Contelog: A Formal Declarative Framework for Contextual Knowledge Representation and Reasoning

Ammar Abdulbasit Alsaig

A Thesis

in

The Department

of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy (Software Engineering) at

Concordia University

Montréal, Québec, Canada

June 2022

CONCORDIA UNIVERSITY

SCHOOL OF GRADUATE STUDIES

This is to certify that the thesis prepared

By: Ammar Abdulbasit Alsaig

Entitled: Contelog: A Formal Declarative Framework for Contextual Knowledge Representation and Reasoning

and submitted in partial fulfillment of the requirements for the degree of

Doctor Of Philosophy (Software Engineering)

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

____ Chair
Dr. Arash Mohammadi

____ External Examiner
Dr. Hasan Jamil

____ Thesis Co-Supervisor
Dr. Vangalur Alagar

____ Thesis Co-Supervisor
Dr. Nematollaah Shiri

____ Examiner
Dr. Gosta Grahne

____ Examiner
Dr. Jamal Bentahar

____ Examiner
Dr. Dhrubajyoti Goswami

Approved by ____
Dr. Leila Kosseim, Graduate Program Director

6/6/2022 ____
Dr. Mourad Debbabi, Dean
Gina Cody School of Engineering and Computer Science

Abstract

Contelog: A Formal Declarative Framework for Contextual Knowledge Representation and Reasoning

Ammar Abdulbasit Alsaig Ph.D.

Concordia University, 2022

Context-awareness is at the core of providing timely adaptations in safety-critical secure applications of pervasive computing and Artificial Intelligence (AI) domains. In the current AI and application context-aware frameworks, the distinction between knowledge and context are blurred and not formally integrated. As a result, adaptation behaviors based on contextual reasoning cannot be formally derived and reasoned about. Also, in many smart systems such as automated manufacturing, decision making, and healthcare, it is essential for context-awareness units to synchronize with contextual reasoning modules to derive new knowledge in order to adapt, alert, and predict. A rigorous formalism is therefore essential to (1) represent contextual domain knowledge as well as application rules, and (2) efficiently and effectively reason to draw contextual conclusions. This thesis is a contribution in this direction. The thesis introduces first a formal context representation and a context calculus used to build context models for applications. Then, it introduces query processing and optimization techniques to perform context-based reasoning. The formal framework that achieves these two tasks is called Contelog Framework, obtained by a conservative extension of the syntax and semantics of Datalog. It models contextual knowledge and infers new knowledge. In its design, contextual knowledge and contextual reasoning are loosely coupled, and hence contextual knowledge is reusable on its own. The significance is that by fixing the contextual knowledge, rules in the program and/or query may be changed. Contelog provides a theory of context, in a way that is independent of the application logic rules. The context calculus developed in this thesis allows exporting knowledge inferred in one context to be used in another context. Following the idea of Magic sets from Datalog, Magic Contexts together with query rewriting algorithms are introduced to optimize bottom-up query evaluation of Contelog programs. A Book of Examples has been compiled for Contelog, and these examples are implemented to showcase a proof of concept for the generality, expressiveness, and rigor of the proposed Contelog framework. A variety of experiments that compare the performance of Contelog with earlier Datalog implementations reveal a significant improvement and bring out practical merits of current stage of Contelog and its potential for future extensions in context representation and reasoning of emerging applications of context-aware computing.

Acknowledgments

First and foremost, praises and thanks to God, the Almighty, for his showers of blessings throughout my journey to complete my research successfully.

I would like to express my deep and sincere gratitude to my research supervisors, Dr. Vangalur Alagar and Dr. Nematollaah Shiri, for their continuous support throughout the research. Dr. Alagar is and has been always a father, a teacher, and a motivator to me throughout my academic journey since the beginning. His wisdom has been a savior to me in many occasions, and has helped me in making important academic decisions. Dr. Nematollaah is a big brother, he has always been there for me when I needed him. His thoughtful comments have helped me even at both academic and personal levels. I am sincerely indebted to both of them, they have both been kind, helpful, wise, and understanding.

My deepest thanks to my wife Sahar El Belbesi, and my kids Wedad, Sarah, Yassir, and Meriem and to my sister Alaa Alsaig for the support they have given me all along. My academic performance would have never been the same without their continuous cheering. They have been around at my highs and my lows, supported me at home and outside. No words can describe my gratitude to them, and I can never thank them enough.

My thanks to my parents, their sacrifices are endless. Thanks for their support, prayers, and patience. Thanks for all sort of support they have provided along the way. My deepest sincere thanks goes to them for their unconditional love that surrounded me with the warmth whenever I needed it.

My special gratitude to the people I lost along the journey, they have helped me but did not last until the end. I am greatly indebted to them and to everyone helped me along the way. It is hard to exhaustively thank everyone, thus I say to all people I love, my friends and family, all people who happened to help me even once, thank you.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Knowledge base (KB) is one of the main components of Expert Systems, a sub-field of Artificial Intelligence (AI). The two main parts of KB *are knowledge specification* (facts and rules) and *an inference engine* for reasoning. The initial set of facts are the static knowledge of an application acquired by the system, and the rules which enable derivation of new facts from the static knowledge. Once a new facts are derived, they are added to the initial set of facts, and used to further generate new facts. The derivation process terminates when no new fact is generated.

Expert Systems use KBs (general and/or specific) to solve problems in many application domains. Examples of Expert Systems include (1) The MYCIN experiments of the Stanford Heuristic Programming Project (Buchanan, Shortliffe, et al., 1984), (2) Rule-based Medical Diagnosis systems (Clancey, 1983; Lamperti & Zanella, 2003), and (3) Rule-based Access Control (Carminati, Ferrari, & Perego, 2006) Systems. The MYCIN project demonstrated that for many of these applications, it is necessary that a fact is "valid" in a given "context" in order to be useful. Hence, rule-based reasoner should not only be able to consider facts but also consider the "contexts" in which those facts are true. This thesis builds on this theme and develops a formal framework, called *Contelog* , for context-aware KB systems.

Context as a *concept* was discussed as early as 1960s in several disciplines, such as linguistics and psychology (Kintsch & van Dijk, 1978), philosophy (Kinneavy, 1971; Moffett, 1968), and sociology (Merton, 1973). It was only in 1993 that it was introduced in the field of AI by McCarthy (McCarthy, 1993). His intention was to generalize rules to not only cover a specific domain but to do so in different contexts. Several other authors (Attardi & Simi,

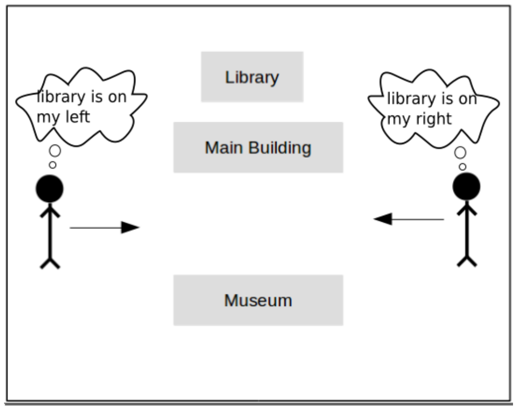


Figure 1.1: Illustration of the Building Locator Example

1995; Buvač & Mason, 1993; Giunchiglia, 1993; Guha, 1991) followed his effort to formalize context-based reasoning and introduced new theories for contextualization of knowledge. Many theories and concepts that discuss similar concerns emerged subsequently. Some notable ones are Situation Theory (Barwise, 1989), Temporal Logic (Lamport, 1980; Ostroff, 1989), and Reasoning in Time and Space (Alvaro et al., 2011). Albeit these great theoretical endeavors, a formal representation of context was not studied. Consequently, contextual information and rules could not be made explicit and expressive. As a result, these theories remained "just theories or domain specific theories" and were not mature enough for use in any practical application domain. As discussed in detail in Chapter 2, a review of all works beginning from the earliest example of "Building Locator" Ackman (Akman & Surav, 1996) to many recent publications on technology-driven Ubiquitous and Pervasive Computing Systems (Brezillon & Abu-Hakima, 1995; Gao & Dong, 2017; Kofod-Petersen & Mikalsen, 2005; Orsi & Tanca, 2011; Perttunen, Riekki, & Lassila, 2009), reveal that only ad-hoc methods have been used to implement "context-dependent features". To expose the pitfalls on the use of notations that lack theoretically sound formal context representation, we discuss the "Building Locator" example below. The discussion below also brings out the several logical inconsistencies that arise when logical notations (Guha, 1991; McCarthy, 1993), although not informal, instead of disambiguating context notation, are used for contextual problem solving. When contexts, as first class citizens are modeled independently and separately from the rest of the facts and rules of an application, ambiguous syntax and inconsistent interpretations disappear.

Building Locator Example

We use the building locator example (Akman & Surav, 1996), depicted in Figure 1.1, to motivate the need for a concise formal context notation. The goal in "Building Locator" example is to direct a user coming from one direction (east or west) to the building situated at a side (left or right) of the road. Clearly, the directives to people to reach their destination buildings depend on the directions from where they are coming, as the locations of buildings are static. At the time Akman and Surav (Akman & Surav, 1996) discussed this problem, the context information inside the "rich context notion" cannot be separated and represented. Consequently, they gave a simple narrative in "natural language" to present the possible solutions for different people coming from different directions. In order to use a formal reasoner to derive the results, we need to formally represent information involved in the scenario including contexts. Due to the lack of any formal representation of contexts, McCarthy (McCarthy, 1993), Guha (Guha, 1991), and Brewka (Brewka & Eiter, 2007) used propositional logic. Following Guha (1991), we use predicate logic notation to represent contextual information. This sheds some light on the non-triviality of formalizing the problem.

Let $p(X, Y)$ and $side(X, Z)$ be binary predicates, where X represents a person, Y is the direction from which the person is coming, and Z is the side to where the person should be going. Let $from(W)$ and $to(M)$ be unary context predicates (they appear in contexts only), where W is an ordinal direction (east/west) and M is the side (right/left) of the library with respect to a direction of W. We use McCarthy's expression $ist(C, P)$ that asserts "predicate P is true in context C." Let the contexts in the program domain be c_1 and c_2, where $c_1 = \{from(east), to(right)\}$ qualifies a person who comes from "east" and the library is located on his/her "right", and $c_2 = \{from(west), to(left)\}$ qualifies a person who comes from "west" and the library is on his/her "left." For example, a person who is coming from the east is in context c_1, and based on this context the library is on his right. The symbol "$\equiv''$ is used to state equivalence of the two expressions. The symbol r_1 is used to label the rules, and the symbol $r_1(c_1)$ is used to label a rule r_1 inside the context c_1. Program *IMPL-M* shown below is a possible logic programming implementation using the notation and concepts introduced in (McCarthy, 1993). In this rather straightforward implementation we observe the following drawbacks.

(1) Assertions on contexts are given as part of the facts. Because contexts are not separated from facts, there is lexical ambiguity due to the possibility of overlapping vocabularies for facts and contexts.

(2) The rules are not purely context-based, because the contexts are parts of the facts. Because facts are *static*, the rules that are static will always remain true. This means that we lose the concept of "contextual change or dynamic context".

(3) As a consequence of the above drawbacks, the implementation of program *IMPL-M* lacks dynamism, and might lead to inconsistent (or ambiguous) interpretations.

Program: IMPL-M

```
#FACTS
$p(john, east)$.
$p(rose, west)$.
$ist(c_1, from(east) \rightarrow to(right))$.
$ist(c_2, from(west) \rightarrow to(left))$.
#RULES
$r_1$: $p(X,Y), ist(C, from(Y) \rightarrow to(Z)) \rightarrow ist(C, side(X,Z))$
```

Using the concept of "lifting rules" proposed in (Guha, 1991), we can remove some drawback in the program *IMPL-M* to derive the program *IMPL-G*. Lifting rules allows importing assertions from a context to another. In lifting rule lr_1, the predicate $p(X,Y)$ ranges over the facts and the variable C ranges over the contexts. The variable "X" in $C(X)$ is used only to *carry* the information back to the problem solving context. This would not change the content or meaning of the context. As shown in the implementation, the assertion $ist(C, p(X,Y))$ can be lifted to $ist(C(X), from(Y))$. It lifts the assertion $p(X,Y)$ from a context C to the assertion $from(Y)$ in the context $C(X)$. This makes the implementation "dynamic." To clarify, using the rule r_1 and the fact $p(john, east)$, we can conclude $ist(C, p(john, east))$. Because C is a variable, we can *substitute* c_1 for C and use the first lifting rule lr_1 to conclude $ist(c_1(john), from(east))$. This substitution process is referred to as "entering" a context. Since c_1 includes the predicate $from(east)$, we derive $to(right)$, and hence $ist(c_1(john), to(right))$. This in turn enables the second lifting rule lr_2, using which we derive $ist(c_1, side(john, right))$. We then can use r_2 to derive

$side(john, right)$. That is, with respect to the direction from which john is approaching the building, we conclude that the building is on his right. This substitution results in the so-called "exiting" a context.

Program: IMPL-G

```
#FACTS
$p(john, east)$.
$p(rose, west)$.
#RULES
$r_1$: $p(X,Y) \rightarrow ist(C, p(X,Y))$.
$r_2$: $ist(C, side(X,Y)) \rightarrow side(X,Y)$.
$c_1$: $from(east) \rightarrow to(right)$.
$c_2$: $from(west) \rightarrow to(left)$.
#LIFTING RULES
$lr_1$: $ist(C, p(X,Y)) \equiv ist(C(X), from(Y))$.
$lr2$: $ist(C(X), to(Z)) \equiv ist(C, side(X,Z))$.
```

A comparison of the implementations of *IMPL-M* and *IMPL-G* reveals that the former lacks dynamism, while *IMPL-G* lacks simplicity. Both implementations, due to lack of proper notation and formal representation of context, lack economy of representation and formal separation of concerns. In turn, programs lack simplicity and reusability. Motivated by the above, we propose the formal framework *Contelog* in which the “rich context” of McCarthy and Guha are disassembled into (1) a formal meta-information part, which we call *context in our work* , and (2) a formal specification of facts and rules, which are parts of the KB system. They are separate, and each part can exist without the other. So, using those separated Context and *Contelog* notations, the program for the building locator problem can be concisely expressed by the following context 1.1 facts and rules in Contelog 1.2.

Listing 1.1: Program: Context of Building Locator Example

```
$c_1 = \{from : [east], to : [right]\}$
$c_2 = \{from : [west], to : [left]\}$
```

Listing 1.2: Contelog Program of Building Locator Example

```
$r_1$: $p(X,Y)@C \leftarrow p(X,Y), from(Y)@C$.
$r_2$: $side(X,Z)@C \leftarrow p(X,Y)@C, to(Z)@C$.
```

where $P@C$ corresponds to the Mccarthy's predicate $ist(C,P)$. Later on we will justify that our proposed framework is more general, flexible, modular, reusable, with the advantages of the declarative and fixpoint semantics.

1.1 Contributions

The problem of "context notation" was brought up in (McCarthy, 1993), and carried on by (Guha, 1991) in the following statement

> "the careful reader of the derivation will wonder what system of logic permits the manipulations involved, especially the substitution of sentences for variables followed by immediate use of the results of the substitution. There are various notation systems that can be used, e.g. quasi-quotation as used in Lisp, or Knowledge Interchange Format (KIF) and use of back-quotes Buvac (Buvač & Mason, 1993). But all have disadvantages. At present we are more attached to the derivation than to any specific logical system"

Later Akman Akman and Surav (1996) postulated that the following five advantages can be gained through a formal context representation.

- *Economy of Representation:* Data can be represented in different views using same structure of knowledge base.
- *Efficiency of Reasoning:* By limiting down the domain of reasoning to specific context reasoning power is increased.
- *Allowance for Inconsistent knowledge:* Inconsistencies can be identified and resolved by interpreting data in different contexts.
- *Resolving Lexical Ambiguity:* Explicitly introducing context removes lexical ambiguity.
- *Removing Ambiguities of Language:* Overlapping vocabularies may exist in knowledge base but are disambiguated by context.

But Akman and his followers did not propose a notation. Having established the need to have a formal representation, the first contribution of this thesis is a formal context representation that fulfills the above postulates and yet easy to fit into a logical reasoning system. The second contribution is the choice of contextual knowledge representation platform in which contextual reasoning can be conducted. In making these contributions we have critically studied existing literature to make diligent choices that can be extended to meet design principles such as *simplicity*, *generality*, *extensibility*, and *reuse*.

Our context formalism captures many popular definitions, such as "context is a settings of event, or environment specifications of event" that are used by a large volume of literature in context-aware computing (Brézillon, 1999; et al., 2009; Strang & Linnhoff-Popien, 2004), thus enabling them to use *Contelog* framework to formalize their applications. We make our relational algebraic context notation richer than the earlier functional notation (Wan, 2006; Wan, Alagar, & Paquet, 2005). Our context formalization includes a context calculus, and a context lattice. Keeping in mind that contextual knowledge should be reusable, we model context separately from the *Contelog* program itself. That is, set of contexts can be used for many *Contelog* programs. We require that the context structure is a *complete lattice*, in which the *meet* and *join* operations are used to define the declarative and fixpoint semantics of *Contelog* programs. This provides a theoretically sound and practically feasible framework for representing and reasoning with contextual information.

We extend Datalog (Abiteboul, Hull, & Vianu, 1995a; Ceri, Gottlob, & Tanca, 1989) in a conservative manner for representing contextual knowledge and reason about them. The relational syntax of context fits well with our extension. That is, in extended Datalog syntax these contexts are used in defining contextual facts and rules. We take advantage of the strong theoretical base, namely, simple syntax, declarative semantics, and powerful query processing and optimization techniques of Datalog, and extend it conservatively for contextual knowledge representation and contextual reasoning. That is, the syntax of contextual rules and semantics of their evaluations do not violate those of Datalog. We show in the thesis that the complexity of *Contelog* query processing is of the same order as evaluation of Datalog queries, namely polynomial in the number of constants in D (Grau, Horrocks, Kaminski, Kostylev, & Motik, 2020; Liang, Fodor, Wan, & Kifer, 2009).

Towards realizing a practical system, we have developed a context toolkit and an interface for a naive user to create and run *Contelog* programs. By a loose coupling, the program body (rules) can be executed by changing contextual facts. Thereby, we achieve reasoning in different contexts. Alternately, one set of contextual facts can be imported to different program bodies (rules get changed here). Thereby, contextual knowledge becomes reusable, enabling different types of reasoning on one set of knowledge. From our experience in a number of problems that we have solved (Alsaig, 2017) using the prototype system developed, we are optimistic that our approach has the potential to fill the gap between theory and practice that has existed in the art of contextual reasoning. The above contributions are structured in this thesis as follows.

