propositional logic:

$$\mathcal{K}_1 = \{a, b \vee c, \neg a \wedge \neg b, d\} \qquad \mathcal{K}_2 = \{a, \neg a, b, \neg b\}$$

Both knowledge bases are classically inconsistent as for $\mathcal{K}_1$ we have $\{a, \neg a \wedge \neg b\} \models \perp$ and for $\mathcal{K}_2$ we have, e. g., $\{a, \neg a\} \models \perp$. These inconsistencies render the knowledge bases useless for reasoning if one wants to use classical reasoning techniques. In order to make the knowledge bases useful again, one can either use non-monotonic/paraconsistent reasoning techniques [11, 12] or one revises the knowledge bases appropriately to make them consistent [4]. Looking again at the knowledge bases $\mathcal{K}_1$ and $\mathcal{K}_2$ one can observe that the *severity* of their inconsistency is different. In $\mathcal{K}_1$, only two out of four formulas (a and $\neg a \wedge \neg b$) are *participating* in making $\mathcal{K}_1$ inconsistent while for $\mathcal{K}_2$ all formulas contribute to its inconsistency. Furthermore, for $\mathcal{K}_1$ only two propositions (a and b) participate in a conflict and using, e. g., paraconsistent reasoning one could still infer meaningful statements about c and d. For $\mathcal{K}_2$ no such statement can be made. This leads to the assessment that $\mathcal{K}_2$ should be regarded *more* inconsistent than $\mathcal{K}_1$. Inconsistency measures can be used to quantitatively assess the inconsistency of knowledge bases and to provide a guide for how to repair them, cf. [3]. Moreover, they can be used as an analytical tool to assess the quality of knowledge representation. For example, one simple inconsistency measure is to take the number of *minimal inconsistent subsets* (MIs) as an indicator for the inconsistency: the more MIs a knowledge base contains, the more inconsistent it is. For $\mathcal{K}_1$ we have then 1 as its inconsistency value and for $\mathcal{K}_2$ we have 2.

In this paper, we consider the computational problems of inconsistency measurement, particularly with respect to scalable inconsistency measurement on large knowledge bases, as they appear in, e. g., Semantic Web applications. To this end we present a novel inconsistency measure $\mathcal{I}_{hs}$ that approximates the η-inconsistency measure from [8] and is particularly apt to be applied to large knowledge bases. This measure is based on the notion of a *hitting set* which (in our context) is a minimal set of classical interpretations such that every formula of a knowledge base is satisfied by at least one element of the set. In order to investigate the problem of measuring inconsistency in large knowledge bases we also present a stream-based processing framework for inconsistency measurement. More precisely, the contributions of this paper are as follows:

1. We present a novel inconsistency measure $\mathcal{I}_{hs}$ based on hitting sets and show how this measure relates to other measures and, in particular, that it is a simplification of the η-inconsistency measure [8] (Section 3).
2. We formalize a theory of inconsistency measurement in streams and provide approximations of several inconsistency measures for the streaming case (Section 4).
3. We conduct an extensive empirical study on the behavior of those inconsistency measures in terms of runtime, accuracy, and scalability. In particular, we show that the stream variants of $\mathcal{I}_{hs}$ and of the *contension* measure $\mathcal{I}_c$ are effective and accurate for measuring inconsistency in the streaming setting and, therefore, in large knowledge bases (Section 5).

We give necessary preliminaries for propositional logic and inconsistency measurement in Section 2 and conclude the paper with a discussion in Section 6. Proofs of technical results can be found in Appendix A.

2 Preliminaries

Let At be a propositional signature, i. e., a (finite) set of propositions, and let $\mathcal{L}(\mathsf{At})$ be the corresponding propositional language, constructed using the usual connectives $\wedge$ (*and*), $\vee$ (*or*), and $\neg$ (*negation*). We use the symbol $\perp$ to denote contradiction. Then a knowledge base $\mathcal{K}$ is a finite set of formulas $\mathcal{K} \subseteq \mathcal{L}(\mathsf{At})$. Let $\mathbb{K}(\mathsf{At})$ be the set of all knowledge bases. We write $\mathbb{K}$ instead of $\mathbb{K}(\mathsf{At})$ when there is no ambiguity regarding the signature. Semantics to $\mathcal{L}(\mathsf{At})$ is given by *interpretations* $\omega : \mathsf{At} \to \{\mathsf{true}, \mathsf{false}\}$. Let $\mathsf{Int}(\mathsf{At})$ denote the set of all interpretations for At. An interpretation ω *satisfies* (or is a *model* of) an atom $a \in \mathsf{At}$, denoted by $\omega \models a$ (or $\omega \in \mathsf{Mod}(a)$), if and only if $\omega(a) = \mathsf{true}$. Both $\models$ and $\mathsf{Mod}(\cdot)$ are extended to arbitrary formulas, sets, and knowledge bases as usual.

Inconsistency measures are functions $\mathcal{I} : \mathbb{K} \to [0, \infty)$ that aim at assessing the severity of the inconsistency in a knowledge base $\mathcal{K}$,

[1] This paper has also been published in the Proceedings of the 37th German Conference on Artificial Intelligence (KI 2014)

[2] Institute for Web Science and Technologies, University of Koblenz-Landau, Germany, thimm@uni-koblenz.de

cf. [3]. The basic idea is that the larger the inconsistency in $\mathcal{K}$ the larger the value $\mathcal{I}(\mathcal{K})$. However, inconsistency is a concept that is not easily quantified and there have been a couple of proposals for inconsistency measures so far, see e. g. [8, 10, 1, 2, 5, 13]. There are two main paradigms for assessing inconsistency [5], the first being based on the (number of) formulas needed to produce inconsistencies and the second being based on the proportion of the language that is affected by the inconsistency. Below we recall some popular measures from both categories but we first introduce some necessary notations. Let $\mathcal{K} \in \mathbb{K}$ be some knowledge base.

Definition 1. A set $M \subseteq \mathcal{K}$ is called *minimal inconsistent subset* (MI) of $\mathcal{K}$ if $M \models \perp$ and there is no $M' \subset M$ with $M' \models \perp$. Let $\mathsf{MI}(\mathcal{K})$ be the set of all MIs of $\mathcal{K}$.

Definition 2. A formula $\alpha \in \mathcal{K}$ is called *free formula* of $\mathcal{K}$ if there is no $M \in \mathsf{MI}(\mathcal{K})$ with $\alpha \in M$. Let $\mathsf{Free}(\mathcal{K})$ denote the set of all free formulas of $\mathcal{K}$.

We adopt the following definition of a (basic) inconsistency measure from [3].

Definition 3. A *basic inconsistency measure* is a function $\mathcal{I} : \mathbb{K} \to [0, \infty)$ that satisfies the following three conditions:

1. $\mathcal{I}(\mathcal{K}) = 0$ if and only if $\mathcal{K}$ is consistent,
2. if $\mathcal{K} \subseteq \mathcal{K}'$ then $\mathcal{I}(\mathcal{K}) \leq \mathcal{I}(\mathcal{K}')$, and
3. for all $\alpha \in \mathsf{Free}(\mathcal{K})$ we have $\mathcal{I}(\mathcal{K}) = \mathcal{I}(\mathcal{K} \setminus \{\alpha\})$.

The first property (also called *consistency*) of a basic inconsistency measure ensures that all consistent knowledge bases receive a minimal inconsistency value and every inconsistent knowledge base receive a positive inconsistency value. The second property (also called *monotony*) states that the value of inconsistency can only increase when adding new information. The third property (also called *free formula independence*) states that removing harmless formulas from a knowledge base—i. e., formulas that do not contribute to the inconsistency—does not change the value of inconsistency. For the remainder of this paper we consider the following selection of inconsistency measures: the MI measure $\mathcal{I}_{\mathsf{MI}}$, the MI^c measure $\mathcal{I}_{\mathsf{MI}^c}$, the contension measure $\mathcal{I}_c$, and the η measure $\mathcal{I}_\eta$, which will be defined below, cf. [3, 8]. In order to define the contension measure $\mathcal{I}_c$ we need to consider three-valued interpretations for propositional logic [12]. A three-valued interpretation υ on At is a function $\upsilon : \mathsf{At} \to \{T, F, B\}$ where the values T and F correspond to the classical true and false, respectively. The additional truth value B stands for *both* and is meant to represent a conflicting truth value for a proposition. The function υ is extended to arbitrary formulas as shown in Table 1. Then, an interpretation υ satisfies a formula α, denoted by $\upsilon \models^3 \alpha$ if either $\upsilon(\alpha) = T$ or $\upsilon(\alpha) = B$.