(1) **Context Theory**

  (a) An extensive literature survey on context representations and operations.

  (b) Complete representation of context component.

  (c) Context Calculus: Set of operations and semantics of contexts.

  (d) Complete Theory: Context Lattice.

(2) **Context-Based Reasoning**

  (a) A study of high volume of research on logic-based context-based reasoning.

  (b) Define a link between contexts and logic in terms of rules and facts.

  (c) *Contelog* : A conservative extension of Datalog syntax, semantics, and theories to include Contexts as first class citizen.

  (d) Provide a proof of Declarative semantics (Horn Clasues) on *Contelog*

  (e) Develop bottom-up approach and proof of fix-point termination.

(3) **Query Processing and Peformance**

  (a) Naive bottom-up evaluation.

  (b) Semi-naive bottom-up evaluation.

  (c) Magic Context: An efficient transformation approach for *Contelog* and Datalog.

(4) **Prototyping and Testing**

(a) Provide a book of examples that provides diverse set of *Contelog* programs.

(b) Provide a complete implementation of *Contelog* with all its features and query processing techniques.

(c) Provide a complete implementation of Context ToolKit that helps at defining and adding new contexts.

(d) Provide a complete implementation of Profiling Kit that measures *Contelog* performance at several stages: reading (tokenization), scanning (syntax and semantics checking), and inferencing.

(e) Provide an extensive performance testing with different engines.

The structure of the thesis is as follows: Chapter 2 provides complete literature review on context 2.2 and on context-based reasoning 2.3. Chapter 3 provides the methodology and the evaluation criteria of the thesis. Chapter 4 introduces context theory via formalization, representation, semantics, operations, and finally context-lattice. Chapter 5, discusses *Contelog* framework and incrementally extends and builds its components, which includes, syntax, semantics, declarative semantics, model theory, and proof theory. Query Optimization techniques, and discussions on performance optimization is introduced in Chapter 6. In Chapter 7, the applications of *Contelog* engine prototype are discussed. The comparative studies to test the performance of *Contelog* is provided in Chapter 8. Finally, Chapter 9 provides a conclusion of the work and potential extensions for *Contelog* in the future research endeavours.

Chapter 2

Literature Survey

Context is a term that has been around for centuries, and it has been used in multiple disciplines with different synonyms. It has been extensively used in the study of Philosophical discourses (Kinneavy, 1971; Wan, 2006), Natural Languages Processing (Brézillon, 1996; Wan, 2006), Linguistics (Carnap, 1947; Kintsch & van Dijk, 1978), and Psychology (Clark & Carlson, 1981). The synonyms used in these disciplines are , event (Mueller, 2014), temporal context (Moldovan, Clark, & Harabagiu, 2005), situation (Henricksen & Indulska, 2006), neoma (Brézillon, 1996), explanation (Brézillon, 1996), ontology (Akman & Surav, 1997), view point (Attardi & Simi, 1995), and perspective (Giunchiglia, 1993). Given this large diversity of literature and terms for context, it is not possible to go into an exhaustive literature survey. In this chapter we restrict the review of literature that is relevant to this thesis. First, we motivate the structuring for our presentation. Following it the survey is given in detail within the restricted scope.

2.1 Motivation for Structuring the Review

Although each synonym had been used in a specific circumstance and towards diverse purposes, all terms are faces of the same coin "context". They all refer to the Greek meaning of the word "context" which consists of the two words "con" and "texere", where the word "con" means "together" and the word "texere" means "to weave" (Wan, 2006). Combined, it means "to weave together". This refers to the goal of using context, namely "to weave" circumstances together to understand the knowledge behind spoken or written information. Three decades ago Ackman (Akman & Surav, 1996) suggested the use of context in computer science. Since then, research in context has contributed to different fields of study in computing. The following is a quick overview of the multiple directions of research. This summary sets the stage for structuring our in-depth literature survey.

(1) **Informal Approaches to Computing Applications:**

- *Interdisciplinary Emphasis:* The nature of diversity and interdisciplinary research can be seen in LNAI publication series "Modeling and Using Context", proceedings of CONTEXT International conference being held from 1997 (Interdisciplinary & Series, 1997-). More recent collection of papers in (Brèzillon & Gonzalez, 2014) only reinforces the practice of a variety of mostly informal notations and views in this interdisciplinary research on context.
- *Human Computer Interface Group (HCI)*: Early in 2001 the works of Dey, Abowd, and Salber (Dey et al., 2001) and Winograd (Winograd, 2001) in HCI are based on intuitive ad-hoc notations, bordering on vagueness and informality. These papers bring out also the disagreement within this group in conceptualizing and modeling contexts.
- *Pervasive Computing:* Based on the survey papers (Dey, 2001; et al., 2009), it is evident that the notion of context in pervasive computing was mostly ad-hoc. The formalisms attempted by a few (Brézillon, 1996) are non-rigorous, either domain-specific or application dependent . Besides, there is no consensus in this community on what a context should be and how it should be represented for pervasive computing application domain. While formal representation and reasoning procedures are quite important to reason about pervasive computing applications that involve human safety and privacy, they are not emphasized in these works.

(2) **First Attempts to Formalize Context Notation:**

- *Languages:* The works of Dowley, Wall, and Peters (Dowley et al., 1981), Sato, Sakurai, and Kameyama (Sato et al., 2001), Wan (Wan, 2006), and Wan, Alagar, and Pacquet (Wan, Alagar, & Pacquet, 2005) use contexts at different levels of abstraction in intensional and functional programming languages. Wan uses a formal context representation based on functional semantics while Sato et al. uses "place-holder" notation with λ calculus semantics. A context toolkit, based on the context calculus of Wan (Wan, 2006), was developed subsequently and its reuse potential in different applications such as privacy and security enforcement, was discussed in (Alagar & Wan, 2008).

- *Context-aware Computing:* With the advent of the term "ubiquitous computing" and "personalized services", the context term re-emerged and the term "context-aware" was born. Due to the lack of practically feasible theories and the gap between theory and practice, many context-aware applications (Korkea-Aho, 2000) use only ad-hoc notations and application-tailored approaches to deal with contexts. These approaches tend to solve only the specific problem the system was made for. The introduction of dimensions by Dowley et al. (Dowley et al., 1981) and Wan, Alagar, and Pacquet (Wan, Alagar, & Pacquet, 2005) to deal with hidden contexts in intensional programming languages perhaps influenced Schilit et al. (Schilit et al., 1994) to incorporate it in the development of context-aware systems, although there is no evidence that such an attempt resulted in a concrete syntax. Later on, the context representation formalized by Wan (Wan, 2006) was used in building context-aware systems (Alagar, Mohammad, Wan, & Hnaide, 2014b).

(3) **AI and Contextual Reasoning:** Context was brought into the field of Artificial Intelligence (AI) by McCarthy (McCarthy, 1993). Following this, his student Guha (Guha, 1991) continued to enrich the logical framework of McCarthy with more details and conciseness. Later, other researchers (Akman & Surav, 1996; Brézillon, 1996, 1999; Strang & Linnhoff-Popien, 2004) contributed to that basic foundation and built logical frameworks with different flavors. The works of Weyhrauch (Weyhrauch, 1980), McCarthy (McCarthy, 1993; McCarthy & Buvac, 1997), Guha (Guha, 1991), Akman and Surav (Akman & Surav, 1997), Shoham (Shoham, 1991), and Giunchiglia (Giunchiglia, 1993) are some of the early ground-breaking works on logic of contexts and context-based reasoning. In general, they either used propositional logic which restricted the expressive power of their framework or higher order logic which over

complicated it, and none of them use any formal notation to represent contexts as first class objects.

From this overview, it is evident that the literature is largely divided between works on context representation (both informal and formal) and framework for reasoning with contexts. So, we first survey the literature related to context representation, and next we survey the literature related to contextual reasoning. In reviewing each, we restrict to mostly work related to computing, AI, and logic. We present the methodology of the survey, and criteria of evaluating the surveyed methods. This criteria enables us to critically evaluate the surveyed methods and choose an approach that can subsume those methods in a formal way. We present a detailed technical analysis with respect to the criteria of evaluation under each reviewed method. Finally, we summarize our overall observations and critical comparisons at the end of the chapter.

2.2 A Critical Review of Context Modeling

In this part of review we focus solely on context, regardless of the framework where it may be used. That is, regardless of its applications, we look at context and try to see how previous endeavors conceptualized it. In order to have a fair and reasonable evaluation of the methods in the survey, a set of criteria of evaluation has been fixed. They act as single reference point to evaluate the merits of all methods. The selected criteria of evaluation come from the study of early work. The theoretical criteria of evaluation are related to *conceptualization*, *modeling*, and *representation* and they come from (Akman & Surav, 1996; Guha, 1991; McCarthy, 1993). The empirical criteria are that context should be *expressive*, *extensible* and *manipulable*. These come from (Alagar et al., 2014b; Held, Buchholz, & Schill, 2002a; Strang & Linnhoff-Popien, 2004). Below is an explanation and breakdown to each criterion.

(1) *Formal conceptualization* refers to the notation that will abstractly and sufficiently define context theory.

(2) *Formal model* means that entities in context are precise, formally related, well-defined, complete, and sound.

(3) *Formal representation* means that context is definable precisely using mathematical notations. There is no implicit assumption on context relationships.

(4) *Expressiveness* has three aspects.

  (a) Freedom of structure (hierarchical vs flat).

  (b) Fixed Knowledge. (static information)

  (c) Inferred knowledge/rules. (dynamic information).

(5) *Extensibility* has three aspects.

  (a) Supports heterogeneous data; not limited to specific type of information.

  (b) Innumerable; not limited to a specific number of entities.

  (c) Detachable; can be used as a standalone component in different systems.

(6) *Manipulable* means that the context formal model is supported by operations that are well-defined within the proposed context theory.

In what follows, each reviewed method is explained, then critically analyzed with respect to the above-mentioned criteria of evaluation. At the end of this chapter a summary of the analysis is provided in Table 2.1.

2.2.1 Modeling Context Using Informal Notation

In addition to the papers mentioned under - Informal Approaches to Computing Applications - in Section 2.1, there are several that use only informal notations for context in several applications of computer science. They are reviewed here.

Context as Intention. Early philosophical and psychological research thought of context as an implicit intention (Brézillon, 1996). In these studies, the context was not defined nor was it introduced formally. There was no explicit notion of context. However, they agree on the existence of context and how it can implicitly describe an incident. Based on the intention concept, some researchers (Costa et al., 2012; Liu et al., 2015) have introduced context in the study of Intention-aware computer-based recommender systems. The intention-awareness refers to the ability of the system to know the user intention when requesting a service in order that it may provide correct services in an intelligently manner. The recommender systems use the users' profile as intentions. By understanding the profiles, the system infers the correct marketing strategy and materials to provide expected services. Although the awareness concept is clear, in the design of such systems only ad-hoc informal notations for context, intention, and awareness-centric actions that are specific to solve a particular application have been used. These efforts lack both generality and formality. For example, tables are used in (Costa et al., 2012), list and set of items are used in (Liu et al., 2015), and plain textual description is used in text (Chen et al., 2002) to represent context. These data structures are not backed-up with any theory for context. For these reasons, methods under this category are not considered in our comparisons.

Critical Analysis of Intention Notation

References in this category: (Costa et al., 2012) (Liu et al., 2015)

In this category, context concept was not defined. The term 'context' was mentioned along with other synonyms such as 'noema' and 'intention'. The work in this group does not meet any requirement in the evaluation criteria.

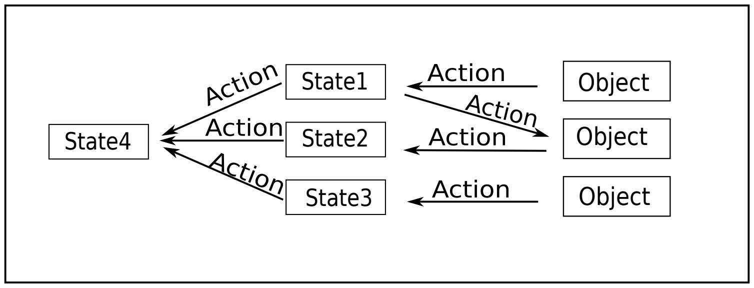


Figure 2.1: Schema-based explanation representation

Context as Explanation. In the field of Natural Language Processing (NLP) and 'Explanation based' learning which are studies by a large number of researchers as part of computing application, "explanation" is used as a supporting information that makes a sentence understandable. There are numerous NLP systems such as GENESIS (Mooney & DeJong, 1985), ACCEPTER (Leake, 2014), SWALE (Kass et al., 1986), and FAUSTUS (Norvig,

1983) that implement "explanations" as schemas. A schema structure uses "state" and "objects" structure to break down sentences in order to answer queries about input sentences. GENESIS, for instance, is based on a technique called "explanatory schema acquisition". Its idea is to represent explanation as set of states, actions and objects. These three form a "schema". States are linked to each other in a hierarchical manner depending on which states appeared first in the paragraph. A schema is expandable as new states/actions are introduced in the paragraph. Regardless of how schema evolves to collect more information, our concern is the structure of the explanation. Figure 2.1 depicts the structure of a schema for an explanation. Explanation can only have one schema, a schema can have set of unique states that are linked to other states/objects through actions. The literature provides no explicit framework of notation for explanation, schema, and/or actions. The explanation structures introduced for explanation-based learning and explanation-based NLP systems are almost identical. Thus, we will use GENESIS structure to evaluate the explanation representation and formalism in this category. Summary of evaluation and observations on the structure are given in the observation and analysis section 2.3.1.

Critical Analysis of Explanation Notation

References in this category: (Kass et al., 1986; Leake, 2014; Mooney & DeJong, 1985; Norvig, 1983)

All works in this category were inherited from the field of NLP. Although the notation is structured, it is only informal and incomplete, because a triple is not explicitly associated with the triple that specifies a new $object, state, action$. Also, other questions such as "whether a state/object pair is unique?", "are object/state pairs finite?", and "can every context be represented in this notation?". Similar to intention, we can summarize that none of the work we reviewed in this group have met any of our requirements.

Modeling Context Using Ontology. In this section we review only works that use informal structures for ontology. As stated in (Guarino et al., 1998), many works use XML and graph-based notation only informally to capture the hierarchical object-oriented structure and multi-sorted relations in an ontology. Although they all refer to the same basic concept of ontology, they differ in how they detail their ontological trees. For instance, in a very recent work (Gao & Dong, 2017), the authors use attributes and values for contextual information. For example, they define user context ontology as $UCA = UBI(ID, AGE, SEX)$, where UCA is the user context ontology, UBI is the domain, and ID, AGE, and SEX are attributes. This is simply an ad-hoc notation. Similarly, the work in (Wang et al., 2004a) which is referred to as CONON project, defines context based entities only informally. The entities introduced are user, activity, location, and computation. Due to lack of formalism, representation, and entity-specific operations, there is lexical ambiguity that limits implementation correctness being verified. The work in (Ejigu et al., 2007) is identical to the previous work, the only exception is that the context structure in this work starts with a root named "Context Entity". The works on context reported in (Borgo, Cesta, Orlandini, & Umbrico, 2019; Gu et al., 2004; Lee et al., 2007; Shehzad et al., 2004; Umbrico, Cesta, Cortellessa, & Orlandini, 2020) are more concerned with the "content of context" relevant to an application, and use informal notations to describe them. The following thematic commonalities exist among those approaches.

- Each approach addresses one specific application.
- They are concerned with content and not the representation of context.
- They use XML/Graph notation to describe the content of context.
- They mention ontology, but due to the lack of formal structure no operation on any ontology is defined.

Critical Analysis of Ontology Notation

References in this category: (Gao & Dong, 2017) (Wang et al., 2004a) (Ejigu et al., 2007) (Gu et al., 2004; Lee et al., 2007; Shehzad et al., 2004)

There are many papers that use ontology to introduce context. They do not offer any formal conceptualization or notation of context. They do not provide a formal model that connects the pieces of an ontology together. However, they are credited for their expressiveness and extensibility. Although no operation has ever been provided to deal with context models (ontology), all methods inherit operations that have been proposed on the ontology.

(1) *Formal conceptualization:* The review works only provide the ability to structure context content. It does not offer a conceptualization of the context entity.

(2) *Formal model:* No precise definition of context is given in any of the works.

(3) *Formal representation:* No standard notation exists. Many implicit assumptions on relations between contexts/ontologies exist.

(4) *Expressiveness:*

  (a) Freedom of structure (hierarchical vs flat): All works on ontology support hierarchies and have high level of flexibility with respect to structuring.

  (b) Fixed Knowledge. (static information): An ontology can encapsulate static information.

  (c) Inferred knowledge/rules. (dynamic information): Ontological rules are only informal. Some of the works (Gao & Dong, 2017) do not support inferencing. Some support inference based on the structure of the ontology and the type of relations between the nodes.

(5) *Extensibility:*

  (a) Supports heterogeneous data: A single ontology is domain-related and cannot hold/deal with data from multiple domains.

  (b) Innumerable: Due to informal description, an ontology has no limitation on data it can hold. Therefore, it is innumerable.

  (c) Detachable: Perhaps the most challenging aspect and major shortcoming of ontology is the inability of detaching contextual data from factual information.

(6) *Manipulable:* Ontology operations are not formal. Hence, this requirement is not met.

2.2.2 Modeling Context Using Formal Notations

Modeling Context Using Formal Concept Analysis (FCA). There were many attempts to formalize context using FCA. FCA consists of its own "FCA contexts" (Sarmah, Hazarika, & Sinha, 2015) that are different from contexts discussed in AI and Computing. An FCA context consists of a set of objects, attributes, and their relationships expressed as a binary relation. A Formal concept is a pair of two FCA contexts that share the same objects and attributes. Formally, a FCA context is a triplet $FC =< G, A, I >$ where G is set of objects, A is set of attributes, I is a binary relation such that $I \subseteq G \times M$. The binary relation, referred to as *incidence*, expresses which object is linked with which attribute.

FCA context has the two well-defined operations *intent* and *extent*, also referred to as *Concept Forming Operators.* Operation *Intent* retrieves the set of attributes associated with a goal g, while operation *extent* retrieves the set of goals associated with an attribute m. Formally,

$$Intent = A^{\uparrow} = \{y \in Y | \text{for each} x \in A :< x, y >\in I\}$$

$$Extent = B^{\downarrow} = \{x \in X | \text{for each} y \in B :< x, y >\in I\}$$

A formal concept is defined as the fixpoint of the above operators. That is, if applying the intent on an FCA context F and then extent on the result produces F, then that is a fixedpoint, i.e. formal concept. That is, a formal concept in an FCA context $< X, Y, I >$ is a pair $< A, B >$, where $A \subseteq X$ and $B \subseteq Y$ such that $A^{\uparrow} = B$ and $B^{\downarrow} = A$. Also, *subsconcept-superconcept* ordering, which is the relationship between a more general concept and more specific concept, is well-defined. Particularly, for two formal concepts $< A_1, B_1 >$ and $< A_2, B_2 >$ of an FCA context <X,Y,I>, we define $< A_1, B_1 >\leq< A_1, B_2 >$ iff $A_1 \subseteq A_2$ (iff $B_2 \subseteq B_1$). This definition paves the way for the definition of the concept lattice, which contains all possible formal concepts for an FCA context closed with respect to the operators *Intent* and *Extent*.

Using FCA, the context notion in AI and Computing can be formally represented (Alsaig et al., 2015). In their work, they introduced FCA as a formal model to model and represent data of wearable devices. A wearable device is a device that is used to sense vital signs such as heart rate and temperature, and motion, then suggests a plan for best managing exercises and monitor calories burned. Usually, those wearable devices depends on three data components in the decision making procedure. First, the goal to be achieved by the user. Second, the attribute on which the user should focus on to be measured, i.e. weight, heart rate... etc. Third, the technique the user should follow to achieve their health goal.

The authors then constructed two different contexts, "the user context" which are the goals and attributes, and "the technique context" which contains the attributes that should be followed for certain goals. Using FCA, the authors used FCA-objects to represent goals, FCA-attributes to represent attributes, and FCA-node to represent the technique. Then, used the FCA table to construct the FCA-lattice that contains all nodes (techniques) for all user goals and attributes to be acheived. Using the operators (intent and extent) the suggested techniques can be calculated by traversing the tree towards the goals/attributes defined by the user. By using these two operators all possible nodes/techniques can be inferred.

Critical Analysis of FCA Notation

References in this category: (Alsaig et al., 2015)

FCA is credited for its set theoretical basis, and hence for well-defined operations and expressiveness. However, the goal in FCA is to formally capture "relationships between entities". Below is an analysis of using FCA notation for context.

(1) *Formal conceptualization:* Since FCA does offer a formal notation to structure concepts, it offers a forum for formal conceptualization.

(2) *Formal model:* FCA is silent on stating (1) whether or not sets of objects and attributes are finite, (2) are the set of variables static or can dynamically be refined, (3) what are the types and domains of variables. Hence, it does not meet this requirement.

(3) *Formal representation:* although it does have fixed notation, FCA provides no support to represent context.

(4) *Expressiveness:*

   (a) Freedom of structure (hierarchical vs flat): FCA has flexible structuring.

   (b) Fixed Knowledge. (static information): It supports only static information.

   (c) Inferred knowledge/rules. (dynamic information): There is no support for inferencing or reasoning in FCA.

(5) *Extensibility:*

(a) Supports heterogeneous data: FCA supports domain-free data and heterogeneous data.

(b) Innumerable: FCA tree can be infinite. Hence, it is innumerable.

(c) Detachable: With the help of concepts and goals definition in FCA, "context/-concept" can be detached from goals/attributes.

(6) *Manipulable:* It supports intent/extent operations that traverse the FCA tree upward/downward respectively.

Context as Event. Mueller (Mueller, 2014) defined *event* as an action that happens at a specific time/setting, and formalized it in propositional logic. It was not directly linked to context. However, in the related work of event-based processing (Shanahan, 1999a) event was defined as a *situation*. In these logic-based frameworks, event is treated as a constant and formalized as the ground predicate $p(a, e_1, t)$ to mean "event e_1 is true at time t with respect to attribute a". McCarthy who's considered to be a key player in the evolution of context theory has also contributed to *event calculus*. He first introduced the predicate $holds(c, e)$ (Shanahan, 1999b) to mean that "event e occurs during context c". That is, event and context were distinguished as separate entities. Later on, he changed it to the is-true predicate $ist(Context, Predicate)$, where context may be replaced by event. Therefore, we let event to be a synonym for context.

Critical Analysis of Event Notation

References in this category: (Mueller, 2014)

Although McCarthy (Shanahan, 1999b) has worked on event, situation and finally context, he did not use the work on event towards context. Even though it's unstated, it looks as though he wanted to avoid inadequacy in the event theory to be employed for context. Also, in all papers event definition associates only the single property "time of occurrence" (Brézillon, 1996) to event. The "time" dimension in a context is only one of many other possible dimensions that may be necessary to capture the specific "setting" (context).

(1) *Formal conceptualization:* No clear conceptualization exists for context/event entity.

(2) *Formal model:* No precise definition exists for context/event.

(3) *Formal representation:* It does have a fixed and abstract notation, hence, event notation meets this criterion.

(4) *Expressiveness:*

 (a) Freedom of structure (hierarchical vs flat): No structuring ability or relation can be established between events.

 (b) Fixed Knowledge. (static information): Event can model flat/static information. Hence, it meets this criterion.

 (c) Inferred knowledge/rules. (dynamic information): No notion of inferencing exists in the study of events.

(5) *Extensibility:*

 (a) Supports heterogeneous data: Events can deal with any type of data, and hence it supports heterogeneity.

 (b) Innumerable: It does not have any limitation on data it can hold. Therefore, it is innumerable.

 (c) Detachable: Only static information can be handled. Hence, it fails to meet this requirement.

(6) *Manipulable:* Some works have discussed event operations and event calculus. Hence, this requirement is met.

Context as a Partial Mathematical Entity.
McCarthy (McCarthy, 1993) (McCarthy & Buvac, 1997) made the first attempt at formally realizing context entity in the field of computer science. He originally proposed the idea of incorporating context with logic. He described context as rich objects that cannot be fully described in logic. Thus, every context is a subset of a larger context that contains it. He also considered context as first class objects in the sense that they belong to the domain of interpretation of a formal language. This means that the formal language of a theory of context should contain terms denoting contexts and should allow one to pass contexts as function arguments and to express relationships between contexts. In his view "one context is more general than another". Contexts in this category are *incomplete abstract*

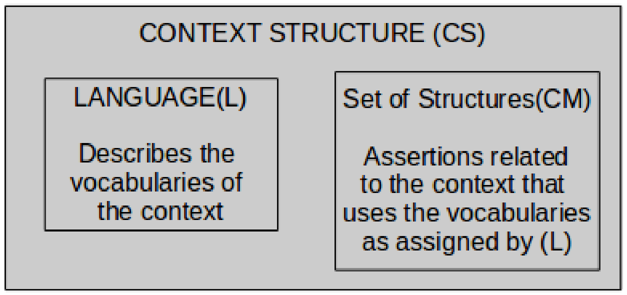

CONTEXT STRUCTURE (CS)
LANGUAGE(L)
Describes the vocabularies of the context
Set of Structures(CM)
Assertions related to the context that uses the vocabularies as assigned by (L)

Figure 2.2: Context structure as defined in Guha's work

objects that partially explains the world of discourse. Technically, context is an abstract mathematical entity with properties useful for AI. However, McCarthy's work only included a few highlights that intuitively motivated context formalization. It was not supported with any particular formalism, explicit definitions, and modeling of any sort.

Guha (Guha, 1991) regarded context as a *microtheory* that partially explains the world. Microtheories are theories of limited domains. They define the "context way" of seeing the world. A microtheory has its own Language and structure. It has its own set of facts and rules. Every theory (category) can have infinite number of microtheories that describe it. These microtheories can only describe part of the theory rather than complete theory. The goals of Guha's research are similar to those of McCarthy. The extension of Guha's work can be concluded in his definition of context as microtheory rather than general mathematical entities as McCarthy mentioned. Also, Guha introduced language and structure for context. It is characterized as follows:

- Every context is a *first class object*
- Each context has its own language (L) and structure (CM) (context way of describing the world)
- Context Structure (CS) is a function that assigns to every context C a language L(C), and a structure CM(C). Figure 2.2 shows the structure of context as proposed by Guha.

Although Guha's framework accomplishes a substantial progress towards formalizing context, it does not offer any explicit representation for context. Also, he defined a general framework for context, but did not explicitly investigate the details of context content.

Buvač and Mason (Buvač & Mason, 1993), extended the work of McCarthy while retaining the view that context is a partial and approximate mathematical entity. However, in this work, extension is only within propositional logic framework. In this research, every context has its own vocabulary, a set of propositional atoms that are meaningful/relevant in/to that context. Buvač and Mason modeled context using sets of partial truths. Also, they introduced the notion of *dependent contexts*, by which context-based contexts can be defined. However, Buvač and Mason did not introduce any formalization, modeling or representation for context.

In (Shoham, 1991), the authors introduced a new notation for viewing context, although their definition is similar to that of McCarthy. This is the first work in which operations and relations between contexts were introduced formally. They defined the partial order relation ($\bullet \supset$) that is considered a weak partial ordering between contexts. The five operations on context introduced in this research are (1) *and-closed* (closed under conjunction). (2) *or-closed* (closed under disjunction). (3) *and-or-closed* (closed under both). (4) *not-closed* (closed under negation). (5) simply *closed* if it is closed under all operations.

Critical Analysis of Mathematical Entity and Logic Notation

References in this category (McCarthy, 1993; McCarthy & Buvac, 1997) (Guha, 1991) (Buvač & Mason, 1993) (Shoham, 1991)

The work in this category is similar in general. Some slight differences are between individual research is highlighted in our analysis below,

(1) *Formal conceptualization:* All works in this group provide a clear conceptualization of context.

(2) *Formal model:* Only the works (Guha, 1991) and (Buvač & Mason, 1993) discuss modeling aspect, while others in this group do not discuss those aspects.

(3) *Formal representation:* No work in this group provide a formal representation and notation for context.

(4) *Expressiveness:*

   (a) Freedom of structure (hierarchical vs flat): No work has discussed the notion of "content of a context". In (Guha, 1991) a hierarchical structure is only implicitly implied. Therefore, we consider this requirement unmet.
   (b) Fixed Knowledge. (static information): This requirement is met by all the works in this group.
   (c) Inferred knowledge/rules. (dynamic information): Except Shoham (Shoham, 1991), all others in this category discuss inferencing contextual knowledge.

(5) *Extensibility:*
   (a) Supports heterogeneous data: Heterogeneity is supported by all of the works reviewed in this group except for (McCarthy, 1993; McCarthy & Buvac, 1997) where it is highly coupled with domain.
   (b) Innumerable: The theory in (Buvač & Mason, 1993) and (Shoham, 1991) is capable of modeling infinite data. The other two works do not consider this aspect in their study.
   (c) Detachable: None of the works reviewed consider detaching contextual knowledge from factual knowledge.

(6) *Manipulable:* Only the works (Guha, 1991) and (Shoham, 1991) discussed operations and manipulation of context content.

Context as Viewpoint. Giunchiglia and Attardi research group (Attardi & Simi, 1995) regarded context as a limited version of the world that comes from an "agent point of view" about the world. (Giunchiglia, 1993) is the first to thoroughly compare situations with contexts. Formally Giunchiglia defines context as a triple $C_i :< L_i, A_i, \delta_i >$, where L_i is the vocabulary of the context, A_i is the set of axioms that are true in the context, and δ_i is the set of rules governing the inference mechanism within the context. His definition of context as an agent view point supports the idea of having many agents looking at a database at the same time each of whom can see a different view of the same database. The work (Attardi & Simi, 1995) can be regarded as an extension of agents and multi-agents concept introduced in (Giunchiglia, 1993). However, instead of defining context as a triple, they introduced the notion of "view point" (*vp*). Their definition of view point is sets of sentences that represent the axioms of a theory. They are the first to introduce the concept of defining rules at a "metalevel", outside of the context (vp). Thus, the operations between different vp's are carried out with metalevel rules. As in all previous works, there is a clear absence of formal explicit structural model for context.

Critical Analysis of Viewpoint Notation

References in this category: (Giunchiglia, 1993) (Attardi & Simi, 1995)

Context was defined as a view point, perspective, or agent. Although they provided a conceptualization and formal model for an agent, they overlooked the formalism of representation. Below is an extensive evaluation for this notation.

(1) *Formal conceptualization:* All the works reviewed on viewpoints notation offer a clear conceptualization of the context/viewpoint entity.

(2) *Formal model:* Viewpoints provide precise definition of *vp*, relations, completeness and soundness.

(3) *Formal representation:* Viewpoint notation is not standard, and does not avoid implicit assumptions on relations between contexts/ontologies.

(4) *Expressiveness:*

  (a) Freedom of structure (hierarchical vs flat): The structuring ability of *vp* has not been discussed in any of the reviewed works. Hence, this criterion is not met.

  (b) Fixed Knowledge. (static information): All reviewed works meet this requirement.

  (c) Inferred knowledge/rules. (dynamic information): All reviewed works support the concept of inference.

(5) *Extensibility:*

  (a) Supports heterogeneous data: It is unclear whether or not different datatypes can be part of viewpoint structure.

(b) Innumerable: The work in (Giunchiglia, 1993) meets this requirement. However, the second work does not support it.

(c) Detachable: Both reviewed works do not support the detachment requirement.

(6) *Manipulable:* No operation has been defined for viewpoints.

Context as Situation. The pioneering work of Barwise (Barwise, 1989) considered a *situation* as context at a specific "time and location" in philosophical discourses. In his view (for the Study of Language et al., 1989), situation is context in the study of languages and philosophy. A formal structure for situation defined in Situation Theory (Seligman & Moss, 1997) has two components (Jon & John, 1983), called *infons* and *situation/role*. Infons are basic information units (discrete items of information) formalized as $\alpha =<< P, a_1, ..., a_n, \iota >>$, where P is an $n-place$ relation, $a_1, ..., a_n$ are arguments where a_j is the j_{th} argument in the relation P, and ι is a polarity $(0, 1)$ which determines whether or not relation P holds. The second component *situation/role* itself was not explicitly formally defined, but introduced as the *support* relationship with infons. That is, a situation s is supported by an infon α, denoted as $s \models \alpha$, and this means that the situation s is true in an infon α. For example, to describe the 'buying' situation between a buyer A and seller B using the infon notation, we say A is buying from B. According to (Akman & Surav, 1996), this can be modeled as $buy_s =<< buying, A, B, t, loc, 1 >>$ where, 'buying' is the relation, A,B, t(time) and loc(location), are arguments of the relation buying, and the right-most position holds '1' for true and '0' for false.

Later studies, such as (Akman & Surav, 1997; McCarthy, 1963) have touched upon situation theory for context modeling, without contributing to situation-based examples that can be used formally in context-aware application. More recent research (Dey, Abowd, & Salber, 2000; Henricksen & Indulska, 2006) moved away from situation theory, and called contexts modeled using Object-Role Modeling (ORM) (Halpin, 1998) as situations. They mapped this graph-based model to a relational model, and then mapped it into predicate logic as situation abstraction. In this approach, the representation of situation/context is $S(v_1, ..., v_n) : \varphi$, where S is the name of the situation, $v_1, ..., v_n$ is a set of variables defining the situation, and φ is a logical ground expression in which the free variables correspond to the set$\{v_1, ..., v_n\}$, to say that situation S is true with respect to the set of ground atoms φ.

The concept of situation involves only location and time, and hence it is only a partial view of the world. Also, it is still ambiguous in terms of structure and representation. Perhaps, for these reasons situation-based modeling was not influential in the history of context modeling. Also, as indicated in (Giunchiglia, 1993), "context is the most granular concept in the information structure", i.e. situation can be made up of several contexts, but one context cannot be a situation. Contexts can be looked at as the building block of the settings to form a situation. Therefore, having a formal representation for contexts is a step forward for formalising situations.

Critical Analysis of Situation Notation

References in this category: (Akman & Surav, 1997; for the Study of Language et al., 1989)

A detailed discussion on the "philosophical" and "technical" aspects of situation (Giunchiglia, 1993) (Mechkour, 2007) (Akman & Surav, 1997) concluded that "situation and context are two different things". Situation is a complete settings at specific "time" and "location". In one situation, there can be many contexts. For example, "being sick" can be a situation, while "age", "symptoms", "complaints" are different contextual information that describe or explain the situation. Our interest is to evaluate only the technical aspect of situation.

(1) *Formal conceptualization:* The work on situation notation is clearly conceptualized. Although it was not conceptualized as context, but it has the necessary ingredients of conceptualization such as definition, notation, and clear components.

(2) *Formal model:* Situations and infons and are not clearly modeled to bring out the relationship between them. All contextual information are captured in infon including the truth assignment of an infon at certain situation. So, the same infon cannot define for another situation?. It is not made clear whether we can have different instances of the same infon with different truth assignments.

(3) *Formal representation:* A formal representation is proposed for the "support" relationship between infon and situation. Infon has a fixed notation, however, situation and infons component structure are not provided in the reviewed works.

(4) *Expressiveness:*

   (a) Freedom of structure (hierarchical vs flat): There is no clear structure adopted in situation theory for situation, infons, or their relation taken together. Thus, this requirement is not met.

   (b) Fixed Knowledge. (static information): All infons represent only static information or fixed knowledge.

   (c) Inferred knowledge/rules. (dynamic information): Situation and infons have the ability to infer new situations. This requirement is met.

(5) *Extensibility:*

   (a) Supports heterogeneous data: situation theory meets this criterion.

   (b) Innumerable: situation theory supports infinite set of infons/information. Hence, it meets this criterion.

   (c) Detachable: infons can be used without a situation, but what would be assigned in the polarity (last) argument? how many arguments can an infon have?

(6) *Manipulable:* no operations are defined on infons/situations.

Context as Dimension and Attributes. Following the suggestion made by Shoham (Shoham, 1991), Schilit et al. (Schilit et al., 1994) introduced the notion of dimensions for context, and laid the foundation of many context-aware systems development. They conceptualized a context to have infinite number of dimensions that completely describe a setting, although in practice only finitely many dimensions can be introduced. Because Schilit et al. did not introduce any notation for context, only ad-hoc informal context representations were used in system development. As observed in (Perttunen et al., 2009), Schilit et al. is the most influential in the practical development of context-aware systems. Due to the fact that all systems have followed the same fundamental concept of Schilit et al. but use a variety of ad-hoc notations for context, we will not review all of them here. We refer the interested readers to the publications (Kofod-Petersen & Mikalsen, 2005) (Wang, Zhang, Gu, & Pung, 2004b) (Knappmeyer, Kiani, Frà, Moltchanov, & Baker, 2010) (Held, Buchholz, & Schill, 2002b).