For defining the η-inconsistency measure [8] we need to consider probability functions P of the form $P : \mathsf{Int}(\mathsf{At}) \to [0, 1]$ with $\sum_{\omega \in \mathsf{Int}(\mathsf{At})} P(\omega) = 1$. Let $\mathcal{P}(\mathsf{At})$ be the set of all those probability functions and for a given probability function $P \in \mathcal{P}(\mathsf{At})$ define the probability of an arbitrary formula α via $P(\alpha) = \sum_{\omega \models \alpha} P(\omega)$.

Definition 4. Let $\mathcal{I}_{\mathsf{MI}}, \mathcal{I}_{\mathsf{MI}^c}, \mathcal{I}_c$, and $\mathcal{I}_\eta$ be defined via

$$\mathcal{I}_{\mathsf{MI}}(\mathcal{K}) = |\mathsf{MI}(\mathcal{K})|,$$

$$\mathcal{I}_{\mathsf{MI}^c}(\mathcal{K}) = \sum_{M \in \mathsf{MI}(\mathcal{K})} \frac{1}{|M|},$$

$$\mathcal{I}_c(\mathcal{K}) = \min\{|\upsilon^{-1}(B)| \mid \upsilon \models^3 \mathcal{K}\},$$

$$\mathcal{I}_\eta(\mathcal{K}) = 1 - \max\{\xi \mid \exists P \in \mathcal{P}(\mathsf{At}) : \forall \alpha \in \mathcal{K} : P(\alpha) \geq \xi\}$$

The measure $\mathcal{I}_{\mathsf{MI}}$ takes the number of minimal inconsistent subsets of a knowledge base as an indicator for the amount of inconsistency: the more minimal inconsistent subsets the more severe the inconsistency. The measure $\mathcal{I}_{\mathsf{MI}^c}$ refines this idea by also taking the size of the minimal inconsistent subsets into account. Here the idea is that larger minimal inconsistent subsets indicate are less severe than smaller minimal inconsistent subsets (the less formulas are needed to produce an inconsistency the more “obvious” the inconsistency). The measure $\mathcal{I}_c$ considers the set of three-valued models of a knowledge base (which is always non-empty) and uses the minimal number of propositions with conflicting truth value as an indicator for inconsistency. Finally, the measure $\mathcal{I}_\eta$ (which always assigns an inconsistency value between 0 and 1) looks for the maximal probability one can assign to every formula of a knowledge base. All these measures are basic inconsistency measures as defined in Definition 3.

Example 1. For the knowledge bases $\mathcal{K}_1 = \{a, b \vee c, \neg a \wedge \neg b, d\}$ and $\mathcal{K}_2 = \{a, \neg a, b, \neg b\}$ from the introduction we obtain $\mathcal{I}_{\mathsf{MI}}(\mathcal{K}_1) = 1$, $\mathcal{I}_{\mathsf{MI}^c}(\mathcal{K}_1) = 0.5$, $\mathcal{I}_c(\mathcal{K}_1) = 2$, $\mathcal{I}_\eta(\mathcal{K}_1) = 0.5$, $\mathcal{I}_{\mathsf{MI}}(\mathcal{K}_2) = 2, \mathcal{I}_{\mathsf{MI}^c}(\mathcal{K}_2) = 1, \mathcal{I}_c(\mathcal{K}_2) = 2, \mathcal{I}_\eta(\mathcal{K}_2) = 0.5$.

For a more detailed introduction to inconsistency measures see e. g. [2, 3, 8] and for some recent developments see e. g. [1, 7].

As for computational complexity, the problem of computing an inconsistency value wrt. any of the above inconsistency measures is at least FNP-hard[3] as it contains a satisfiability problem as a sub problem.

3 An Inconsistency Measure based on Hitting Sets

The basic idea of our novel inconsistency measure $\mathcal{I}_{hs}$ is inspired by the measure $\mathcal{I}_\eta$ which seeks a probability function that maximizes the probability of all formulas of a knowledge base. Basically, the measure $\mathcal{I}_\eta$ looks for a minimal number of models of parts of the knowledge base and maximizes their probability in order to maximize the probability of the formulas. By just considering this basic idea we arrive at the notion of a *hitting set* for inconsistent knowledge bases.

Definition 5. A subset $H \subset \mathsf{Int}(\mathsf{At})$ is called a *hitting set* of $\mathcal{K}$ if for every $\alpha \in \mathcal{K}$ there is $\omega \in H$ with $\omega \models \alpha$. H is called a card-minimal hitting set if it is minimal wrt. cardinality. Let $h_\mathcal{K}$ be the cardinality of any card-minimal hitting set (define $h_\mathcal{K} = \infty$ if there does not exist a hitting set of $\mathcal{K}$).

Definition 6. The function $\mathcal{I}_{hs} : \mathbb{K} \to [0, \infty]$ is defined via $\mathcal{I}_{hs}(\mathcal{K}) = h_\mathcal{K} - 1$ for every $\mathcal{K} \in \mathbb{K}$.

Note, that if a knowledge base $\mathcal{K}$ contains a contradictory formula (e. g. $a \wedge \neg a$) we have $\mathcal{I}_{hs}(\mathcal{K}) = \infty$. In the following, we assume that $\mathcal{K}$ contains no such contradictory formulas.

Example 2. Consider the knowledge base $\mathcal{K}_3$ defined via

$$\mathcal{K}_3 = \{a \vee d, a \wedge b \wedge c, b, \neg b \vee \neg a, a \wedge b \wedge \neg c, a \wedge \neg b \wedge c\}$$

Then $\{\omega_1, \omega_2, \omega_3\} \subset \mathsf{Int}(\mathsf{At})$ with $\omega_1(a) = \omega_1(b) = \omega_1(c) = \mathsf{true}$, $\omega_2(a) = \omega_2(c) = \mathsf{true}$, $\omega_1(b) = \mathsf{false}$, and $\omega_3(a) = \omega_3(b) = \mathsf{true}$, $\omega_3(c) = \mathsf{false}$ is a card-minimal hitting set for $\mathcal{K}_3$ and therefore $\mathcal{I}_{hs}(\mathcal{K}_3) = 2$. Note that for the knowledge bases $\mathcal{K}_1$ and $\mathcal{K}_2$ from Example 1 we have $\mathcal{I}_{hs}(\mathcal{K}_1) = \mathcal{I}_{hs}(\mathcal{K}_2) = 1$.

[3] FNP is the generalization of the class NP to functional problems.

Table 1 Truth tables for propositional three-valued logic [12].

α	β	$\alpha \wedge \beta$	$\alpha \vee \beta$	$\neg\alpha$	α	β	$\alpha \wedge \beta$	$\alpha \vee \beta$	$\neg\alpha$	α	β	$\alpha \wedge \beta$	$\alpha \vee \beta$	$\neg\alpha$
T	T	T	T	F	B	T	B	T	B	F	T	F	T	T
T	B	B	T	F	B	B	B	B	B	F	B	F	B	T
T	F	F	T	F	B	F	F	B	B	F	F	F	F	T

Proposition 1. *The function $\mathcal{I}_{hs}$ is a (basic) inconsistency measure.*

The result below shows that $\mathcal{I}_{hs}$ also behaves well with some more properties mentioned in the literature [5, 13]. For that, we denote with $\mathsf{At}(F)$ for a formula or a set of formulas F the set of propositions appearing in F. Furthermore, two knowledge bases $\mathcal{K}_1, \mathcal{K}_2$ are *semi-extensionally equivalent* ($\mathcal{K}_1 \equiv^\sigma \mathcal{K}_2$) if there is a bijection $\sigma : \mathcal{K}_1 \to \mathcal{K}_2$ such that for all $\alpha \in \mathcal{K}_1$ we have $\alpha \equiv \sigma(\alpha)$.