Reddy and Gupta (Reddy & Gupta, 1995) followed the concepts introduced in (Shoham, 1991), (Schilit et al., 1994) and introduced set theory and lattice structures for context representation. With this structure they defined the binary operations union, and intersection for a collection of contexts. They are the first ones in the literature to introduce lattice as a formal structure to represent context dependency. Their focus was to introduce a hierarchy of contexts, capture the multilevel relationship between contexts and how contexts can be linked and compared with each other. Also, they introduced a method to query contexts, considering each context as a separate database. This technique is similar to the one introduced in (Giunchiglia, 1993) for a logical framework. However, Reddy and Gupta (Reddy & Gupta, 1995) considered context and data in one database as one entity that cannot be separated.

Wan (Wan, 2006) extended the concept of Schilit et al. (Schilit et al., 1994) to represent context with dimensions, where each dimension is associated with a set of typed attributes. For the sake of simplicity and practicality, only contexts with a finite set of dimensions were considered. Moreover, each dimension is associated with only one attribute from its domain. Wan (Wan, 2006) introduced operations on a set of contexts with this structure. Wan, Alagar, and Paquet's work (Wan, Alagar, & Paquet, 2005) is considered to be the first that has a theoretical basis and yet practical. From theoretical point of view, their work on extending Programming Language (IPL) Lucid with contexts to create the contextualized IPL Lucx, was highly influenced by the work of McCarthy, Guha, and Schilit. they demonstrated its utility in formally specifying timed systems, agent communication, and constraint programming. Alagar et al. were the first to introduce a context toolkit, comprising of operators to manipulate contexts, for dynamically generating context-dependent responses to system events in context-aware systems.

Critical Analysis of Dimension and Attribute Notation

References in this category: (Schilit et al., 1994) (Reddy & Gupta, 1995) (Wan, Alagar, & Paquet, 2005)

A key player to the development in this version of context formalism is the research pursued by Schilit et al. (Schilit et al., 1994). This group is the best match to our criteria of evaluation as shown below.

(1) *Formal conceptualization:* All the works reviewed here have conceptualized context as set of "dimension - attribute" pairs. Therefore, they all meet this criterion.

(2) *Formal model:* Only the work in (Reddy & Gupta, 1995) provide a precise definition of context, relations, completeness and soundness. The other two do not provide clear model.

(3) *Formal representation:* Only the work in (Wan, Alagar, & Paquet, 2005) offers a formal representation of the context content. It provides information about its dimension, attributes, and how different attribute types are modeled.

(4) *Expressiveness:*

  (a) Freedom of structure (hierarchical vs flat): The only work that considers hierarchies and flat structures among the reviewed works is (Reddy & Gupta, 1995). The remaining two works provide only flat structure of context.

  (b) Fixed Knowledge. (static information): All works in this group provide support for static information.

  (c) Inferred knowledge/rules. (dynamic information): Only the work in (Reddy & Gupta, 1995) considers inferring information about context.

(5) *Extensibility:*

  (a) Supports heterogeneous data: The studies in (Schilit et al., 1994) and (Wan, Alagar, & Paquet, 2005) do support heterogeneity. However,the study in (Reddy & Gupta, 1995) does not discuss the data involved at all, so we are uncertain about its support to multiple data types and domain-free information.

  (b) Innumerable: In general, the context notion " collection of dimension - attribute" does not restrict the sets to be finite. They only restrict it when it is coupled with an application to make its manipulation practically feasible.

  (c) Detachable: The context component is clearly detachable from any reasoning framework. Hence, this group meets this requirement.

(6) *Manipulable:* Only the works in (Reddy & Gupta, 1995) and (Wan, Alagar, & Paquet, 2005) discuss context operations.

2.3 Context-based Reasoning

In this section we review possible reasoning frameworks to build on top of the context component. We only look at framework level in this section regardless of how well the theory of context was developed and employed. This will help us to match make a complete theory of context with a complete framework of reasoning which allows dynamism in fetching and using context, yet make the context detachable and re-usable in other frameworks.

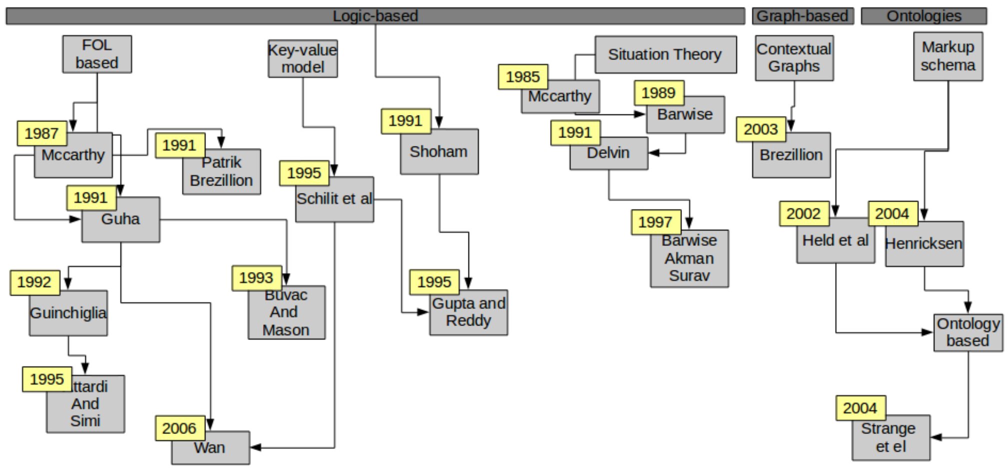


Figure 2.3: Concepts used to reason with context, full surveys can be found in (Serafini & Bouquet, 2004) (Akman & Surav, 1996) (Strang & Linnhoff-Popien, 2004) (Loyola, 2007) (Brézillon, 1999)

The goal is to investigate formal approaches that used context as a first class citizen in a framework of reasoning. The key methods investigated are depicted in Figure 2.3. In order to compare our work with as broad spectrum of methodologies as possible, after an extensive study and analysis of literature we have chosen those that we consider as most influential works in different formalization and reasoning categories.

The different conceptualization of context, introduced in Section 2.2, have led to different reasoning frameworks. We classify those frameworks into four main categories based on the used definition of context. The first category defines context as "an incomplete mathematical abstract entity that is rich and cannot be fully described by logic". The second category considers context as "a situation, which is a complete entity at a specific time and location". The third category defines context as "a set of dimensions and rules that construct the settings of an event". Finally, the forth category defines context as "an ontology".

Context-based reasoning framework is made up of the two main components: (1) context, and (2) framework of reasoning. Numerous survey papers (Serafini & Bouquet, 2004) (Akman & Surav, 1996) (Strang & Linnhoff-Popien, 2004) (Loyola, 2007) (Brézillon, 1999) have studied context/reasoning with emphasis on the level of formality, richness, expressiveness, soundness, consistency, and completeness. Motivated by different goals, we consider different criteria of evaluation inspired from the works in (Akman & Surav, 1996) (Alagar et al., 2014b) (Schilit et al., 1994) (Serafini & Bouquet, 2004).

(1) *Ability to Reuse:* In the related work, only some works made clear that context can be reused globally in different programs and applications. However, as mentioned in (Akman & Surav, 1996) and (Bazire & Brézillon, 2005), it is required that knowledge bases should be in one structure and it is reused by different programs and applications without affecting one another. Hence, the first criterion is that "it must be possible to use context with multiple programs with no side effect to the context being shared".

(2) *Ability for Multiple Reasoners to exist:* It should be possible to use rules to define and derive relationships and meta information about contexts.

(3) *Local Dynamism without Global Side Effect:* It must be possible to have dynamic context locally and static context globally. This criterion comes from the ability to reuse contexts by different programs at the same time. That is, infer new facts locally without affecting other programs that use the same knowledge.

(4) *Separating Data from Context:* It must be possible to distinguish context-related information (meta data) from other descriptions of facts/data in the system. By definition, context is the setting that makes a piece of information understandable. However, it is not the information itself.

(5) *Representation Simplicity:* It must be possible to express and construct contextual information easily in such a way that it does not increase complexity of reasoning and allow users to use it and comprehend it with ease.

(6) *Extensibility:* By definition (Johansson & Löfgren, 2009) extending a framework

means being able to build on top of a framework without affecting its original functionality. The extension of a framework should be conservative, in the sense that the extended framework can inherit all the features and retain the merits of the original framework.

Logic-based Contextual Reasoning Framework (First Category). This is the first and most influential work on context-based reasoning. It is the first attempt at realizing context entity in the field of computer science. In (McCarthy, 1993), McCarthy originally proposed the idea of incorporating context with logic. The main goal of McCarthy's research is to be able to transcend context to another context and solve the problem of generality by moving from general context to a specific context and vise versa (McCarthy & Buvac, 1997). Additionally, McCarthy introduced the $ist(C, F)$ function which indicates that context 'C' is true in formula 'F'. However, he did not introduce any reasoning with context or any other explicit operations that can deal with the context vocabularies and language. Motivations of his research are as follows:

- Allow simple axioms for common sense phenomena, i.e. axioms for something lifted to more specific thing.
- Treat/deal with particular circumstances which is at specific context. i.e. conversing at specific context may include words that are of different meaning in general context.
- AI systems never get stuck with concepts as they can transcend the context they are in. Thus, $ist(C, P)$ is self-asserted and it is within the context $ist(C', ist(C, P))$.

Guha (Guha, 1991) regarded context as a *microtheory* that partially explains the world. In his framework of reasoning every theory (category) can have infinite number of microtheories that describe it. The goals of Guha's research are extensions to those of McCarthy. In the proposed framework, he formalized the concepts of "*entering*" and "*exiting*" context. He formalized the meaning of meaningfulness of a microtheory in reasoning. The following are the goals of his research:

- Theories put together related axioms
- Integrate different contexts by integrating different microtheories.

- Microtheories create multiple models for a task.
- Lifting rules are of two types:
    - Compositional lifting: specify lifting rules for indirect predicates. For instance, when the predicates are of different arities.
    - Coreference lifting: no modifications needed to lift unless explicitly stated.

In summary, the concepts understood from his research are the following:

- Every context is a first class object
- $ist(C, F)$ and $presentIn(C, F)$ are the only functions that include a context argument.
- Every problem (program) has a specific context called the "problem solving context", to which every clause that does not specify any context explicitly is relevant.
- Formula F is meaningful in C iff:
    - F uses the terms in C correctly, e.g., predicates as predicates and constants as constants etc.
    - F uses the terms in C with the right arities according to the language of C.
- Entering a context 'C' means evaluating a formula 'F' in the context 'C' using the language and structure of 'C'. Once entered a context, the proposed idea uses the standard techniques of First Order Logic (FOL) and the functions ist/ispresent are not used inside the context (when context is entered).
- Exiting a context 'C' means to return to the problem solving context.
- Lifting rules are used to bring in rules and facts of two different contexts into the problem solving context to reason about it. Lifting rules assume that the same constants used in the two contexts to be combined mean the same thing. For example, John in context A is the same as John in context B. This assumption washes away the fear of conflicting facts.

In (Buvač & Mason, 1993), Buvač has extended Guha's theory using propositional logic. They introduced $ist(K, X)$ to classical propositional logic, which means sentence X is true in context K. Every context has its own vocabulary which are set of propositional atoms that are meaningful in that context. In the framework, they express context using partial truths.

In (Giunchiglia, 1993), the researcher follows the same concept of McCarthy's context with the notion of "agent point of view". He introduced bridging rules instead of lifting rules to link between different contexts and to transcend axioms from context to another. His definition of context as an agent view point supports the idea of having many agents looking at database at the same time each of whom can see a different view of the same database.

In (Attardi & Simi, 1995), view point is sets of sentences that represent the axioms of a theory. Their main relation of view point is defined as $in('A', vp)$ which means an axiom 'A' is true in view point vp. This is very similar to McCarthy's relation, i.e. $ist(C, P)$. The authors of this work were the first to introduce the concept of defining rules outside of the context (vp) at a "metalevel" layer. Thus, the operations of between different vp's are carried out with metalevel rules. These rules use vp to infer new knowledge.

In (Shoham, 1991), the researcher introduced a different way of reasoning with context in terms of notation and meaningfulness. He introduced P^C which denotes assertion (P) holds in context (C) which can be equivalent to $ist(C, P)$ relation introduced by McCarthy. For Shoham, every assertion is meaningful in every context but same assertion might have different truth values in different contexts. Thus, Shoham's logic formalism is different from the logical framework of other researchers. Also, he was first to define operations and relations between contexts within his framework. He defined the partial order relation ($\bullet \supset$) that is considered a weak partial ordering between contexts. That is, not every pair of context is comparable under it. He left the concept of upper and lower context as an open question, while in practice, he considered upper and lower as a result of the operations (X $\dot{\wedge}$ Y) and (X $\dot{\vee}$ Y) respectively. Hence, his framework is neither complete nor sound. He also defined non-monotonicity by introducing "negation" defined as context "$\dot{\neg} X$", which is not comparable to X under the relation ($\bullet \supset$).

Critical Analysis of Logic-based Reasoning Framework (First Category)

References in this category: (Guha, 1991; McCarthy, 1993) (Buvač & Mason, 1993) (Giunchiglia, 1993) (Attardi & Simi, 1995) (Shoham, 1991)

This category is the most influential in the study of context-based reasoning. Below is the evaluation based on our criteria of evaluation introduced earlier.

(1) *Ability to Reuse:* The ability to use context with multiple programs is not clear in most of the frameworks in this group. That is, it is not clear if context is a knowledge that can be used/reused by multiple agents/AI systems at the same time. Also, it is not mentioned explicitly that context is a knowledge base that consists of different contexts that can be used in different programs and by different reasoners. Frameworks in (Giunchiglia, 1993) and (Attardi & Simi, 1995) meet this requirement as they introduced the concept of view points to support the idea of being able to view the same data from different perspectives or views.

(2) *Ability for Multiple Reasoners to exist:* The ability to model relationship between different contexts had not been explicitly discussed. In most of the works in this category, the relations between contexts are referred to implicitly. For example, in (Guha, 1991; McCarthy, 1993), it is mentioned that lifting rules can be done if two contexts are referring to the same object. However, the two questions left open and unresolved are: (i) how to know that two contexts are referring to the same (or to different) entities?, and (ii) how to know two contexts are related, in the first place? Framework provided in (Attardi & Simi, 1995) in this group is an exclusion as authors refer to metalevel layer that sets rules and link contexts together

(3) *Local Dynamism without Global Side Effect:* Local dynamism and context reusing is a goal in (Guha, 1991; McCarthy, 1993). However, it is not stated clearly in the other frameworks in the group.

(4) *Separating Data from Context:* For all frameworks in this group, data and context are separated.

(5) *Representation Simplicity:* Simple structures are used in this group of frameworks. However, some of them like (Guha, 1991; McCarthy, 1993) and (Attardi & Simi,

1995) did not provided a structure of context. Thus, we could not tell if the structure of context is simple or not. They have only dealt with context as a "black box". As for the basis of the frameworks, they mostly exploit propositional and first order logic to represent context. Some of frameworks like (Attardi & Simi, 1995) considered higher order logic to reason with context which make it more complex

(6) *Extensibility:* The concept of extension is present in (Guha, 1991; McCarthy, 1993) and (Attardi & Simi, 1995), but not considered in the other works in this group.

Situation-based Reasoning (Second Category). This category considers context as "situation", defined as a mathematical theory of information. A situation is defined as limited portion of the world (over some location and time). Albeit the work in (Akman & Surav, 1997) and (Barwise, 1989), we could not find a framework of reasoning that use situations as context. Giunchiglia (Giunchiglia, 1993) and most other researchers in the literature define situation as a complete state of the world at a given time which explains why this type of extension was not influential in the history of context-based reasoning.

Critical Analysis of Situation-based Reasoning Framework (Second Category)

References in this category: (Barwise, 1989) (Akman & Surav, 1996)

Below is an evaluation of the situation-based reasoning frameworks included in the review.

(1) *Ability to Reuse:* The property of multiple programs/agents is not considered in this group of framework.

(2) *Ability for Multiple Reasoners to exist:* They did not consider separating rules from the contexts/situation. This is because their theory of situation has two parts "infons and situation" which contain data and rules of context.

(3) *Separating Data from Context:* Dynamism was only considered by (Akman & Surav, 1996) and not in the early work on situation theory.

(4) *Representation Simplicity:* Separating data from situation/context was not considered in this group of frameworks.

(5) *Extensibility:* There is no extension attempted in those works.

Context as Dimension and Attribute Reasoning (Third Category). In this category context is conceptualized as a set of dimensions and attributes. It was introduced in (Schilit et al., 1994). This category conceptualizes context as set of "dimension-attribute pairs" . The framework is rather simple and flexible. Detailed evaluation is provided below.

Critical Analysis of Dimension and Attribute Reasoning Framework (Third Category)

References in this category: (Schilit et al., 1994) (Reddy & Gupta, 1995) (Wan, Alagar, & Paquet, 2005)

(1) *Ability to Reuse:* The property of multiple programs/agents is not considered in this group of framework.

(2) *Ability for Multiple Reasoners to exist:* The works reviewed did not consider separating rules from the contexts. The study in (Schilit et al., 1994) did not even consider inferencing altogether.

(3) *Local Dynamism without Global Side Effect:* Dynamism was only considered in (Schilit et al., 1994), but only for specific practical problems that have well defined context before the run of the application. That is, they have mentioned that one of the goals of their system is to create contexts at runtime. However, they did not say if multiple programs running on this context knowledge will be affected by this dynamism. Thus local dynamism is not clearly discussed.

(4) *Separating Data from Context:* Separating data from context was only considered in (Reddy & Gupta, 1995), though it was mentioned only implicitly.

(5) *Representation Simplicity:* Works reviewed here adopted simple syntax. They have well-defined structure for their frameworks. Hence, this requirement is met.

(6) *Extensibility:* The extension aspect is demonstrated in (Wan, Alagar, & Paquet, 2005), where Lucid Intentional Logic Programming Language extension is discussed in detail.

Ontology-based Reasoning(Forth Category). In this category a context is viewed as an ontology (Strang et al., 2003), as set of concepts and attributes. It also considers every context/ontology have its own inferencing rules that can infer new facts within the ontology. Thus, every ontology has its own inferencing engine. The focus of this approach is the formal capturing of the interdependence of contextual information. Also, this approach considers every ontology as both the context (meta information) and the associated data. To remedy this, some researchers have introduced the concept of contextualization of ontologies (Bouquet, Giunchiglia, van Harmelen, Serafini, & Stuckenschmidt, 2004) to capture the contextual information (meta info) from the ontology to be able to re-use it or link it to another ontology. Below is an evaluation of this research.