Proposition 2. *The measure $\mathcal{I}_{hs}$ satisfies the following properties:*

- *If $\alpha \in \mathcal{K}$ is such that $\mathsf{At}(\alpha) \cap \mathsf{At}(\mathcal{K} \setminus \{\alpha\}) = \emptyset$ then $\mathcal{I}_{hs}(\mathcal{K}) = \mathcal{I}_{hs}(\mathcal{K} \setminus \{\alpha\})$ (*safe formula independence*).*
- *If $\mathcal{K} \equiv^\sigma \mathcal{K}'$ then $\mathcal{I}_{hs}(\mathcal{K}) = \mathcal{I}_{hs}(\mathcal{K}')$ (*irrelevance of syntax*).*
- *If $\alpha \models \beta$ and $\alpha \not\models \perp$ then $\mathcal{I}_{hs}(\mathcal{K} \cup \{\alpha\}) \geq \mathcal{I}_{hs}(\mathcal{K} \cup \{\beta\})$ (*dominance*).*

The measure $\mathcal{I}_{hs}$ can also be nicely characterized by a consistent *partitioning* of a knowledge base.

Definition 7. A set $\Phi = \{\Phi_1, \ldots, \Phi_n\}$ with $\Phi_1 \cup \ldots \cup \Phi_n = \mathcal{K}$ and $\Phi_i \cap \Phi_j = \emptyset$ for $i, j = 1, \ldots, n$, $i \neq j$, is called a *partitioning* of $\mathcal{K}$. A partitioning $\Phi = \{\Phi_1, \ldots, \Phi_n\}$ is consistent if $\Phi_i \not\models \perp$ for $i = 1, \ldots, n$. A consistent partitioning Φ is called card-minimal if it is minimal wrt. cardinality among all consistent partitionings of $\mathcal{K}$.

Proposition 3. *A consistent partitioning Φ is a card-minimal partitioning of $\mathcal{K}$ if and only if $\mathcal{I}_{hs}(\mathcal{K}) = |\Phi| - 1$.*

As $\mathcal{I}_{hs}$ is inspired by $\mathcal{I}_\eta$ we go on by comparing these two measures.

Proposition 4. *Let $\mathcal{K}$ be a knowledge base. If $\infty > \mathcal{I}_{hs}(\mathcal{K}) > 0$ then*

$$1 - \frac{1}{\mathcal{I}_{hs}(\mathcal{K})} < \mathcal{I}_\eta(\mathcal{K}) \leq 1 - \frac{1}{\mathcal{I}_{hs}(\mathcal{K}) + 1}$$

Note that for $\mathcal{I}_{hs}(\mathcal{K}) = 0$ we always have $\mathcal{I}_\eta(\mathcal{K}) = 0$ as well, as both are basic inconsistency measures.

Corollary 1. *If $\mathcal{I}_\eta(\mathcal{K}_1) \leq \mathcal{I}_\eta(\mathcal{K}_2)$ then $\mathcal{I}_{hs}(\mathcal{K}_1) \leq \mathcal{I}_{hs}(\mathcal{K}_2)$.*

However, the measures $\mathcal{I}_\eta$ and $\mathcal{I}_{hs}$ are not equivalent as the following example shows.

Example 3. Consider the knowledge bases $\mathcal{K}_1 = \{a, \neg a\}$ and $\mathcal{K}_2 = \{a, b, \neg a \vee \neg b\}$. Then we have $\mathcal{I}_{hs}(\mathcal{K}_1) = \mathcal{I}_{hs}(\mathcal{K}_2) = 1$ but $\mathcal{I}_\eta(\mathcal{K}_1) = 0.5 > 1/3 = \mathcal{I}_\eta(\mathcal{K}_2)$.

It follows that the order among knowledge bases induced by $\mathcal{I}_\eta$ is a refinement of the order induced by $\mathcal{I}_{hs}$. However, $\mathcal{I}_{hs}$ is better suited for approximation in large knowledge bases than $\mathcal{I}_\eta$, cf. the following section.

The idea underlying $\mathcal{I}_{hs}$ is also similar to the contension inconsistency measure $\mathcal{I}_c$. However, these measures are not equivalent as the following example shows.

Example 4. Consider the knowledge bases $\mathcal{K}_1$ and $\mathcal{K}_2$ given as

$$\mathcal{K}_1 = \{a \wedge b \wedge c, \neg a \wedge \neg b \wedge \neg c\} \quad \mathcal{K}_2 = \{a \wedge b, \neg a \wedge b, a \wedge \neg b\}$$

Then we have $\mathcal{I}_{hs}(\mathcal{K}_1) = 2 < 3 = \mathcal{I}_{hs}(\mathcal{K}_2)$ but $\mathcal{I}_c(\mathcal{K}_1) = 3 > 2 = \mathcal{I}_c(\mathcal{K}_2)$.

4 Inconsistency Measurement in Streams

In the following, we discuss the problem of inconsistency measurement in large knowledge bases. We address this issue by using a stream-based approach of accessing the formulas of a large knowledge base. Formulas of a knowledge base then need to be processed one by one by a stream-based inconsistency measure. The goal of this formalization is to obtain stream-based inconsistency measures that approximate given inconsistency measures when the latter would have been applied to the knowledge base as a whole. We first formalize this setting and, afterwards, provide concrete approaches for some inconsistency measures.

4.1 Problem Formalization

We use a very simple formalization of a stream that is sufficient for our needs.

Definition 8. A *propositional stream* $\mathcal{S}$ is a function $\mathcal{S} : \mathbb{N} \to \mathcal{L}(\mathsf{At})$. Let $\mathbb{S}$ be the set of all propositional streams.

A propositional stream models a sequence of propositional formulas. On a wider scope, a propositional stream can also be interpreted as a very general abstraction of the output of a linked open data crawler (such as LDSpider [6]) that crawls knowledge formalized as RDF (*Resource Description Framework*) from the web, enriched, e. g. with OWL semantics. We model large knowledge bases by propositional streams that indefinitely repeat the formulas of the knowledge base. For that, we assume for a knowledge base $\mathcal{K} = \{\phi_1, \ldots, \phi_n\}$ the existence of a *canonical enumeration* $\mathcal{K}^c = \langle \phi_1, \ldots, \phi_n \rangle$ of the elements of $\mathcal{K}$. This enumeration can be arbitrary and has no specific meaning other than to enumerate the elements in an unambiguous way.

Definition 9. Let $\mathcal{K}$ be a knowledge base and $\mathcal{K}^c = \langle \phi_1, \ldots, \phi_n \rangle$ its canonical enumeration. The *$\mathcal{K}$-stream* $\mathcal{S}_\mathcal{K}$ is defined as $\mathcal{S}_\mathcal{K}(i) = \phi_{(i \bmod n)+1}$ for all $i \in \mathbb{N}$.

Given a $\mathcal{K}$-stream $\mathcal{S}_\mathcal{K}$ and an inconsistency measure $\mathcal{I}$ we aim at defining a method that processes the elements of $\mathcal{S}_\mathcal{K}$ one by one and approximates $\mathcal{I}(\mathcal{K})$.

Definition 10. A *stream-based inconsistency measure* $\mathcal{J}$ is a function $\mathcal{J} : \mathbb{S} \times \mathbb{N} \to [0, \infty)$.

Definition 11. Let $\mathcal{I}$ be an inconsistency measure and $\mathcal{J}$ a stream-based inconsistency measure. Then $\mathcal{J}$ *approximates* (or *is an approximation of*) $\mathcal{I}$ if for all $\mathcal{K} \in \mathbb{K}$ we have $\lim_{i \to \infty} \mathcal{J}(\mathcal{S}_\mathcal{K}, i) = \mathcal{I}(\mathcal{K})$.