Critical Analysis of Ontology-based Reasoning Framework (Forth Category)

References in this category: (Strang et al., 2003)

(1) *Ability to Reuse:* XML representation is used to represent context. However, no formal representation of context was proposed.

(2) *Ability for Multiple Reasoners to exist:* Every context has its own set of inferencing rules, which means rules and context form one entity.

(3) *Local Dynamism without Global Side Effect:* Ontologies consider dynamism in the sense that nodes are created, modified and changed at runtime of applications. However, when more than one program are using the same ontology they are all affected with this change. It is more like single database to which applications are connected. In other words, ontology grows globally. Meaning, if a program is using an ontology then all other programs will be aware of the changes (improvement) in the knowledge of that ontology.

(4) *Separating Data from Context:* Data is included within the ontology. That is, context models that are based on ontology may not be considered a meta information because it includes a huge entity that contains everything, i.e. data, rules and meta information.

(5) *Representation Simplicity:* Due to the way ontologies are structured, complex inferencing engines will be required to infer knowledge.

(6) *Extensibility:* no clear extension is manifested.

2.3.1 Summary of Context and Context-based Reasoning Analysis

In this section, we summarize our observations on the previous methods, from the context and framework of reasoning perspectives. Table 2.1 comprehensively summarizes all the reviewed works on context modeling and formalism. Table 2.2 compares the works from reasoning framework level perspective. At framework level, we find that McCarthy and Guha's work are the most influential since it was the pioneering work on context. The works that extended their intuition have shared the same features and properties. Figure 2.3, shows the relationship between different works in the previous endeavors. They indeed induced a measurable step towards building a formal framework to reason about contexts. However, the technical details were not developed. A well defined framework that uses context and its operations in a seamless manner remained missing in all the reviewed work.

#	Category	Research	Formalism			Expressiveness			Extensibility			Manipulable
			FC	FM	FR	FrS	FxK	InK	SHD	Inn	Det	
1	Intention	A	×	×	×	×	×	×	×	×	×	×
2	Explanation	B	×	×	×	×	×	×	×	×	×	×
3	Formal Concept Analysis	C	✓	×	×	✓	✓	×	✓	✓	✓	✓
4	Ontology	D	×	×	×	✓	✓	×	×	✓	×	×
5	Ontology	E	×	×	×	✓	✓	✓	×	✓	×	×
6	Ontology	F	×	×	×	✓	✓	✓	×	✓	×	×
7	Ontology	G	×	×	×	✓	✓	✓	×	✓	×	×
8	Logical. Event	H	×	×	✓	×	✓	×	✓	✓	×	✓
9	Logical. Math. Entity	I	✓	×	×	×	✓	✓	×	×	×	×
10	Logical. Math. Entity	J	✓	✓	×	?	✓	✓	✓	×	×	✓
11	Logical. Math. Entity	K	✓	✓	×	×	✓	✓	✓	✓	×	×
12	Logical. Math. Entity	L	✓	×	×	✓	✓	?	✓	✓	×	✓
13	Logical. Viewpoint	M	✓	✓	×	×	✓	✓	?	✓	✓	×
14	Logical. Viewpoint	N	✓	✓	×	×	✓	✓	?	✓	×	×
15	Logical. Situation	O	✓	✓	×	×	✓	✓	✓	✓	×	×
16	Dimension and Attribute	P	✓	×	×	×	✓	?	✓	?	✓	×
17	Dimension and Attribute	R	×	✓	×	✓	✓	✓	?	✓	✓	✓
18	Dimension and Attribute	S	✓	×	✓	×	✓	×	✓	✓	✓	✓

Table 2.1: Summary of evaluation of all research reviewed in sections Informal Notations 2.2.1 and Formal Notations 2.2.2

*FC: Formal Conceptualization, FM: Formal Modeling, FR: Formal Representation, FrS:Freedom of Structure, FxK:Fixed Knowledge,InK: Inferred Knowledge, SHD: Support Heterogeneous Data, Inn:Innumerable, Det: Detachable

A: (Costa et al., 2012) (Liu et al., 2015), B: (Kass et al., 1986; Leake, 2014; Mooney & DeJong, 1985; Norvig, 1983), C: (Alsaig et al., 2015), D: (Gao & Dong, 2017) , E: (Wang et al., 2004a), F: (Ejigu et al., 2007), G: (Gu et al., 2004; Lee et al., 2007; Shehzad et al., 2004), H: (Mueller, 2014), I: (McCarthy, 1993; McCarthy & Buvac, 1997), J: (Guha, 1991) , K: (Buvač & Mason, 1993), L: (Shoham, 1991), M: (Giunchiglia, 1993), N: (Attardi & Simi, 1995), O: (Akman & Surav, 1997; for the Study of Language et al., 1989), P: (Schilit et al., 1994), R: (Reddy & Gupta, 1995) , S: (Wan, Alagar, & Paquet, 2005)

#	Category	Research	Multiple Programs	Separate Rules	Local Dynamism	Separate Data	Simplicity	Extension
				First Category				
[a]	Logic-based	A	?	?	✓	✓	×	✓
[b]	Logic-based	B	×	×	?	✓	✓	×
[c]	Logic-based	C	✓	×	?	✓	×	×
[d]	Logic-based	D	✓	✓	?	✓	?	✓
[e]	Logic-based	E	?	?	?	✓	✓	×
				Second Category				
[i]	Situation	F	×	×	?	×	?	?
[j]	Situation	G	✓	×	✓	×	×	×
				Third Category				
[f]	Dimension & Attribute	H	×	×	?	×	✓	×
[g]	Dimension & Attribute	I	?	×	×	?	✓	×
[h]	Dimension & Attribute	J	×	✓	✓	×	✓	×
				Forth Category				
[k]	Ontology	K	×	×	✓	×	×	?

Table 2.2: Framework level evaluation
* A: (Guha, 1991) (McCarthy, 1993), B: (Buvač & Mason, 1993), C: (Giunchiglia, 1993), D: (Attardi & Simi, 1995), E: (Shoham, 1991), F: (Barwise, 1989), G: (Akman & Surav, 1996) H: (Schilit et al., 1994), I: (Reddy & Gupta, 1995), J: (Wan, Alagar, & Paquet, 2005), K: (Strang et al., 2003)

Chapter 3

Research Methodology of *Contelog* Framework

Having explained what the thesis is about and why it is significant in Chapter 1, a wide range of context concepts was surveyed and compared in Chapter 2. From this discussion, the key features of our research methodology in designing *Contelog* framework are identified. In this chapter, the characteristics that govern how we design and develop *Contelog* framework are explained. The term “framework” refers to the environment in which the following four major components exist, and interact to achieve contextual programming and contextual reasoning. These are

- the *c-programming* component shown in Figure 3.1, and
- the *user interface* component, the *query processing* component, and the *evaluation component* shown in Figure 3.2.

In Section 3.1, the key features of the *c-program* components shown in Figure 3.1 are discussed. In Section 3.2 the key features of the rest of the components are discussed.

3.1 c-Program Components

A *Contelog* program, as shown in Figure 3.1, has the following components:

- *c-facts* is a set of regular/contextual facts,

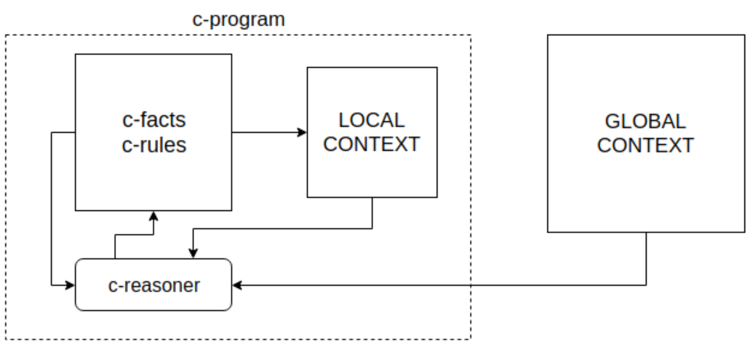


Figure 3.1: Components of *c-programs*

- *c-rules* is a set of regular/contextual rules,
- *LOCAL CONTEXT* is a subset of contexts imported from the set *GLOBAL CONTEXT*, the complete lattice of contexts of interest for an application domain, and
- *c-reasoner* is the reasoning engine that uses c-facts, c-rules, and the set "LOCAL CONTEXT".

A *Contelog* program with these components structured according to the syntax, explained in Chapter 5, is called a *c-program* . As a result of the summary of extensive comparisons shown in Table 2.1 and Table 2.2, the key characteristics for context representation, and the key characteristics for an extension of Datalog were chosen. Those features are emphasized in the following sections in order to concretely position the thesis with respect to the literature.

3.1.1 Key Characteristics for Context

Context notation, defined in Chapter 4, is a finite set of dimension-attribute_set pairs. Each dimension is associated with a type, and the set of attributes associated with the dimension is a subset of the typed domain. This notation extends the attribute-dimension notation reviewed in section 2.2.2 in a rich manner. Using this notation, the two contexts necessary for reasoning in the "Building Locator Example" are written as

$$c_1 = \{from : [east], to : [right]\} \qquad c_2 = \{from : [west], to : [left]\}$$

This choice of notation is best match to the criteria of evaluation shown Table 2.1. From this summary, it is clear that context modeling and structuring are still "a work in progress".

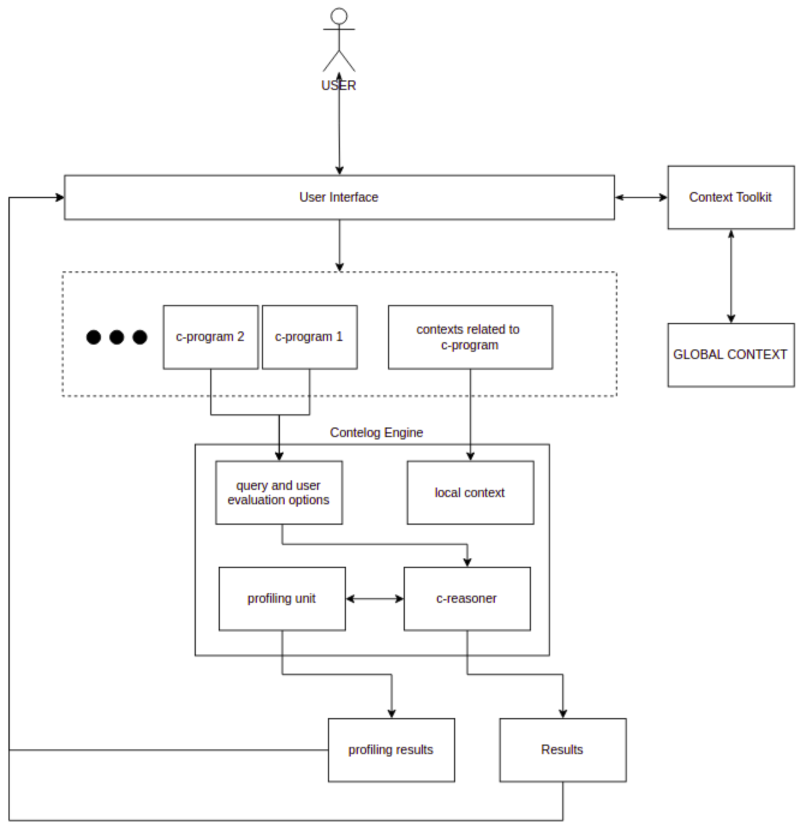


Figure 3.2: Highlevel illustration for *Contelog* Framework components

Considering the theoretical and empirical needs for contextual reasoning, features of context representation are as follows:

(1) *Conceptualization and Modeling:* We decided to go for the same concept introduced in (Shoham, 1991), that context is complete entity at the time of reasoning, although it may never be complete generally. No model can adequately fulfill the conceptualization introduced in this thesis, unless completeness is formally enforced. Hence, context calculus is introduced and complete lattice of contexts is defined to introduce closure of the contextual reasoning system.

(2) *Representation:* Our relational representation is much richer than the functional model introduced in (Wan, 2006; Wan, Alagar, & Paquet, 2005).

(3) *Expressiveness:* Structural simplicity must be given high importance. Inferring knowledge from contexts should be done in a separate layer. Therefore, we only commit

to freedom of structure and fixed-knowledge criteria. No previous work has met this expressiveness requirement.

(4) *Extensibility:* Our context model supports heterogeneity, innumerability, and most importantly detach-ability (separation of concerns).

(5) *Manipulability:* Context operations defined in this thesis manipulate contexts separately from the KB layer. The set of contexts is closed with respect to these operations. Consequently, all contexts are well-defined in the domain of reasoning.

(6) *Reasoning:* The relational syntax of context blends well with our choice of Datalog extension. The closure property of context collection, in "GLOBAL CONTEXT", enforces closure of contextual reasoning when the c-reasoner uses "LOCAL CONTEXT".

With the above features, context become a "first class citizen" in *c-programs* . The context component "LOCAL CONTEXT", imported from "GLOBAL CONTEXT", is independent of the rest of the components, yet it is fully integrated with the *c-reasoner*. The relational syntax of context can be represented as a predicate in a *c-program* such that a dimension in context syntax is a predicate name in *c-program* , and the attributes of that dimension are the attributes of the predicate in *c-program* , annotated with '@' notation. A *c-program* can pass context as an argument, and deal with its operations as a first class citizen incorporated in its syntax and semantics. Thus, many *c-programs* can deal with the same context component and propose their own version of contexts internally in the program without affecting other programs that using the same set of contexts.

3.1.2 Key Characteristics of c-facts and c-rules

After reviewing many methods and comparing their merits summarized in table 2.2 for context-based programming, we identified Datalog (Ceri et al., 1989) as the best match. Datalog has a declarative semantics, simple syntax, and high volume of literature. It has been well-studied in the field of deductive database and programming languages, hence it is well-developed. More importantly, it is sound and complete. It is believed that *simplicity of syntax*, and *theoretical soundness* are two main features of importance for contextual reasoning. The conservative extension of Datalog syntax, briefly introduced below, serve to formalize the set of *c-facts*, and *c-rules* of a *c-program* .

Features of Syntax A *c-fact* is a proposition, having the same syntax as in Datalog. For the "Building Locator Example" the two *c-facts*, written as p(john,east), p(rose,west), are listed in a section following the list of contexts in a *c-program* .

A *c-program* may include regular rules, as in the Datalog syntax

$$L_0 \text{ :- } L_1, L_2, ..., L_n,$$

where each L_i is at least a unary predicate. In addition, a *c-program* will include contextual rules (*c-rules*). In each *c-rule* one or more predicates will be bound to a context. The notation that is used for a context-based predicate is $p(\bar{X})@C$, where $\bar{X}$ is a list of predicate arguments and C is a context. The interpretation of $p(\bar{X})@C$ is "p is true in context C", which is precisely the assertion $ist(C, P)$ of McCarthy. This extension of Datalog rules has two advantages. These are

- With $ist(C, P)$ notation, it is easy to confuse with other normal predicates. Also, since one argument is context and the other is predicate, it is easy to mix those two arguments of different types such as putting one in place of the other. So, our notation avoids ambiguity.
- Our notation improves readability, because with @ notation it is more natural to read a predicate P 'at' context C.

Hence, regardless of the semantics, a full program may look as follows

c- program structure

```
#Contexts
$c_1 = \{dim_1 : [att_1, att_2]\}$
$c_2 = \{dim_1 : [att_3, att_4], dim_2 : [att_1]\}$
#Facts
$q(a, b)@c_2$.
$p(a, b)@c_1$.
#Rules
$p(X, Y) : -l(X, Y)$.
$p(X, Y) : -q(X, Y)@C$.
$r(X, Y)@C : -g(X), m(Y)@C$.
```

3.1.3 Key Features of Semantics For c-reasoner

Model theories in mathematical logic define the semantics of formal systems. That is, it assigns meanings/interpretations to the symbols used in the syntax of the logic programs. However, in general logic, it is complicated as it requires the use of modern algebra as stated in (Ceri et al., 1989). Therefore, in *Contelog* , the model theory of Horn Clause (Apt & Emden, 1982; Lloyd, 1987; Van Emden & Kowalski, 1976) are used, which makes *Contelog* very easy to be described in model theory. In other words, the study of semantics of *Contelog* programs is restricted to *Herbrand* structures. By this, the results of a program can be known in purely model-theoretic approach, without the need to provide a proof for it. For the declarative semantics feature to exist, the importance of the completeness feature in context theory is emphasized, i.e. ‘closed-world assumption’. As a result, if *Contelog* has a declarative semantics feature, it can be described as ‘sound’ and ‘complete’.

With the above explanation, the “Building Locator” program given below can be easily understood. A full discussion of context representation and *Contelog* syntactic and semantic extensions are given in Chapters 4 and 5.