4.2 A Naive Window-based Approach

The simplest form of implementing a stream-based variant of any algorithm or function is to use a window-based approach, i. e., to consider at any time point a specific excerpt from the stream and apply the original algorithm or function on this excerpt. For any propositional stream $\mathcal{S}$ let $\mathcal{S}^{i,j}$ (for $i \leq j$) be the knowledge base obtained by taking the formulas from $\mathcal{S}$ between positions i and j, i. e., $\mathcal{S}^{i,j} = \{\mathcal{S}(i), \ldots, \mathcal{S}(j)\}$.

Definition 12. Let $\mathcal{I}$ be an inconsistency measure, $w \in \mathbb{N} \cup \{\infty\}$, and g some function $g : [0, \infty) \times [0, \infty) \to [0, \infty)$ with $g(x, y) \in [\min\{x, y\}, \max\{x, y\}]$. We define the *naive window-based measure* $\mathcal{J}_{\mathcal{I}}^{w,g} : \mathbb{S} \times \mathbb{N} \to [0, \infty)$ via

$$\mathcal{J}_{\mathcal{I}}^{w,g}(\mathcal{S}, i) = \begin{cases} 0 & \text{if } i = 0 \\ g(\mathcal{I}(\mathcal{S}^{\max\{0,i-w\},i}), \mathcal{J}_{\mathcal{I}}^{w,g}(\mathcal{S}, i-1)) & \text{otherwise} \end{cases}$$

for every $\mathcal{S}$ and $i \in \mathbb{N}$.

The function g in the above definition is supposed to be an aggregation function that combines the new obtained inconsistency value $\mathcal{I}(\mathcal{S}_{\mathcal{K}}^{\max\{0,i-w\},i})$ with the previous value $\mathcal{J}_{\mathcal{I}}^{w,g}(\mathcal{S}, i-1)$. This function can be ,e. g., the maximum function max or a smoothing function $g_\alpha(x, y) = \alpha x + (1 - \alpha)y$ for some $\alpha \in [0, 1]$ (for every $x, y \in [0, \infty)$).

Proposition 5. *Let $\mathcal{I}$ be an inconsistency measure, $w \in \mathbb{N} \cup \{\infty\}$, and g some function $g : [0, \infty) \times [0, \infty) \to [0, \infty)$ with $g(x, y) \in [\min\{x, y\}, \max\{x, y\}]$.*

1. *If w is finite then $\mathcal{J}_{\mathcal{I}}^{w,g}$ is* not *an approximation of $\mathcal{I}$.*
2. *If $w = \infty$ and $g(x, y) > \min\{x, y\}$ if $x \neq y$ then $\mathcal{J}_{\mathcal{I}}^{w,g}$ is an approximation of $\mathcal{I}$.*
3. *$\mathcal{J}_{\mathcal{I}}^{w,g}(\mathcal{S}_{\mathcal{K}}, i) \leq \mathcal{I}(\mathcal{K})$ for every $\mathcal{K} \in \mathbb{K}$ and $i \in \mathbb{N}$.*

4.3 Approximation Algorithms for $\mathcal{I}_{hs}$ and $\mathcal{I}_c$

The approximation algorithms for $\mathcal{I}_{hs}$ and $\mathcal{I}_c$ that are presented in this subsection are using concepts of the programming paradigms of *simulated annealing* and *genetic programming* [9]. Both algorithms follow the same idea and we will only formalize the one for $\mathcal{I}_{hs}$ and give some hints on how to adapt it for $\mathcal{I}_c$.

The basic idea for the stream-based approximation of $\mathcal{I}_{hs}$ is as follows. At any processing step we maintain a candidate set $C \in 2^{\mathsf{Int(At)}}$ (initialized with the empty set) that approximates a hitting set of the underlying knowledge base. At the beginning of a processing step we make a random choice (with decreasing probability the more formulas we already encountered) whether to remove some element of C. This action ensures that C does not contain superfluous elements. Afterwards we check whether there is still an interpretation in C that satisfies the currently encountered formula. If this is not the case we add some random model of the formula to C. Finally, we update the previously computed inconsistency value with $|C| - 1$, taking also some aggregation function g (as for the naive window-based approach) into account. In order to increase the probability of successfully finding a minimal hitting set we do not maintain a single candidate set C but a (multi-)set $Cand = \{C_1, \ldots, C_m\}$ for some previously specified parameter $m \in \mathbb{N}$ and use the average size of these candidate hitting sets.

Definition 13. Let $m \in \mathbb{N}$, g some function $g : [0, \infty) \times [0, \infty) \to [0, \infty)$ with $g(x, y) \in [\min\{x, y\}, \max\{x, y\}]$, and $f : \mathbb{N} \to [0, 1]$ some monotonically decreasing function with $\lim_{n\to\infty} f(n) = 0$. We define $\mathcal{J}_{hs}^{m,g,f}$ via

$$\mathcal{J}_{hs}^{m,g,f}(\mathcal{S}, i) = \begin{cases} 0 & \text{if } i = 0 \\ \mathtt{update}_{hs}^{m,g,f}(\mathcal{S}(i)) & \text{otherwise} \end{cases}$$

for every $\mathcal{S}$ and $i \in \mathbb{N}$. The function $\mathtt{update}_{hs}^{m,g,f}$ is depicted in Algorithm 1.

At the first call of the algorithm $\mathtt{update}_{hs}^{m,g,f}$ the value of $currentValue$ (which contains the currently estimated inconsistency value) is initialized to 0 and the (mulit-)set $Cand \subseteq 2^{\mathsf{Int(At)}}$ (which contains a population of candidate hitting sets) is initialized with m empty sets. The function f can be any monotonically decreasing function with $\lim_{n\to\infty} f(n) = 0$ (this ensures that at any candidate C reaches some stable result). The parameter m increases the probability that at least one of the candidate hitting sets attains the global optimum of a card-minimal hitting set.

Algorithm 1 $\mathtt{update}_{hs}^{m,g,f}(form)$

```
1:  Initialize $currentValue$ and $Cand$
2:  $N = N + 1$
3:  $newValue = 0$
4:  for all $C \in Cand$ do
5:      $rand \in [0, 1]$
6:      if $rand < f(N)$ then
7:          Remove some random $\omega$ from $C$
8:      if $\neg\exists\omega \in C : \omega \models form$ then
9:          Add random $\omega \in \mathsf{Mod}(form)$ to $C$
10:     $newValue = newValue + (|C| - 1)/|Cand|$
11: $currentValue = g(newValue, currentValue)$
12: return $currentValue$
```

As $\mathcal{J}_{hs}^{m,g,f}$ is a random process we cannot show that $\mathcal{J}_{hs}^{m,g,f}$ is an approximation of $\mathcal{I}_{hs}$ in the general case. However, we can give the following result.

Proposition 6. *For every probability $p \in [0, 1)$, g some function $g : [0, \infty) \times [0, \infty) \to [0, \infty)$ with $g(x, y) \in [\min\{x, y\}, \max\{x, y\}]$ and $g(x, y) > \min\{x, y\}$ if $x \neq y$, a monotonically decreasing function $f : \mathbb{N} \to [0, 1]$ with $\lim_{n\to\infty} f(n) = 0$, and $\mathcal{K} \in \mathbb{K}$ there is $m \in \mathbb{N}$ such that with probability greater or equal p it is the case that*

$$\lim_{i\to\infty} \mathcal{J}_{hs}^{m,g,f}(\mathcal{S}_{\mathcal{K}}, i) = \mathcal{I}_{hs}(\mathcal{K})$$

This result states that $\mathcal{J}_{hs}^{m,g,f}$ indeed approximates $\mathcal{I}_{hs}$ if we choose the number of populations large enough. In the next section we will provide some empirical evidence that even for small values of m results are satisfactory.

Both Definition 13 and Algorithm 1 can be modified slightly in order to approximate $\mathcal{I}_c$ instead of $\mathcal{I}_{hs}$, yielding a new measure $\mathcal{J}_c^{m,g,f}$. For that, the set of candidates $Cand$ contains three-valued interpretations instead of sets of classical interpretations. In line 7, we do not remove an interpretation from C but flip some arbitrary proposition from B to T or F. Similarly, in line 9 we do not add an interpretation but flip some propositions to B in order to satisfy the new formula. Finally, the inconsistency value is determined by taking the number of B-valued propositions. For more details see the implementations of both $\mathcal{J}_{hs}^{m,g,f}$ and $\mathcal{J}_c^{m,g,f}$, which will also be discussed in the next section.

5 Empirical Evaluation

In this section we describe our empirical experiments on runtime, accuracy, and scalability of some stream-based inconsistency measures. Our Java implementations[4] have been added to the *Tweety Libraries for Knowledge Representation* [14].

[4] $\mathcal{I}_{\mathsf{MI}}, \mathcal{I}_{\mathsf{MI}^c}, \mathcal{I}_\eta, \mathcal{J}_{\mathcal{I}}^{w,g}$: `http://mthimm.de/r?r=tweety-inc-commons`
$\mathcal{I}_c, \mathcal{I}_{hs}$: `http://mthimm.de/r?r=tweety-inc-pl`
$\mathcal{J}_{hs}^{m,g,f}$: `http://mthimm.de/r?r=tweety-stream-hs`
$\mathcal{J}_c^{m,g,f}$: `http://mthimm.de/r?r=tweety-stream-c`
Evaluation framework: `http://mthimm.de/r?r=tweety-stream-eval`

Table 2 Runtimes for the evaluated measures; each value is averaged over 100 random knowledge bases of 5000 formulas; the total runtime is after 40000 iterations

Measure	RT (iteration)	RT (total)	Measure	RT (iteration)	RT (total)
$\mathcal{J}^{500,\max}_{\mathcal{I}_{\mathsf{MI}}}$	198ms	133m	$\mathcal{J}^{10,g_{0.75},f_1}_{c}$	0.16ms	6.406s
$\mathcal{J}^{1000,\max}_{\mathcal{I}_{\mathsf{MI}}}$	359ms	240m	$\mathcal{J}^{100,g_{0.75},f_1}_{c}$	1.1ms	43.632s
$\mathcal{J}^{2000,\max}_{\mathcal{I}_{\mathsf{MI}}}$	14703ms	9812m	$\mathcal{J}^{500,g_{0.75},f_1}_{c}$	5.21ms	208.422s
$\mathcal{J}^{500,\max}_{\mathcal{I}_{\mathsf{MI}^c}}$	198ms	134m	$\mathcal{J}^{10,g_{0.75},f_1}_{hs}$	0.07ms	2.788s
$\mathcal{J}^{1000,\max}_{\mathcal{I}_{\mathsf{MI}^c}}$	361ms	241m	$\mathcal{J}^{100,g_{0.75},f_1}_{hs}$	0.24ms	9.679s
$\mathcal{J}^{2000,\max}_{\mathcal{I}_{\mathsf{MI}^c}}$	14812ms	9874m	$\mathcal{J}^{500,g_{0.75},f_1}_{hs}$	1.02ms	40.614s

5.1 Evaluated Approaches

For our evaluation, we considered the inconsistency measures $\mathcal{I}_{\mathsf{MI}}$, $\mathcal{I}_{\mathsf{MI}^c}$, $\mathcal{I}_\eta$, $\mathcal{I}_c$, and $\mathcal{I}_{hs}$. We used the SAT solver *lingeling*[5] for the sub-problems of determining consistency and to compute a model of a formula. For enumerating the set of MIs of a knowledge base (as required by $\mathcal{I}_{\mathsf{MI}}$ and $\mathcal{I}_{\mathsf{MI}^c}$) we used MARCO[6]. The measure $\mathcal{I}_\eta$ was implemented using the linear optimization solver *lp_solve*[7]. The measures $\mathcal{I}_{\mathsf{MI}}$, $\mathcal{I}_{\mathsf{MI}^c}$, and $\mathcal{I}_\eta$ were used to define three different versions of the naive window-based measure $\mathcal{J}^{w,g}_{\mathcal{I}}$ (with $w = 500, 1000, 2000$ and $g = \max$). For the measures $\mathcal{I}_c$ and $\mathcal{I}_{hs}$ we tested each three versions of their streaming variants $\mathcal{J}^{m,g_{0.75},f_1}_{c}$ and $\mathcal{J}^{m,g_{0.75},f_1}_{hs}$ (with $m = 10, 100, 500$) with $f_1 : \mathbb{N} \to [0,1]$ defined via $f_1(i) = 1/(i+1)$ for all $i \in \mathbb{N}$ and $g_{0.75}$ is the smoothing function for $\alpha = 0.75$ as defined in the previous section.

5.2 Experiment Setup

For measuring the runtime of the different approaches we generated 100 random knowledge bases in CNF (*Conjunctive Normal Form*) with each 5000 formulas (=disjunctions) and 30 propositions. For each generated knowledge base $\mathcal{K}$ we considered its $\mathcal{K}$-stream and processing of the stream was aborted after 40000 iterations. We fed the $\mathcal{K}$-stream to each of the evaluated stream-based inconsistency measures and measured the average runtime per iteration and the total runtime. For each iteration, we set a time-out of 2 minutes and aborted processing of the stream completely if a time-out occurred.

In order to measure accuracy, for each of the considered approaches we generated another 100 random knowledge bases with specifically set inconsistency values[8], used otherwise the same settings as above, and measured the returned inconsistency values.

To evaluate the scalability of our stream-based approach of $\mathcal{I}_{hs}$ we conducted a third experiment[9] where we fixed the number of propositions (60) and the specifically set inconsistency value (200) and varied the size of the knowledge bases from 5000 to 50000 (with steps of 5000 formulas). We measured the total runtime up to the point when the inconsistency value was within a tolerance of ± 1 of the expected inconsistency value.

The experiments were conducted on a server with two Intel Xeon X5550 QuadCore (2.67 GHz) processors with 8 GB RAM running SUSE Linux 2.6.

5.3 Results

Our first observation concerns the inconsistency measure $\mathcal{I}_\eta$ which proved to be not suitable to work on large knowledge bases[10]. Computing the value $\mathcal{I}_\eta(\mathcal{K})$ for some knowledge base $\mathcal{K}$ includes solving a linear optimization problem over a number of variables which is (in the worst-case) exponential in the number of propositions of the signature. In our setting with $|\mathsf{At}| = 30$ the generated optimization problem contained therefore $2^{30} = 1073741824$ variables. Hence, even the optimization problem itself could not be constructed within the timeout of 2 minutes for every step. As we are not aware of any more efficient implementation of $\mathcal{I}_\eta$, we will not report on further results for $\mathcal{I}_\eta$ in the following.

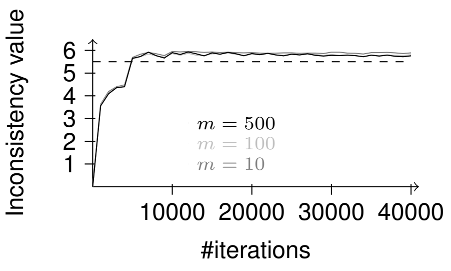
As for the runtime of the naive window-based approaches of $\mathcal{I}_{\mathsf{MI}}$ and $\mathcal{I}_{\mathsf{MI}^c}$ and our stream-based approaches for $\mathcal{I}_c$ and $\mathcal{I}_{hs}$ see Table 2. There one can see that $\mathcal{J}^{w,g}_{\mathcal{I}_{\mathsf{MI}}}$ and $\mathcal{J}^{w,g}_{\mathcal{I}_{\mathsf{MI}^c}}$ on the one hand, and $\mathcal{J}^{m,g,f}_{c}$ and $\mathcal{J}^{m,g,f}_{hs}$ on the other hand, have comparable runtimes, respectively. The former two have almost identical runtimes, which is obvious as the determination of the MIs is the main problem in both their computations. Clearly, $\mathcal{J}^{m,g,f}_{c}$ and $\mathcal{J}^{m,g,f}_{hs}$ are significantly faster per iteration (and in total) than $\mathcal{J}^{w,g}_{\mathcal{I}_{\mathsf{MI}}}$ and $\mathcal{J}^{w,g}_{\mathcal{I}_{\mathsf{MI}^c}}$, only very few milliseconds for the latter and several hundreds and thousands of milliseconds for the former (for all variants of m and w). The impact of increasing w for $\mathcal{J}^{m,g,f}_{c}$ and $\mathcal{J}^{m,g,f}_{hs}$ is expectedly linear while the impact of increasing the window size w for $\mathcal{J}^{w,g}_{\mathcal{I}_{\mathsf{MI}}}$ and $\mathcal{J}^{w,g}_{\mathcal{I}_{\mathsf{MI}^c}}$ is exponential (this is also clear as both solve an FNP-hard problem).