The Building Locator program in Contelog

```
#Contexts
$c_1 = \{from : [east], to : [right]\}$.
$c_2 = \{from : [west], to : [left]\}$.
#Facts
$p$(john,east).
$p$(rose,west).
#Rules
$p(X,Y)@C \leftarrow p(X,Y), from(Y)@C$.
$side(X,Z)@C \leftarrow p(X,Y)@C, to(Z)@C$.
```

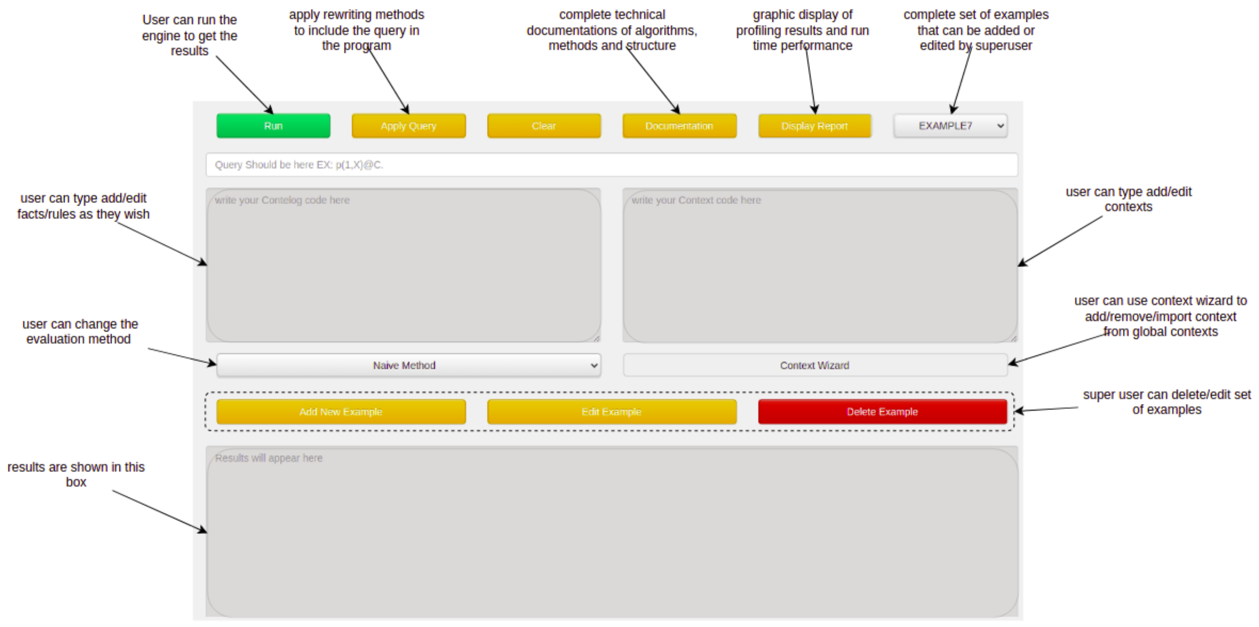

User can run the engine to get the results
apply rewriting methods to include the query in the program
complete technical documentations of algorithms, methods and structure
graphic display of profiling results and run time performance
complete set of examples that can be added or edited by superuser
Run
Apply Query
Clear
Documentation
Display Report
EXAMPLE7
Query Should be here EX: p(1,X)@C.
user can type add/edit facts/rules as they wish
write your Contelog code here
write your Context code here
user can type add/edit contexts
user can change the evaluation method
Naive Method
Context Wizard
user can use context wizard to add/remove/import context from global contexts
Add New Example
Edit Example
Delete Example
super user can delete/edit set of examples
results are shown in this box
Results will appear here

Figure 3.3: Overview of User Interface

3.2 User Interface, Query Processing and Evaluation Components

In this section we explain the key features of *user interface* component, the *query processing* component, and the *evaluation component*, and motivate how they collectively interact with each other to achieve programming and reasoning in *Contelog* framework. Figure 3.2 shows the interactions among these four components. Thus, *Contelog* framework will facilitate a user in constructing contexts, composing and executing *c-programs* , getting results in desired formats, inputting a query and obtaining the reasoned facts that match the query, and getting information on the performance of all executions.

3.2.1 User Interface Component

The user interface for *Contelog* is the communication tool between the user and all other components of the framework as shown in Figure 3.3. Below, the main functions of the interface are explained.

(1) Run: it is to run the engine based on the evaluation method chosen in ‘Evaluation Method Options’.

(2) Apply Query: this button is to apply the query and incorporate it in the rewriting of context/program based on our novel 'Magic Context' rewriting technique developed for *Contelog* . This technique is explained in details in Chapter 6.

(3) Documentation: this is a complete technical documentation for the engine functions, algorithms, and structure. More information regarding documentation is provided in (Alsaig, 2017).

(4) Display Report: it is to display the results of the profiling engine deployed for *Contelog* programs. Through this tool, the user can compare five different runs of the engine in terms of tokenizing and reasoning runtime performance. More details on this is provided in (Alsaig, 2017).

(5) Book of Examples: this is a drop down list that contains all ready-made examples in the library of examples that can be added/edited or removed by super users. All examples are provided in the book of examples in Appendix A

(6) Context and Code Boxes: those two text areas are to edit/add context or *c-programs* with respect to the syntax of each.

(7) Evaluation Method Option: this is a drop down list that gives the user an option of choosing Naive or Semi-naive evaluation techniques. Both techniques are explained verbosely in Chapter 6.

(8) Context Wizard (Toolkit): this toolkit is for the user who are not familiar with the syntax of the context and wants to add/manipulate it using GUI.

(9) Results: this is a text area to show all results or errors in syntax or semantics. More on *Contelog* syntax and semantics is provided in Chapter 5

3.2.2 Query Processing Component

The model theory proposed in Section 3.1.3 allows the presence of a proof-theoretic bottom-up approach to calculate all possible results of a *Contelog* program, and guarantee termination with a *fix-point*. Bottom-up (from facts to query) and Top-down (from query

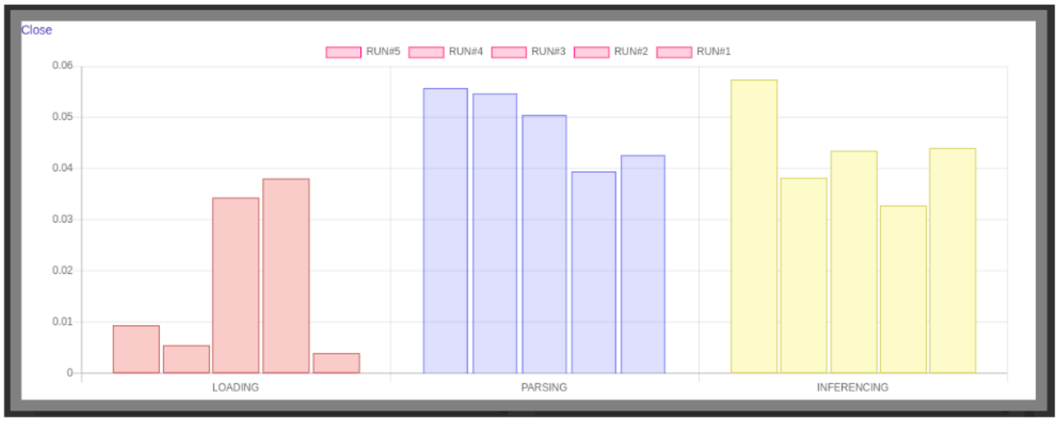


Figure 3.4: Snapshot of the display reports shown in *Contelog* Framework

to facts) are two possible approaches to query processing. In this thesis, the bottom-up approach is the basis for implementing the query processing component. Top-down approach is left for potential future improvement of *Contelog* framework.

The following three query processing techniques are all implemented in *Contelog* System, and will be discussed in detail in Chapter 6.

(1) *Naive Approach:* Starting from facts, rules are used repeatedly to generate new facts until no new facts are generated. The use query will filter the set of results.

(2) *Semi-Naive Approach:* This method starts from facts and use rules to generate new facts, while keeping track of the used facts. This step is repeated only if generated facts are "new", in order not to accumulate already inferred facts. Then, query filter is used to get the set of results.

(3) *Magic Context:* Recognizing the given query as “the context” for inferring all relevant facts, we construct a *Magic Context* from the user query, and write “magic rules” and “dependency rules”, and finally rewrite the program in accordance with the new set of magic rules and context. Last, based on the user’s choice the *Semi-Naive or Naive Approach* is used on the rewritten program.

3.2.3 Performance Evaluation Component

In addition to the theoretical and empirical study of performance introduced in Chapter 8, this component is a profiling engine that is used at each sub-component of *Contelog* engine to measure the runtime performance of the methods used. This includes measuring the runtime of Loading the files into the system, Tokenizing and Scannning process, and finally the Inferencing process. For each process the time is internally being measured with this profiling component and those measurements are displayed as a bar-chart graphic report at the end of the runs as shown in Figure 3.4. It is used by the user to compare the performance of five consecutive runs. This component is useful for comparisons between the optimization and query techniques used for *Contelog* . The advantage of having a separate profiling for each stage, i.e. loading, tokenizing, and reasoning, is to know where exactly each method is preforming better than the other.

3.3 Merits of the *Contelog* Framework Design

The design of *Contelog* framework is motivated by the basic design principles *simplicity*, *formality*, *generality*, *extensibility*, and *reuse*. Simplicity is in the *economy* of representation that allows expressibility and portability while relieving ambiguity. Formality is necessary for removing syntactic and semantic ambiguities and for sound integration with declarative logic for reasoning. Generality has the virtue of supporting variety of typing of meta information captured in contexts. Extensibility is achieved through loose coupling of context world with reasoning world. This separation of concern leads to both reasoning in many context worlds and conducting different reasoning within one context world. Reuse of contexts and the rules are byproducts of loose coupling. One world is not aware of the other, yet will function when they are synchronized.

Contelog framework is designed for use either as a stand alone reasoner or work synchronously with a context-aware system. As a stand alone reasoner, it can be part of Big Data (BD) analysis in applications such as medical diagnosis (Clancey, 1983; Lamperti & Zanella, 2003), clinical decision support (Sordo, Tokachichu, Vitale, Maviglia, & Rocha, 2017), and advanced manufacturing (R.Karni & A.Gal-Tzur, 1990; S.C.Feng, W.Z.Bernstein, T.Hedberg, , & Feeney, 2017). As opposed to general rule-based expert systems (Buchanan et al., 1984) used in BD, by constraining the domain of reasoning to specific contexts, data inconsistency is relieved, and reasoning efficiency is enhanced. *Contelog* can be linked with a context-aware system which arises in many emerging context-aware applications such as Smart City, where both context-awareness and inference of knowledge are pivotal for making strategic decisions. Context-aware computing refers to the ability of the system to gather information about its external environment and adapt itself accordingly. So, it becomes essential to reason about *safety* aspects of adaptation in different contexts. Recognizing that adaptation which is system (requirements) specific, and reasoning with information that is specific to contextual information, are complementary issues, the *Contelog* platform might be combined to advantage with any adaptation platform in a context-aware system. In order to verify that the system adaptations are correct with respect to its critical requirements, it is necessary to have a feed-back loop from the adaptation unit to *Contelog* . A case study on building context-aware system using *Contelog* is provided in Chapter 7. Extensions of *Contelog* necessary for further synchronizing with context-aware systems are explained as part of for future studies in Chapter 9.

Chapter 4

Context Theory

A comprehensive literature survey on context was given in chapter 2. The basis of the proposed thesis structure was given in chapter 3. Interested readers may refer to (Alsaig, Alagar, & Shiri, 2019b).In this chapter, we pick up those notions, generalize and enrich to formally conceptualize the notion of context in three tiers. In Tier 1 *Context Schema* is defined, in Tier 2 *Typed Context Schemas* are derived, and in Tier 3 *Contexts* are formalized as a set of families of *Context Instances* generated from typed schemas. In our discussion, we may refer to context instances as context. For all the three tiers a uniform representation is used, under set theoretical setting. The operations defined in Tier 1 are inherited in successive tiers, and in particular enriched in Tier 3. For applications on our approach readers may refer to (Alsaig, Alagar, & Shiri, 2019a). Our approach, because of its generality and simplicity, has the potential for generating different families of contexts for different applications within an application domain. The ability to generate different typed context families promote a strict development and reuse of *context toolkit*. The merits of context toolkit layer, shown in Figure 4.1, are the following:

- The toolkit meets the criteria of evaluation discussed in Section 2.2.
- It is both “open” and “closed” unit. It is closed in the sense that it is a stand-alone unit within which contexts are defined, manipulated and managed. It is open in the sense that it can be integrated with any reasoning framework. That is, the development of the theory does not depend on or limited to a specific reasoning framework.
- *Contelog* users and developers of any context-aware application system can create the

family of contexts they want without concern on the underlying context formalism. The user interface helps them to create context schemas, types, typed schemas and context instances.

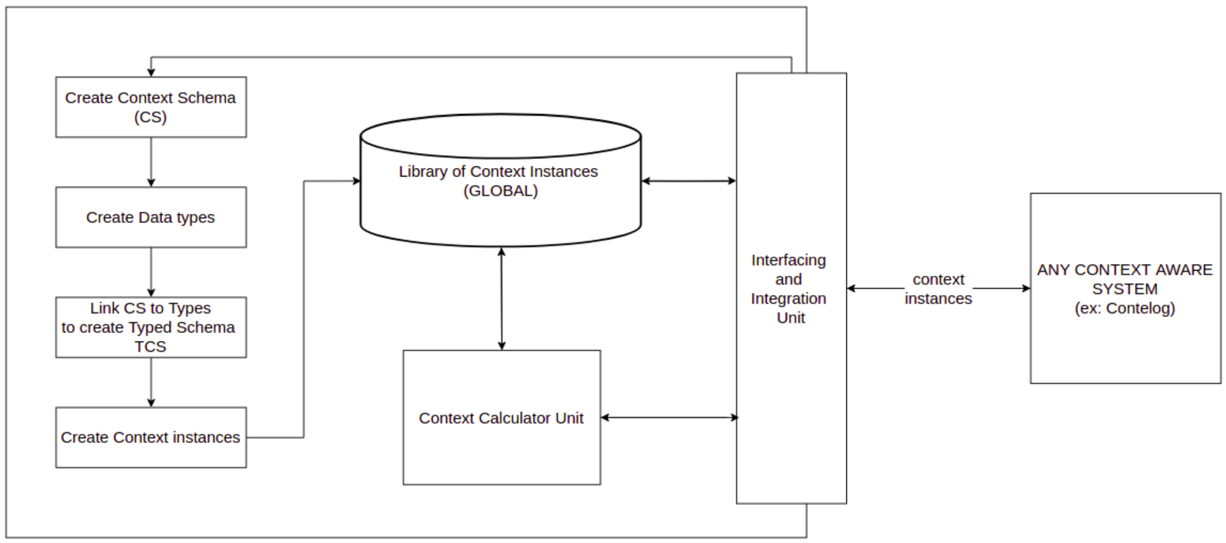

Create Context Schema (CS)
Create Data types
Link CS to Types to create Typed Schema TCS
Create Context instances
Library of Context Instances (GLOBAL)
Context Calculator Unit
Interfacing and Integration Unit
context instances
ANY CONTEXT AWARE SYSTEM (ex: Contelog)

Figure 4.1: Context Toolkit Layer

4.1 Tier 1: Context Schema Representation and Calculus

Context schema (CS) is abstract, and its generality gives flexibility for a practical system designer to choose attributes to be associated with a dimension. Once we fix the (non-empty) set of dimensions $\mathbb{D}$ and the (non-empty) set of attributes $\mathbb{A}$, the set $S_{\mathscr{C}}(\mathbb{D}, \mathbb{A})$ of all CSs is fixed.

Definition 1. *Formally, a context schema (CS) $\mathscr{C}$ over the two pair $\mathbb{D}, \mathbb{A}$ is a set of pairs defined as*

$$\mathscr{C} = \{d : A_d \mid d \in \mathbb{D} \wedge A_d \subseteq \mathbb{A}\} \quad \blacksquare$$

Example 1. *A context schema $\mathscr{C}_{conf}$ for conference that specifies its date, time and location can be defined as follows:*

$$\mathbb{D} = \{Date, Time, Location\}$$

$$\mathbb{A} = \{a_1, a_2, a_3, a_4\}$$

$$\mathscr{C}_{conf} = \{Date : \{a_1, a_2\}, Time : \{a_3\}, Location : \{a_4\}\}$$

In order to access information of an already constructed context schema, we have defined two functions. The "DIM" function is to extract the set of dimensions in a context schema, and the "ATT" function is to extract the set of attributes associated with a dimension in a context schema. Formally,

$$DIM : S_{\mathscr{C}} \to \mathbb{P}(\mathbb{D}) \implies DIM(\mathscr{C}) = \{d| < d, A_d > \in \mathscr{C}\}$$

$$ATT : \mathbb{D} \times S_{\mathscr{C}} \to \mathbb{P}(\mathbb{A}) \implies ATT(d, \mathscr{C}) = A_d$$

Example 2. *Applying the context schema functions on the "conference context schema" introduced in Example 1, we have* $DIM(\mathscr{C}_{conf}) = \{Date, Time, Location\}$*.,* $ATT(Date, \mathscr{C}_{conf}) = \{a_1, a_2\}$*,* $ATT(Time, \mathscr{C}_{conf}) = \{a_3\}$*, and* $ATT(Location, \mathscr{C}_{conf}) = \{a_4\}$*.*

For the sake of completeness we include "Null Context Schema" and "Full Context Schema". If $DIM(\mathscr{C}) = \phi$, $A_d = \phi$ we get the Null Context Schema $\mathscr{C}_{\phi}$. If $DIM(\mathscr{C}) = \mathbb{D}$, $A_d = \mathbb{A}$ we get "Full Context Schema".

4.1.1 Containment Relationship

McCarthy (McCarthy, 1993) postulated that "every context is contained in an outer context". Also, a context can contain an inner context. However, no formal definition for "containment relationship" exists, because of lack of formal representation. Using our context schema formalism we can formally define containment relationship over the set $S_{\mathscr{C}}(\mathbb{D}, \mathbb{A})$.

Definition 2. *Containment Relationship. Let* $\mathscr{C} = \{< d, A_d > | d \in \mathbb{D} \ \wedge \ A_d \subseteq \mathbb{A}\}$ *and* $\mathscr{C}' = \{< d', A'_d > | d' \in \mathbb{D} \ \wedge \ A'_d \subseteq \mathbb{A}\}$ *be two Context Schemas in* $S_{\mathscr{C}}(\mathbb{D}, \mathbb{A})$*. We say that* $\mathscr{C}'$ *is Contained in* $\mathscr{C}$*, denoted as* $\mathscr{C}' \sqsubseteq \mathscr{C}$*, iff:*

(1) $DIM(\mathscr{C}') \subseteq DIM(\mathscr{C})$

(2) $\forall \ d' \in \mathscr{C}' ATT(d', \mathscr{C}') \subseteq ATT(d', \mathscr{C})$ ■

Essentially, a context schema $\mathscr{C}'$ is contained in another context schema $\mathscr{C}$ if all dimensions and their associated attributes that are included in $\mathscr{C}$ are also included in $\mathscr{C}'$. The relation $\sqsubseteq$ on the set $S_{\mathscr{C}}(\mathbb{D}, \mathbb{A})$ is a partial order because of the following properties.

- *reflexive:* $\mathscr{C} \sqsubseteq \mathscr{C}$ is true.
- *non-symmetric:* If $\mathscr{C}' \sqsubseteq \mathscr{C}$ holds then $\mathscr{C} \sqsubseteq \mathscr{C}'$ is false, unless .
- *transitive:* If $\mathscr{C}' \sqsubseteq \mathscr{C}$ and $\mathscr{C}'' \sqsubseteq \mathscr{C}'$ are both true then $\mathscr{C}'' \sqsubseteq \mathscr{C}$ is true.

The set $\{S_{\mathscr{C}}(\mathbb{D}, \mathbb{A}), \sqsubseteq\}$ is a *partially ordered set* (POD).

Example 3. *An example of containment would be another conference context schema $\mathscr{C}'_{conf}$ with the following dimensions and attributes:*

$$\mathscr{C}'_{conf} = \{Date : \{a_1\}, Time : \{a3\}\}$$

Because $DIM(\mathscr{C}'_{conf}) \subseteq DIM(\mathscr{C}_{conf})$ and $A'_d \subseteq A_d$ forall $d \in DIM(\mathscr{C}'_{conf})$ we can say that $\mathscr{C}'_{conf} \sqsubseteq \mathscr{C}_{conf}$.

4.1.2 Operations and Calculus

Not every pair of schemas in the set $\{S_{\mathscr{C}}(D, A), \sqsubseteq\}$ is related. In order to relate and deal with every pair of schemas we are motivated to introduce the two operators "join" ($\oplus$) and "meet" ($\odot$). These two operators go hand in hand with the containment relationship. By "joining" two context schemas we get the "smallest" context schema that contains those two context schemas. By computing the meet of two schemas we get the "largest" context schema that is contained in those two schemas. These operations, when implemented as part of context tool kit in an application will enable "exporting knowledge and reasoning" across contexts. We define three binary operations on the set $S_{\mathscr{C}}(\mathbb{D}, \mathbb{A})$. Let $\mathscr{C}$, $\mathscr{C}' \in S_{\mathscr{C}}(\mathbb{D}, \mathbb{A})$.

Definition 3. *Equality (=)*

$$=: S_{\mathscr{C}}(\mathbb{D}, \mathbb{A}) \times S_{\mathscr{C}}(\mathbb{D}, \mathbb{A}) \to Bool$$

Using the "infix notation",

$$\mathscr{C} = \mathscr{C}', \quad if \ and \ only \ if \quad \mathscr{C} \sqsubseteq \mathscr{C}', and \quad \mathscr{C}' \sqsubseteq \mathscr{C} \blacksquare$$

Definition 4. *Join Operator ($\oplus$)*

Informally, the union of dimension sets and their corresponding attribute sets are calculated *in the result by the join operation.*

$$\oplus : S_{\mathscr{C}}(\mathbb{D}, \mathbb{A}) \times S_{\mathscr{C}}(\mathbb{D}, \mathbb{A}) \rightarrow S_{\mathscr{C}}(\mathbb{D}, \mathbb{A})$$

$\mathscr{C} \oplus \mathscr{C}' = \{< d'', A_{d''} > | d'' \in DIM(\mathscr{C}) \cup DIM(\mathscr{C}') \wedge A_{d''} = \{ATT(d'', \mathscr{C}) \cup ATT(d'', \mathscr{C}'\}\}$ ■

Definition 5. *Meet Operator* ($\odot$)
Informally, the intersection of dimension sets and their corresponding attribute sets are calculated in the result by the join operation.

$$\odot : S_{\mathscr{C}}(\mathbb{D}, \mathbb{A}) \times S_{\mathscr{C}}(\mathbb{D}, \mathbb{A}) \rightarrow S_{\mathscr{C}}(\mathbb{D}, \mathbb{A})$$

$\mathscr{C} \odot \mathscr{C}' = \{< d'', A_{d''} > | d'' \in DIM(\mathscr{C}) \cap DIM(\mathscr{C}') \wedge A_{d''} = \{ATT(d'', \mathscr{C}) \cap ATT(d'', \mathscr{C}'\}\}$ ■

Let $\mathscr{C}_1, \mathscr{C}_2, \mathscr{C}_3$ be three distinct context schemas in $\{S_{\mathscr{C}}(D, A), \sqsubseteq\}$. Join ($\oplus$) and Meet ($\odot$) operators have absorption, commutative, associative, and distributive properties.

(1) absorption: if $\mathscr{C}_1 \sqsubseteq \mathscr{C}_2$ then $\mathscr{C}_1 \oplus \mathscr{C}_2 = \mathscr{C}_2$, and $\mathscr{C}_1 \odot \mathscr{C}_2 = \mathscr{C}_1$.

(2) commutative: $\mathscr{C}_1 \oplus \mathscr{C}_2 = \mathscr{C}_2 \oplus \mathscr{C}_1$ and $\mathscr{C}_1 \odot \mathscr{C}_2 = \mathscr{C}_2 \odot \mathscr{C}_1$

(3) associative: $(\mathscr{C}_1 \oplus \mathscr{C}_2) \oplus \mathscr{C}_3 = \mathscr{C}_1 \oplus (\mathscr{C}_2 \oplus \mathscr{C}_3)$ and $(\mathscr{C}_1 \odot \mathscr{C}_2) \odot \mathscr{C}_3 = \mathscr{C}_1 \odot (\mathscr{C}_2 \odot \mathscr{C}_3)$

(4) $\oplus$ is distributive over $\odot$ and vise versa:

- $\mathscr{C}_1 \oplus (\mathscr{C}_2 \odot \mathscr{C}_3) = (\mathscr{C}_1 \oplus \mathscr{C}_2) \odot (\mathscr{C}_1 \oplus \mathscr{C}_3)$
- $\mathscr{C}_1 \odot (\mathscr{C}_2 \oplus \mathscr{C}_3) = (\mathscr{C}_1 \odot \mathscr{C}_2) \oplus (\mathscr{C}_1 \odot \mathscr{C}_3)$

Example 4. *Applying the join and meet operations on the two conference context schemas* $\mathscr{C}_{conf}$ *and* $\mathscr{C}'_{conf}$ *in Example 1 and Example 3 we get two new context schemas:*

$\mathscr{C}_{conf} \oplus \mathscr{C}'_{conf} = \{Date : \{a_1, a_2\}, Time : \{a_3\}, Location : \{a_4\}\}$
$\mathscr{C}_{conf} \odot \mathscr{C}'_{conf} = \{Date : \{a_1\}, Time : \{a_3\}\}$

Remark. *When coupling the context theory with a reasoning framework, those operations become underlying techniques for the conjunctions operations in reasoning semantics.*

4.1.3 Context Schema Lattice

With the join and meet operations on the POD $\{S_{\mathscr{C}}, \sqsubseteq\}$ we claim that the set $\mathscr{L} = (S_{\mathscr{C}}, \sqsubseteq, \oplus, \odot)$ is a lattice (Grätzer, 1971). In general, the lattice $\mathscr{L} = (S_{\mathscr{C}}, \sqsubseteq, \oplus, \odot)$ is *closed* and *distributive*. It is closed because its minimum element is the null context schema and the maximum element is the context schema composed with all dimensions in set $\mathbb{D}$ and all attributes of $\mathbb{A}$ associated across the dimensions. It is distributive because for $x, y, z \in \mathscr{L}$, we can verify the two properties: (1) $(x \odot y) \oplus (x \odot z) = x \odot (y \oplus z)$, and (2) $(x \oplus y) \odot (x \oplus z) = x \oplus (y \odot z)$. These two properties have enormous consequence on the system level. First, when the set $(\mathbb{D}, \mathbb{A})$ is fixed, the set of all context schemas in the lattice structure is closed with respect to the join and meet operations. Consequently, no typed context and thus context instances (defined in the later sections) are "left unaccountable". That means, we have a "closed world" of families of context instances and the knowledge they enclose. This closed world property fulfills the requirement of "sufficient completeness of actions in contexts" for ensuring safety and privacy properties at system execution stage. The distributive property enables "simplification" of expressions that involve context instances and hence the evaluation of "predicates" that need to be evaluated at context instances.

Definition 6. *$\mathscr{L}(\mathbb{D}, \mathbb{A}) = (S_{\mathscr{C}}(\mathbb{D}, \mathbb{A}), \oplus, \odot, \sqsubseteq)$ is a complete lattice.* ■

In the rest of the discussion we agree that $\mathbb{D}, \mathbb{A}$ is fixed and simply use the notation $S_{\mathscr{C}}$ for context schema and $\mathscr{L}$ for the complete lattice.

4.2 Tier 2: Typed Context Schema Representation and Calculus

Typed Context Schema (TCS) is a context schema in which attributes are associated with types. Each attribute is typed in the sense that it has a domain of values with respect to the type associated with it. Representation and calculus of schemas in TCS are inherited from the un-typed schemas defined in Tier 1.

4.2.1 Typed Context Schema Representation

Let $\mathbb{T} = \{T_1, T_2, ..., T_n\}$ denote a finite set of types such that for each type $T_x \in \mathbb{T}$ there exists a pair $< V_x, OP_x >$, where V_x is a "maximal" set of values, and OP_x denotes a set of operations allowed on the set V_x. By "maximal" we mean (1) V_x does not overlap with the set V_y of values of any other type $T_y \in \mathbb{T}$, and (2) if V'_x is any other set containing the values of T_x then $V'_x \subset V_x$. A type assignment to attributes in $\mathbb{A}$ is achieved through a map (a finite function) $M : \mathbb{A} \rightarrow \mathbb{T}$ that associates a unique type for every attribute in $\mathbb{A}$. We denote the set of all such type assignments by $\mathbb{A}^{\mathbb{T}}$. Two maps $M_1, M_2 \in \mathbb{A}^{\mathbb{T}}$ are *equal* only if $M_1(a) = M_2(a)$ for every $a \in \mathbb{A}$. For two maps $M_1, M_2 \in \mathbb{A}^{\mathbb{T}}$, $M_1 \neq M_2$, it is possible that $M_1 \cap M_2 \neq \emptyset$.

Definition 7. *Let $M \in \mathbb{A}^{\mathbb{T}}$. With respect to M the typed version of context schema $\mathscr{C}$ is denoted as $\mathscr{C}^M$, where*

$$\mathscr{C}^M = \{< d, A_d^M > | d \in DIM(\mathscr{C}), A_d^M = \{a^{M(a)} | a \in A_d\}\}$$

Example 5. *Let $\mathbb{D} = \{d_1, d_2\}$, and $\mathbb{A} = \{a_1, a_2, a_3\}$. A context schema $\mathscr{C}_1$ over $\mathbb{D}$ and $\mathbb{A}$ is defined as follows:*

$$\mathscr{C}_1 = \{d_1 : \{a_1, a_2\}, d_2 : \{a_2, a_3\}\}$$

Let $\mathbb{T}$ be the set of types such that $\mathbb{T} = \{Int, Char, Range\}$ where each type is defined as follows:

- *Int*
  - *Operations*
  - *Values:* $(-\infty, +\infty)$
- *$Char$*
  - *Operations:* $\{=, After, Before\}$
  - *Values:* $[a \cdots z]$
- *Range*

- *Operations:* $\{=, After, Before, Lowest, Highest\}$
- *Values:* $[1 \cdots 10]$

Let $M_1, M_2 \in \mathbb{A}^{\mathbb{T}}$ *as defined below:*

$$M_1 = \{a_1 \to Int, a_2 \to Char, a_3 \to Range\}$$

$$M_2 = \{a_1 \to Int, a_2 \to Int, a_3 \to Char\}$$

The typed attribute sets corresponding to these mappings, respectively, are $\{a_1^{Int}, a_2^{Char}, a_3^{Range}\}$, *and* $\{a_1^{Int}, a_2^{Int}, a_3^{Char}\}$. *Thus, the typed schemas are*

$$\mathscr{C}_1^{M_1} = \{d_1 : \{a_1^{Int}, a_2^{Char}\}, d_2 : \{a_2^{Char}, a_3^{Range}\}\}$$

$$\mathscr{C}_1^{M_2} = \{d_1 : \{a_1^{Int}, a_2^{Int}\}, d_2 : \{a_2^{Int}, a_3^{Char}\}\}$$

Notice that $\mathscr{C}_1^{M_1} \neq \mathscr{C}_1^{M_2}$ *because of the different type assignment* M_1 *and* M_2.

4.2.2 Typed Schema Calculus

We refer to the set of schemas of the same type, denoted as $S_{\mathscr{C}^M}$, as a family of typed schemas. Operations defined at schema level are extended to typed schema family. They are well-defined only within the family. That is, operations are not extended across families of typed schemas. Thus, the operations $\mathscr{C}_1^{M_1} \oplus \mathscr{C}_1^{M_2}$, $\mathscr{C}_1^{M_1} \odot \mathscr{C}_1^{M_2}$ are not defined since $M_1 \neq M_2$. However, for typed context $\mathscr{C}_3^{M_1} = \{d_2 : \{a_1^{Int}\}\}$ we can define $\mathscr{C}_1^{M_1} \oplus \mathscr{C}_3^{M_1}$ and $\mathscr{C}_1^{M_1} \odot \mathscr{C}_3^{M_1}$.

4.2.3 Typed Context Schema Lattice

For $M \in \mathbb{A}^{\mathbb{T}}$ a family of context schema $S_{\mathscr{C}^M}$ with join and meet operators is a complete typed lattice $\mathscr{L}^M$. This lattice includes all possible context schemas of type M. Within each family of schemas we can define schema operations meet, join and containment. In addition we may be able to define additional operations induced by the operators associated with type T induced by map M.

Example 6. *For a set of dimensions $\mathbb{D} = \{d_1, d_2\}$ and a set of attributes $\mathbb{A} = \{a_1, a_2\}$, let $\mathscr{L}$ be a context schema lattice that includes the set of context schemas defined in $S_{\mathscr{C}}$ as follows:*

$$S_{\mathscr{C}} = \{\mathscr{C}_\phi, \quad \{d_1 : \{a_1\}\}, \quad \{d_2 : \{a_2\}\}, \quad \{d_1 : \{a_1\}, d_2 : \{a_2\}\}\}$$

Say we have applied M_1 and M_2 on all contexts in $S_{\mathscr{C}}$, and hence we have $S_{\mathscr{C}^{M_1}}$ and $S_{\mathscr{C}^{M_2}}$. Thus, we can construct two lattices that have different types, i.e. $\mathscr{L}^{M_1}$ and $\mathscr{L}^{M_2}$, which are both a typed version of the general context schema lattice $\mathscr{L}$

In summary we have achieved the following results:

1. If $\alpha = |\mathbb{A}^{\mathbb{T}}|$ then there are potentially α mappings, each associating every attribute of $\mathbb{A}$ to a type in $\mathbb{T}$. Consequently, from one schema $\mathscr{C} \in S_{\mathscr{C}}(\mathbb{D}, \mathbb{A})$ we can generate α different typed schemas. Thus, for a fixed set of dimensions/attributes set (D, A) we can generate $\alpha \times \beta$ typed schemas where $\beta = |S_{\mathscr{C}}(\mathbb{D}, \mathbb{A})|$.

2. Let $S_{\mathscr{C}}(\mathbb{D}, \mathbb{A})^M = \{\mathscr{C}^M | \mathscr{C} \in S_{\mathscr{C}}(\mathbb{D}, \mathbb{A}), M \in \mathbb{A}^{\mathbb{T}}\}$. The operations "containment", "equality", "join" and "meet" defined at the schema level are directly inherited by this set of typed contexts.

3. The operations 'containment", "equality", "join" and "meet" cannot be defined for schemas that are from different typed families.

4. The context schema lattice $\mathscr{L}(\mathbb{D}, \mathbb{A})$ becomes a typed schema lattice $\mathscr{L}(\mathbb{D}, \mathbb{A})^{\mathbb{M}}$.

4.3 Tier 3: Context Instance Representation and Calculus

In all practical applications (et al., 2009; Held, Buchholz, & Schill, 2002c) the term "context" has been used as "meta information" annotating "certain scenarios" or "happenings". This is achieved by associating "values" (sometimes called "tags" or "events" or "situations") to "dimensions". In our theory we arrive at such "contexts" as "instances" of context schemas. The rationale is to provide a more abstract foundation from which we can generate several "families" of context instances. The advantages include (1) levels of abstractions to conceptualize and manipulate schemas and instances, (2) provide a strong typing for attributes, and (3) achieve a potentially infinite number of context "families", where all context instances in a family will have a well-defined set of operations. This rigorous and disciplined theory will enable the correct development of context toolkit and promote reuse..

4.3.1 Context Instance Representation

A Context Instance (CI) is an instantiated TCS in the sense that the attribute names in a TCS are substituted by values from the associated type domain. Hereafter we refer to context instance as context. To formalize, we start with one $\mathscr{C}^M$ be TCS of the lattice $S(\mathbb{D}, \mathbb{A})^M$, and use the substitution notation $[x/v]$ to mean that v is substituted for x. Let θ be a substitution function that assigns to each typed attribute a^T a value from the domain V of values associated with T. By a substitution θ we get an instance $I_\theta(\mathscr{C}^M)$ for the typed context schema $\mathscr{C}^M$ as defined below:

$$\theta : \mathscr{C}^M \to I_1(\mathscr{C}^M)$$

where $DIM(I_\theta(\mathscr{C}^M)) = DIM(\mathscr{C}^M)$, $\forall \quad < d : A_d^T > \in \mathscr{C}^M, \exists \quad < d : val^V > \in I_\theta(\mathscr{C}^M)$, $val^V = \{[a_i\theta/v_i] \mid v_i \in V\}$. That is, from each node (schema) in $\mathscr{L}(\mathbb{D}, \mathbb{A})$ we can generate a context instance for a fixed substitution. Because an atrribute can be substituted by any value from its associated type domain, for each type assignment to an attribute we get a family of contexts generated from one node in $\mathscr{L}(\mathbb{D}, \mathbb{A})$. Therefore, in addition to schema operations we can introduce the operations of the associated type to contexts within a family.

Example 7. *Let* $\theta_1 = \{a_1/1, a2/b, a3/[1-3]\},$. *By applying* θ_1 *on* $\mathscr{C}_1^{M_1}$ *in Example 5 we get one context* $I_1(\mathscr{C}_1^{M_1}) = \{d_1 : [1, \text{“}b\text{”}], d_2 : [\text{“}b\text{”}, [1-3]]\}$. *By applying the substitution* $\theta_2 = \{a_1/100, a2/e, a3/[2-4]\}$ *to* $\mathscr{C}_1^{M_1}$*, we get another instance* $I_2(\mathscr{C}_1^{M_1}) = \{d_1 : [100, \text{“}e\text{”}], d_2 : [\text{“}e\text{”}, [2-4]]\}$. *When* $\theta_3 = \{a_1/10, a_2/20, a_3/c\}$ *is applied to* $\mathscr{C}_1^{M_2}$ *we get the new instance* $I_3(\mathscr{C}_1^{M_2}) = \{d_1 : [10, 20], d_2 : [20, \text{“}c\text{”}]\}$ *of type* M_2. *We remark that operational consistency exists only for contexts within each family.*

Whereas the three tiers provide generality through abstraction levels, within tiers 2 and 3 we achieve regularity and extensionality. For certain types, such as “categorical types”, categories can be incrementally defined in order to suit the needs of specific application in tier 3. We achieve generating a large number of specialized contexts and can keep track of them using the family structure. Consequently, context calculus remain formal and correct with respect to operations associated with types.