As for the accuracy of the different approaches see Figure 1 (a)–(d). There one can see that both $\mathcal{J}^{m,g,f}_{hs}$ and $\mathcal{J}^{m,g,f}_{c}$ (Figures 1a and 1b) converge quite quickly (almost right after the knowledge base has been processed once) into a $[-1, 1]$ interval around the actual inconsistency value, where $\mathcal{J}^{m,g,f}_{c}$ is even closer to it. The naive window-based approaches (Figures 1c and 1d) have a comparable bad performance (this is clear as those approaches cannot *see* all MIs at any iteration due to the limited window size). Surprisingly, the impact of larger values of m for $\mathcal{J}^{m,g,f}_{hs}$ and $\mathcal{J}^{m,g,f}_{c}$ is rather small in terms of accuracy which suggests that the random process of our algorithm is quite robust. Even for $m = 10$ the results are quite satisfactory.

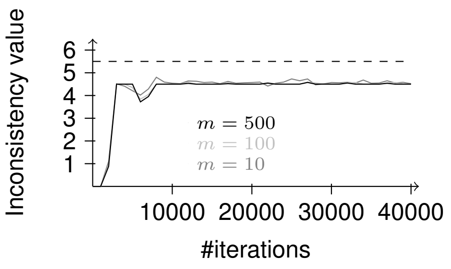
As for the scalability of $\mathcal{J}^{m,g_{0.75},f_1}_{hs}$ see Figure 1e. There one can observe a linear increase in the runtime of all variants wrt. the size of the knowledge base. Furthermore, the difference between the variants is also linearly in the parameter m (which is also clear as each population is an independent random process). It is noteworthy, that

[5] `http://fmv.jku.at/lingeling/`
[6] `http://sun.iwu.edu/~mliffito/marco/`
[7] `http://lpsolve.sourceforge.net`
[8] The sampling algorithms can be found at `http://mthimm.de/r?r=tweety-sampler`
[9] We did the same experiment with our stream-based approach of $\mathcal{I}_c$ but do not report the results due to the similarity to $\mathcal{I}_{hs}$ and space restrictions.

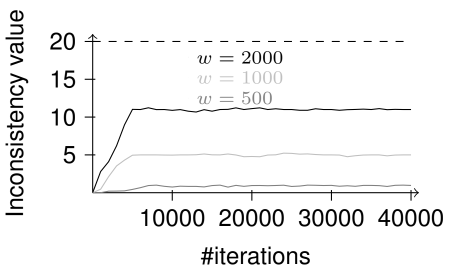
[10] More precisely, our implementation of the measure proved to be not suitable for this setting



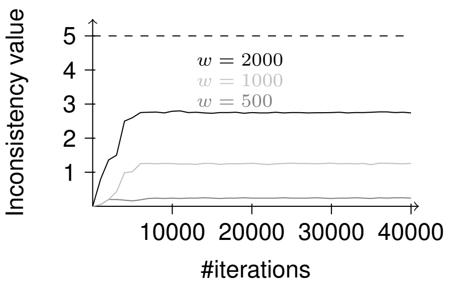
(a) Accuracy $\mathcal{J}_{hs}^{m,g_{0.75},f_1}$



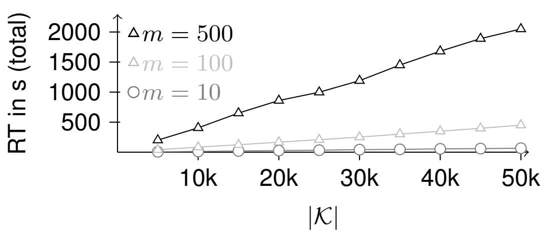
(b) Accuracy $\mathcal{J}_{c}^{m,g_{0.75},f_1}$



(c) Accuracy $\mathcal{J}_{\mathcal{I}_{\mathsf{MI}}}^{w,\max}$



(d) Accuracy $\mathcal{J}_{\mathcal{I}^c_{\mathsf{MI}}}^{w,\max}$



(e) Scalability $\mathcal{J}_{hs}^{m,g_{0.75},f_1}$

Figure 1: (a)–(d): Accuracy performance for the evaluated measures (dashed line is actual inconsistency value); each value is averaged over 100 random knowledge bases of 5000 formulas (30 propositions) with varying inconsistency values; (e): Evaluation of the scalability of $\mathcal{J}_{hs}^{m,g_{0.75},f_1}$; each value is averaged over 10 random knowledge bases of the given size

the average runtime for $\mathcal{J}_{hs}^{10,g_{0.75},f_1}$ is about 66.1 seconds for knowledge bases with 50000 formulas. As the significance of the parameter m for the accuracy is also only marginal, the measure $\mathcal{J}_{hs}^{10,g_{0.75},f_1}$ is clearly an effective and accurate stream-based inconsistency measure.

6 Discussion and Conclusion

In this paper we discussed the issue of large-scale inconsistency measurement and proposed novel approximation algorithms that are effective for the streaming case. To the best of our knowledge, the computational issues for measuring inconsistency, in particular with respect to scalability problems, have not yet been addressed in the literature before. One exception is the work by Ma and colleagues [10] who present an anytime algorithm that approximates an inconsistency measure based on a 4-valued paraconsistent logic (similar to the contension inconsistency measure). The algorithm provides lower and upper bounds for this measure and can be stopped at any point in time with some guaranteed quality. The main difference between our framework and the algorithm of [10] is that the latter needs to process the whole knowledge base in each atomic step and is therefore not directly applicable for the streaming scenario. The empirical evaluation [10] also suggests that our streaming variant of $\mathcal{I}_{hs}$ is much more performant as Ma et al. report an average runtime of their algorithm of about 240 seconds on a knowledge base with 120 formulas and 20 propositions (no evaluation on larger knowledge bases is given) while our measure has a runtime of only a few seconds for knowledge bases with 5000 formulas with comparable accuracy[11]. A deeper comparison of these different approaches is planned for future work.

Our work showed that inconsistency measurement is not only a theoretical field but can actually be applied to problems of reasonable size. In particular, our stream-based approaches of $\mathcal{I}_{hs}$ and $\mathcal{I}_c$ are accurate and effective for measuring inconsistencies in large knowledge bases. Current and future work is about the application of our work on linked open data sets [6].

REFERENCES

[1] J. Grant and A. Hunter, ‘Distance-based Measures of Inconsistency’, in *Proceedings of the 12th Europen Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU’13)*, pp. 230–241, (2013).

[2] John Grant and Anthony Hunter, ‘Measuring inconsistency in knowledgebases’, *Journal of Intelligent Information Systems*, **27**, 159–184, (2006).

[3] John Grant and Anthony Hunter, ‘Measuring consistency gain and information loss in stepwise inconsistency resolution’, in *Proc. of the 11th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU 2011)*, pp. 362–373, (2011).

[4] S. O. Hansson, *A Textbook of Belief Dynamics*, Kluwer Academic Publishers, 2001.

[5] Anthony Hunter and Sebastien Konieczny, ‘On the measure of conflicts: Shapley inconsistency values’, *Artificial Intelligence*, **174**(14), 1007–1026, (July 2010).

[6] Robert Isele, Jürgen Umbrich, Chris Bizer, and Andreas Harth, ‘LDSpider: An open-source crawling framework for the web of linked data’, in *Proceedings of 9th International Semantic Web Conference (ISWC 2010) Posters and Demos*, (2010).

[7] Said Jabbour, Yue Ma, and Badran Raddaoui, ‘Inconsistency measurement thanks to mus decomposition’, in *Proc. of the 13th Int. Conference on Autonomous Agents and Multiagent Systems*, (2014).

[11] Although hardware specifications for these experiments are different this huge difference is significant.

[8] Kevin M. Knight, *A Theory of Inconsistency*, Ph.D. dissertation, University Of Manchester, 2002.

[9] D. Lawrence, *Genetic Algorithms and Simulated Annealing*, Pitman Publishing, 1987.

[10] Yue Ma, Guilin Qi, Guohui Xiao, Pascal Hitzler, and Zuoquan Lin, 'An anytime algorithm for computing inconsistency measurement', in *Knowledge Science, Engineering and Management*, 29–40, Springer, (2009).

[11] D. Makinson, *Bridges from Classical to Nonmonotonic Logic*, College Publications, 2005.

[12] G. Priest, 'Logic of Paradox', *Journal of Philosophical Logic*, **8**, 219–241, (1979).

[13] Matthias Thimm, 'Inconsistency measures for probabilistic logics', *Artificial Intelligence*, **197**, 1–24, (April 2013).

[14] Matthias Thimm, 'Tweety - A Comprehensive Collection of Java Libraries for Logical Aspects of Artificial Intelligence and Knowledge Representation', in *Proceedings of the 14th Int. Conference on Principles of Knowledge Representation and Reasoning (KR'14)*, (2014).

A Proofs of technical results

Proposition 1. *The function $\mathcal{I}_{hs}$ is a (basic) inconsistency measure.*

Proof. We have to show that properties 1.), 2.), and 3.) of Definition 3 are satisfied.

1. If $\mathcal{K}$ is consistent there is a $\omega \in \mathsf{Int}(\mathsf{At})$ such that $\omega \models \alpha$ for every $\alpha \in \mathcal{K}$. Therefore, $H = \{\omega\}$ is a card minimal hitting set and we have $h_{\mathcal{K}} = 1$ and therefore $\mathcal{I}_{hs}(\mathcal{K}) = 0$. Note that for inconsistent $\mathcal{K}$ we always have $h_{\mathcal{K}} > 1$.
2. Let $\mathcal{K} \subseteq \mathcal{K}'$ and let H be a card-minimal hitting set of $\mathcal{K}'$. Then H is also a hitting set of $\mathcal{K}$ (not necessarily a card-minimal one). Therefore, we have $h_{\mathcal{K}} \leq h_{\mathcal{K}'}$ and $\mathcal{I}_{hs}(\mathcal{K}) \leq \mathcal{I}_{hs}(\mathcal{K}')$.
3. Let $\alpha \in \mathsf{Free}(\mathcal{K})$ and define $\mathcal{K}' = \mathcal{K} \setminus \{\alpha\}$. Let H be a card-minimal hitting set of $\mathcal{K}'$ and let $\omega \in H$. Furthermore, let $\mathcal{K}'' \subseteq \mathcal{K}'$ be the set of all formulas such that $\omega \models \beta$ for all $\beta \in \mathcal{K}''$. It follows that $\mathcal{K}''$ is consistent. As α is a free formula it follows that $\mathcal{K}'' \cup \{\alpha\}$ is also consistent (otherwise there would be a minimal inconsistent subset of $\mathcal{K}''$ containing α). Let ω' be a model of $\mathcal{K}'' \cup \{\alpha\}$. Then $H' = (H \setminus \{\omega\}) \cup \{\omega'\}$ is a hitting set of $\mathcal{K}$ and due to 2.) also card-minimal. Hence, we have $h_{\mathcal{K}'} = h_{\mathcal{K}}$ and $\mathcal{I}_{hs}(\mathcal{K}') = \mathcal{I}_{hs}(\mathcal{K})$.

□

Proposition 2. *The measure $\mathcal{I}_{hs}$ satisfies the following properties:*

- *If $\alpha \in \mathcal{K}$ is such that $\mathsf{At}(\alpha) \cap \mathsf{At}(\mathcal{K} \setminus \{\alpha\}) = \emptyset$ then $\mathcal{I}_{hs}(\mathcal{K}) = \mathcal{I}_{hs}(\mathcal{K} \setminus \{\alpha\})$ (*safe formula independence*).*
- *If $\mathcal{K} \equiv^{\sigma} \mathcal{K}'$ then $\mathcal{I}_{hs}(\mathcal{K}) = \mathcal{I}_{hs}(\mathcal{K}')$ (*irrelevance of syntax*).*
- *If $\alpha \models \beta$ and $\alpha \not\models \perp$ then $\mathcal{I}_{hs}(\mathcal{K} \cup \{\alpha\}) \geq \mathcal{I}_{hs}(\mathcal{K} \cup \{\beta\})$ (*dominance*).*

Proof.

- This is satisfied as safe formula independence follows from free formula independence, cf. [5, 13].
- Let H be a card-minimal hitting set of $\mathcal{K}$. So, for every $\alpha \in \mathcal{K}$ we have $\omega \in H$ with $\omega \models \alpha$. Due to $\alpha \equiv \sigma(\alpha)$ we also have $\omega \models \sigma(\alpha)$ and, thus for very $\beta \in \mathcal{K}'$ we have $\omega \in H$ with $\omega \models \beta$. So H is also a hitting set of $\mathcal{K}'$. Minimality follows from the fact that σ is a bijection.
- Let H be a card-minimal hitting set of $\mathcal{K}_1 = \mathcal{K} \cup \{\alpha\}$ and let $\omega \in H$ be such that $\omega \models \alpha$. Then we also have that $\omega \models \beta$ and H is also a hitting set of $\mathcal{K}_2 = \mathcal{K} \cup \{\beta\}$. Hence, $h_{\mathcal{K}_1} \geq h_{\mathcal{K}_2}$ and $\mathcal{I}_{hs}(\mathcal{K}_1) \geq \mathcal{I}_{hs}(\mathcal{K}_2)$.

□

Proposition 3. *A consistent partitioning Φ is a card-minimal partitioning of $\mathcal{K}$ if and only if $\mathcal{I}_{hs}(\mathcal{K}) = |\Phi| - 1$.*

Proof. Let $\Phi = \{\Phi_1, \ldots, \Phi_n\}$ be a consistent partitioning and let $\omega_i \in \mathsf{Int}(\mathsf{At})$ be such that $\omega_i \models \Phi_i$ (for $i = 1, \ldots, n$). Then $\{\omega_1, \ldots, \omega_n\}$ is a hitting set of $\mathcal{K}$ and we have $h_{\mathcal{K}} \leq |\Phi|$. With the same idea one obtains a consistent partitioning Φ from every hitting set H of $\mathcal{K}$ and thus $h_{\mathcal{K}} \geq |\Phi'|$ for every card-minimal partitioning of $\mathcal{K}$. Hence, $\mathcal{I}_{hs}(\mathcal{K}) = |\Phi| - 1$ for every card-minimal partitioning Φ of $\mathcal{K}$. □

Proposition 4. *Let $\mathcal{K}$ be a knowledge base. If $\infty > \mathcal{I}_{hs}(\mathcal{K}) > 0$ then*

$$1 - \frac{1}{\mathcal{I}_{hs}(\mathcal{K})} < \mathcal{I}_{\eta}(\mathcal{K}) \leq 1 - \frac{1}{\mathcal{I}_{hs}(\mathcal{K}) + 1}$$

Proof. For the right inequality, let H be a card-minimal hitting set of $\mathcal{K}$, i. e., we have $\mathcal{I}_{hs}(\mathcal{K}) = |H| - 1$. Define a probability function $P : \mathsf{Int}(\mathsf{At}) \to [0, 1]$ via $P(\omega) = 1/|H|$ for every $\omega \in H$ and $P(\omega') = 0$ for every $\omega' \in \mathsf{Int}(\mathsf{At}) \setminus H$ (note that P is indeed a probability function). As H is a hitting set of $\mathcal{K}$ we have that $P(\phi) \geq 1/|H|$ for every $\phi \in \mathcal{K}$ as at least one model of ϕ gets probability $1/|H|$ in P. So we have $\mathcal{I}_{\eta} \leq 1 - 1/|H| = 1 - 1/(\mathcal{I}_{hs}(\mathcal{K}) + 1)$. For the left inequality we only sketch a proof. Assume that $\mathcal{I}_{\eta}(\mathcal{K}) \leq 1/2$, then we have to show that $\mathcal{I}_{hs}(\mathcal{K}) < 2$ which is equivalent to $\mathcal{I}_{hs}(\mathcal{K}) \leq 1$ as the co-domain of $\mathcal{I}_{hs}$ is a subset of the natural numbers. If $\mathcal{I}_{\eta}(\mathcal{K}) \leq 1/2$ then there is a probability function P with $P(\phi) \geq 1/2$ for all $\phi \in \mathcal{K}$. Let $\Gamma_P = \{\omega \in \mathsf{Int}(\mathsf{At}) \mid P(\omega) > 0\}$ and observe $\sum_{\omega \in \Gamma_P} P(\omega) = 1$. Without loss of generality assume that $P(\omega) = P(\omega')$ for all $\omega, \omega' \in \Gamma_P$[12]. Then every $\phi \in \mathcal{K}$ has to be satisfied by at least half of the interpretations in Γ_P in order for $P(\phi) = \sum_{\omega \in \Gamma_P, \omega \models \phi} P(\omega) \geq 1/2$ to hold. Then due to combinatorial reasons there have to be $\omega_1, \omega_2 \in \Gamma_P$ such that either $\omega_1 \models \phi$ or $\omega_2 \models \phi$ for every $\phi \in \mathcal{K}$. Therefore, $\{\omega_1, \omega_2\}$ is a hitting set and we have $\mathcal{I}_{hs}(\mathcal{K}) \leq 1$. By analogous reasoning we obtain $\mathcal{I}_{hs}(\mathcal{K}) \leq 2$ if $\mathcal{I}_{\eta}(\mathcal{K}) \leq 2/3$ (and therefore $P(\phi) \geq 1/3$ for all $\phi \in \mathcal{K}$) and the general case $\mathcal{I}_{hs}(\mathcal{K}) \leq i$ if $\mathcal{I}_{\eta}(\mathcal{K}) \leq (i-1)/i$ and, thus, the claim. Note finally that $\mathcal{I}_{\eta}(\mathcal{K}) = 1$ if and only if $\mathcal{K}$ contains a contradictory formula which is equivalent to $\mathcal{I}_{hs}(\mathcal{K}) = \infty$ and thus ruled out. □

Corollary 1. *If $\mathcal{I}_{\eta}(\mathcal{K}_1) \leq \mathcal{I}_{\eta}(\mathcal{K}_2)$ then $\mathcal{I}_{hs}(\mathcal{K}_1) \leq \mathcal{I}_{hs}(\mathcal{K}_2)$.*

Proof. We show the contraposition of the claim, so assume $\mathcal{I}_{hs}(\mathcal{K}_1) > \mathcal{I}_{hs}(\mathcal{K}_2)$ which is equivalent to $\mathcal{I}_{hs}(\mathcal{K}_1) \geq \mathcal{I}_{hs}(\mathcal{K}_2) + 1$ as the co-domain of $\mathcal{I}_{hs}$ is a subset of the natural numbers. By Proposition 4 we have

$$\mathcal{I}_{\eta}(\mathcal{K}_1) > 1 - \frac{1}{\mathcal{I}_{hs}(\mathcal{K}_1)} \geq 1 - \frac{1}{\mathcal{I}_{hs}(\mathcal{K}_2) + 1} \geq \mathcal{I}_{\eta}(\mathcal{K}_2)$$

which yields $\mathcal{I}_{\eta}(\mathcal{K}_1) > \mathcal{I}_{\eta}(\mathcal{K}_2)$. □

Proposition 5. *Let $\mathcal{I}$ be an inconsistency measure, $w \in \mathbb{N}$, and g some function $g : [0, \infty) \times [0, \infty) \to [0, \infty)$ with $g(x, y) \in [\min\{x, y\}, \max\{x, y\}]$.*

1. If w is finite then $\mathcal{J}_{\mathcal{I}}^{w,g}$ is not *an approximation of $\mathcal{I}$.*

[12] Otherwise let $k \in \mathbb{Q} \cap [0, 1]$ be the least common denominator of all $P(\omega)$, $\omega \in \Gamma_P$, and replace in Γ_P every ω by k duplicates of ω with probability $P(\omega)/k$ each; for that note that P can always be defined using only rational numbers, cf. [8]

2. *If $w = \infty$ and $g(x,y) > \min\{x,y\}$ if $x \neq y$ then $\mathcal{J}_{\mathcal{I}}^{w,g}$ is an approximation of $\mathcal{I}$.*
3. *$\mathcal{J}_{\mathcal{I}}^{w,g}(\mathcal{S}_{\mathcal{K}}, i) \leq \mathcal{I}(\mathcal{K})$ for every $\mathcal{K} \in \mathbb{K}$ and $i \in \mathbb{N}$.*

Proof.

1. Assume $\mathcal{K}$ is a minimal inconsistent set with $|\mathcal{K}| > w$. Then $\mathcal{I}(\mathcal{S}^{\max\{0,i-w\},i}) = 0$ for all $i > 0$ (as every subset of $\mathcal{K}$ is consistent) and $\mathcal{J}_{\mathcal{I}}^{w,g}(\mathcal{S}, i) = 0$ for all $i > 0$ as well. As $\mathcal{I}$ is an inconsistency measure it holds $\mathcal{I}(\mathcal{K}) > 0$ and, hence, $\mathcal{J}_{\mathcal{I}}^{w,g}$ does not approximate $\mathcal{I}$.
2. If $w = \infty$ we have $\mathcal{I}(\mathcal{S}^{\max\{0,i-w\},i}) = \mathcal{I}(\mathcal{K})$ for all $i > i_0$ for some $i_0 \in \mathbb{N}$. As $g(x,y) > \min\{x,y\}$ the value $\mathcal{I}(\mathcal{K})$ will be approximated by $\mathcal{J}_{\mathcal{I}}^{w,g}$ eventually.
3. This follows from the fact that $\mathcal{I}$ is a basic inconsistency measure and therefore satisfies $\mathcal{I}(\mathcal{K}) \leq \mathcal{I}(\mathcal{K}')$ for $\mathcal{K} \subseteq \mathcal{K}'$.

□

Proposition 6. *For every probability $p \in [0,1)$, g some function $g : [0,\infty) \times [0,\infty) \to [0,\infty)$ with $g(x,y) \in [\min\{x,y\}, \max\{x,y\}]$ and $g(x,y) > \min\{x,y\}$ if $x \neq y$, a monotonically decreasing function $f : \mathbb{N} \to [0,1]$ with $\lim_{n\to\infty} f(n) = 0$, and $\mathcal{K} \in \mathbb{K}$ there is $m \in \mathbb{N}$ such that with probability greater or equal p it is the case that $\lim_{i\to\infty} \mathcal{J}_{hs}^{m,g,f}(\mathcal{S}_{\mathcal{K}}, i) = \mathcal{I}_{hs}(\mathcal{K})$.*

Sketch. Consider the evolution of single candidate set $C_1 \in Cand$ during the iterated execution of $\texttt{update}_{hs}^{m,g,f}(form)$, initialized with the empty set $\emptyset$. Furthermore, let $\hat{C}$ be a card-minimal hitting set of $\mathcal{K}$. In every iteration the probability of selecting one $\omega \in \hat{C}$ to be added to C_1 is greater zero as at least one $\omega \in \hat{C}$ is a model of the current formula. Furthermore, the probability of *not* removing any interpretation $\omega' \in C_1$ is also greater zero as f is monotonically decreasing (ignoring the very first step). Therefore, the probability p_1 that C_1 evolves to $\hat{C}$ (and is not modified thereafter) is greater zero. Furthermore, the evolution of each candidate set $C_i \in Cand$ is probabilistically independent of all other evolutions and by considering more candidate sets, i. e., by setting the value m large enough, more candidate sets will evolve to some card-minimal hitting set of $\mathcal{K}$ and the average cardinality of the candidate sets approximates $\mathcal{I}_{hs}(\mathcal{K}) + 1$. □