Remark. *In context-based applications, such as contextual reasoning frameworks and context-aware applications, only context instances are used. The first two tiers are the theoretical basis for constructing typed context instances. We assume that Tiers 1 and Tier 2 have been accomplished already to generate Tier 3 contexts. Usually, context schemas are application-dependant, and once they are defined only context instances are relevant afterwards.*

4.4 Modeling Example

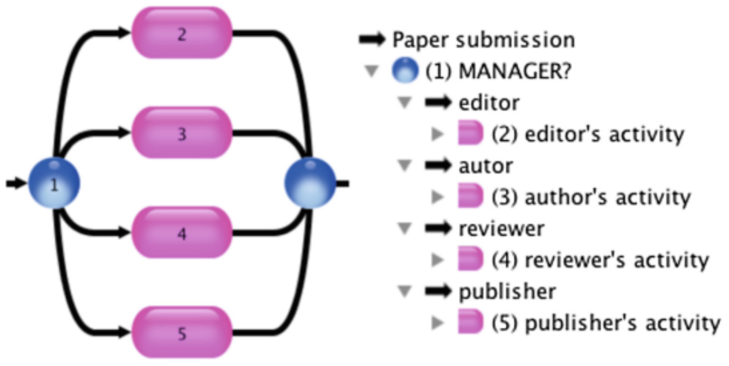


Figure 4.2: Snapshot of the paper submission example in (García & Brézillon, 2017)

Many different ad-hoc notations for modeling contexts can be found in (Interdisciplinary & Series, 1997-). One of them (García & Brézillon, 2015, 2017) is "paper submission context" shown in Figure 4.2. This notation, called contextual graph, has several ambiguities. For instance, there are modeling ambiguities such as what is the type of information that can be attached to each node? what does a node represent?. Also there are structural ambiguity such as how can a hierarchical relation be structured? Finally, there are technical and performance ambiguity such as what type of data can a contextual graph encapsulate? what the complexity of processing such graphs? We can eliminate these ambiguities by formally representing "paper submission context" using our 3-tier context representation. The actors in "paper submission" process are "publisher, editor, author, and reviewer". They share some activities like "canceling, submitting, checking". However, each actor is independent with respect to many other activities. For instance, editor can edit a paper, but cannot reject a paper, only a reviewer can. We explain how our approach can formalize this contextual graph model.

Step.1 We develop the context schema after identifying a set of dimensions, a set of attributes set, and a set of types. Based on the example, each actor has information such as “name”, “date-of-last-login”, thus we have ”Info“ dimension. An actor also has unique activities and common activities shared among all types of actors. Therefore, we have also another two dimensions which are “Unique”, and “Common”. Therefore, we have the set of dimensions $\mathbb{D} = \{Info, Unique, Common\}$. The attributes of each dimension can be identified by the information that a dimension holds. “Info” dimension for instance, holds the “name”, “lastlogindate” attributes. These attributes can simply be named anything, but we use meaningful names to make attributes readable and comprehensible. Based on example, each author has a list of unique activities, and a list of common activities. Therefore, both dimensions “Unique” and “Common” should have one attribute (later will be of type list). Therefore, our attribute set is $\mathbb{A} = \{name, lastlogindate, act\_common, act\_unique\}$. The types can be customized/user defined, or can be basic types like integer and string. The information in our attributes are either string, integer, or list of a generic type. Therefore, we define our set of types set as $\mathbb{T} = \{string, date, common, editor, reviewer, publisher, author\}$. The domain of values and operations associated with each type can be defined as follows:

- string: operations:$\{concat, intersect\}$, domain: set of alphanumeric values
- integer: operations: $\{+, -\}$, domain: $(date)$
- common: operations: $\{before, after, add, remove\}$, domain: list of common activities.
- editor: operations: $\{before, after, add, remove\}$, domain: list of unique activities for editor.
- reviewer: operations: $\{before, after, add, remove\}$, domain: list of unique activities for reviewer.

- publisher: operations: $\{before, after, add, remove\}$, domain: list of unique activities for publisher.
- author: operations: $\{before, after, add, remove\}$, domain: list of unique activities for author.

Based on the above choice the "Paper Submission Context Schema", denoted $\mathscr{C}_{PS}$, is defined as follows:

$$\mathscr{C}_{PS} = \{Info : \{name, lastlogindate\}, Unique : \{activities\}, Common : \{activities\}\}$$

Step.2 We assign types to attributes and generate typed schemas. The above schema is general to all types of actors.The tyes are assigned as below:

- $\mathscr{C}_{PS}^{edi} = \{Info : \{name^{string}, lastlogindate^{date}\}, Unique : \{activities^{editor}\}, Common : \{activities^{common}\}\}$
- $\mathscr{C}_{PS}^{rev} = \{Info : \{name^{string}, lastlogindate^{date}\}, Unique : \{activities^{reviewer}\}, Common : \{activities^{common}\}\}$
- $\mathscr{C}_{PS}^{pub} = \{Info : \{name^{string}, lastlogindate^{date}\}, Unique : \{activities^{publisher}\}, Common : \{activities^{common}\}\}$
- $\mathscr{C}_{PS}^{aut} = \{Info : \{name^{string}, lastlogindate^{date}\}, Unique : \{activities^{author}\}, Common : \{activities^{common}\}\}$

Step.3 We generate context instances from the typed schema . In this level values are "substituted" to attributes. For instance, an editor instance can be $\mathscr{C}_{PS}^{edi} = \{Info : \{\text{``}Jhon\ Dazer\text{''}, \text{``2017−05−04''}\}, Common : \{[submit, cancel, delete]\}, Unique : \{[edit, submit\_notes, con$
We can generate a family of instances for each type assignment.
This simple example illustrates the following advantages of our formalism.

- *Precision:* Ambiguities are eliminated.
- *Generality:* Once the first two tiers are done, in the third tier we can generate a family of context instances. In the informal graph model only one specific context could be shoe. Consequently, we have achieved generality.

- *Accuracy and Completeness:* We are able to generate *all meaningful contexts*, whereas in the graphical notation it is possible to identify contexts that are not meaningful and not all meaningful contexts can be represented.

4.5 Evaluation of the Context Theory

The merits of our context theory are compared below with other existing methods. We use the same evaluation criteria stated in Section 2.3.1 for this comparison. A summary of this evaluation is provided in Table 4.1.

(1) Formal Conceptualization (FC): Each dimension in a context represents a category of an information related to the context, and each attribute value associated with a dimension is a specific detail of significance to that category. Therefore, the context notation formalizes a multi-dimensional perspective of any circumstance conceptualized as context.

(2) Formal Model (FM): It is precise, typed, complete, and sound. No notational ambiguity, nor semantics ambiguity exists. Also, every context in the lattice is well defined and the set of contexts in the lattice is maximal.

(3) Formal Representation (FR): The representation as a collection of ordered pairs brings out all meaningful relations, once the set of dimensions, the set of types, and their domains are defined.

(4) Freedom of Structure (FrK): Context theory supports both flat and hierarchical structures.

(5) Fixed Knowledge (FxK): Context instances are static information that contains constants.

(6) Inferred Knowledge (InK): Once paired with an inference engine (*Contelog*), it is structurally capable of collecting all knowledge that can be inferred in the contexts enclosed by the complete lattice.

(7) Support Heterogeneous Data (SHD): The context schema allows heterogeneous types and their values for attributes associated with the dimensions. The heterogeneity of

#	Category	Research	Formalism			Expressiveness			Extensibility			Manipulable
			FC	FM	FR	FrS	FxK	InK	SHD	Inn	Det	
1	Context Theory	T	✓	✓	✓	✓	✓	×	✓	✓	✓	✓

Table 4.1: Evaluation of Our Context Theory against the same criteria of evaluation used to evaluate previous methods in section 2.3.1

*FC: Formal Conceptualization, FM: Formal Modeling, FR: Formal Representation, FrK:Freedom of Structure, FxK:Fixed Knowledge,InK: Inferred Knowledge, SHD: Support Heterogeneous Data, Inn:Innumerable, Det: Detachable

data in Tier 3 context instances is fixed to the types defined in Tier 2. Higher-order types (structured types) can be defined in Tier 2, which can generate heterogeneous hierarchical contexts in Tier 3.

(8) Innumerable (Inn): In theory, a context can have infinite number of dimensions. The context theory is capable of dealing with unlimited number of dimensions and attribute. However, in order to link it to a practical systems, it is mandatory to limit the theory to finite sets of dimensions and attributes.

(9) Detachable (Det): Context theory is completely detachable from any other component.

(10) Manipulable: Using operations defined in Context Toolkit, contexts can easily be manipulated.

Chapter 5

The Contelog Framework

Contelog is a Contextual Knowledge Base System (CKBS) that evaluates a given set of facts and rules in specific contexts to derive new facts that hold true in those contexts. In this chapter, the declarative, fixpoint, and proof theoretic semantics of *Contelog* programs are developed and their equivalences are established. The framework is explained via examples throughout the chapter. *Contelog* , conservatively extends the syntax and semantics of Datalog to reason with contextual knowledge. In this setting, contextual knowledge is reusable on its own. The significance is that by fixing the contextual knowledge, goal-specific analysis rules may be changed, and vice versa. By providing a theory of context (dicussed in Chapter 4), independent of the logic of the rules, we have developed a simple and sound context calculus using which it is possible to export knowledge reasoned in one context to another context. A more abstract and brief application of *Contelog* framework can be found in (Alsaig, Alagar, & Nematollaah, 2020). In this chapter, only the theoretical aspects of *Contelog* are discussed. In Chapters 6, 7, and 8, query processing algorithms, applications, and performance optimizations are discussed.

5.1 Note on Contexts

In the rest of the thesis "context" refers to context instances discussed in Chapter 4. When an application domain is fixed for *Contelog* reasoning, Context Schema, and Typed Context Schema are done using the context toolkit to make context instances. The syntax of context instances or simply context adopted from now on is,

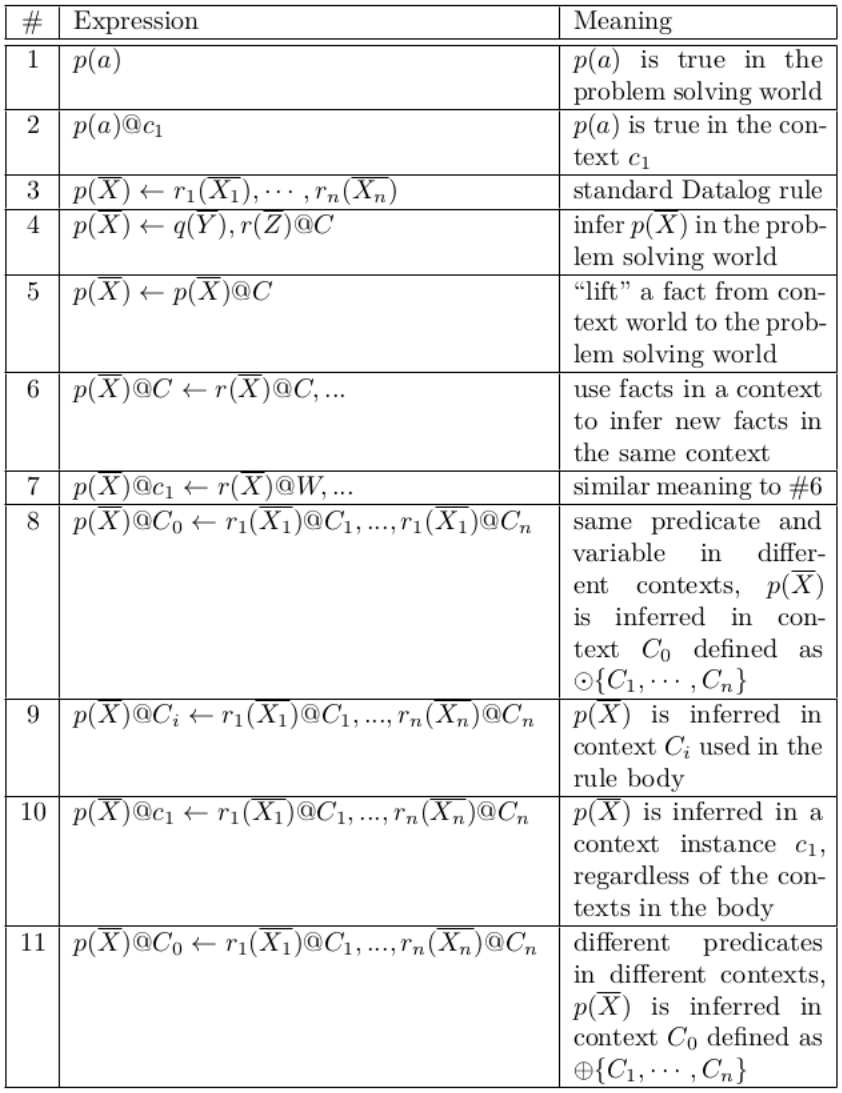
$c_1 = \{dimension_1 : attribute{-}set_1, dimension_2 : attribute{-}set_2, \cdots dimension_n : attribute{-}set_n\}$

5.2 The Syntax

A definite *Contelog* program P (or *c-program* , for short) is a finite set of facts, rules, and contexts. Each rule r in P is an expression of the form:

$$L_0 \leftarrow L_1, L_2, ..., L_n$$

where L_i is an atom of the form $p(\overline{X})$ or $p(\overline{X})@C$, and $\overline{X}$ is a list of variables and/or constants of the right arity. We refer to an atom of the form $p(\overline{X})$ as a *normal* atom, and refer to $p(\overline{X})@C$ as an *annotated* atom with context, *contextual/context* atom, or simply as an *annotated* atom. An annotated atom is used to express contextual information, that is, facts that are true only in specific context(s). For instance, $p(\overline{X})@C$ states that $p(\overline{X})$ is true at (or in) some specific context C. A fact is a special case of a rule in which $n = 0$. As in standard Datalog, we use lower case letters for predicate names, normal or context. A lower case letter in an argument or annotation of a predicate represents a constant and an upper case letter represents a variable. Note that we do not allow negation or function symbols in *Contelog* programs.
The atomic formula L_0 in r is called the rule head, while the conjunction of atoms $L_i(i > 0)$ is called the rule body. Each L_i in the body may be referred to as a subgoal. A *fact* in P is a special case of a rule in which the conjunction in the body is empty. As in the standard Datalog, we require the rule safety, that is, every normal or context variable appearing in a rule head must also appear in the body of that rule. This essentially ensures that every relation defined by a rule head is finite.



#	Expression	Meaning
1	$p(a)$	$p(a)$ is true in the problem solving world
2	$p(a)@c_1$	$p(a)$ is true in the context c_1
3	$p(\overline{X}) \leftarrow r_1(\overline{X_1}), \cdots, r_n(\overline{X_n})$	standard Datalog rule
4	$p(\overline{X}) \leftarrow q(\overline{Y}), r(\overline{Z})@C$	infer $p(\overline{X})$ in the problem solving world
5	$p(\overline{X}) \leftarrow p(\overline{X})@C$	"lift" a fact from context world to the problem solving world
6	$p(\overline{X})@C \leftarrow r(\overline{X})@C, ...$	use facts in a context to infer new facts in the same context
7	$p(\overline{X})@c_1 \leftarrow r(\overline{X})@W, ...$	similar meaning to #6
8	$p(\overline{X})@C_0 \leftarrow r_1(\overline{X_1})@C_1, ..., r_1(\overline{X_1})@C_n$	same predicate and variable in different contexts, $p(\overline{X})$ is inferred in context C_0 defined as $\odot\{C_1, \cdots, C_n\}$
9	$p(\overline{X})@C_i \leftarrow r_1(\overline{X_1})@C_1, ..., r_n(\overline{X_n})@C_n$	$p(\overline{X})$ is inferred in context C_i used in the rule body
10	$p(\overline{X})@c_1 \leftarrow r_1(\overline{X_1})@C_1, ..., r_n(\overline{X_n})@C_n$	$p(\overline{X})$ is inferred in a context instance c_1, regardless of the contexts in the body
11	$p(\overline{X})@C_0 \leftarrow r_1(\overline{X_1})@C_1, ..., r_n(\overline{X_n})@C_n$	different predicates in different contexts, $p(\overline{X})$ is inferred in context C_0 defined as $\oplus\{C_1, \cdots, C_n\}$

Figure 5.1: Possible formulas allowed in the *Contelog* framework

5.3 The Semantics of *c-programs*

A *c-program* P in *Contelog* can be viewed as having two components. The first component includes the information expressed as the facts and rules, and the second one includes the contextual information organized as a collection of context modules, which can be incorporated and used during query processing. In (Guha, 1991; McCarthy, 1993), they refer to these two types of information, respectively, as the "problem solving" world and the "context" world. The explanation in Figure 5.1 emphasizes that, for instance, a fact is true in problem solving world (#1) or in context world (#2). Or using a rule, it indicates that the facts inferred are in problem solving world (#4) or in context world (#6).

5.3.1 Model Theory

Without loss of generality, we restrict the study of semantics of *c-programs* to *Herbrand* structures. We begin with the development of the model theory, also called the declarative semantics. We show that every *c-program* in *Contelog* has the least model. We will then provide a fixpoint semantics of *c-program* and establish its equivalence to the least model. Finally, we propose a proof-theoretic semantics and show that it coincides with the least model.

The Herbrand universe of a *c-program* P includes the set U_P of all constants mentioned in the facts and rules in P, and those mentioned in any context incorporated in P. The Herbrand base of P, denoted as B_P, is the set of all ground atoms that can be constructed using the normal or context predicates, and the constants in U_P. The *Herbrand instantiations* of P, denoted as P^*, contains the ground instances of every fact and rule in P w.r.t. the normal and context atoms in B_P. Since function symbols are not allowed in *c-program* P, the Herbrand universe and Herbrand base of P are both finite, and hence the instantiated program P^* is finite as well.

As in the standard case, a substitution θ is a function that maps every variable to a term. In *Contelog* , a term is either a variable or a constant. Let L_1 and L_2 be two atomic formulas, normal or with contexts. We say that L_1 subsumes L_2, denoted as $L_1 \rhd L_2$, if there is a substitution θ such that $L_1\theta = L_2$, i.e., applying θ on L_1 yields L_2. If $L_1 \rhd L_2$, we say that L_2 is an instance of L_1. For example, $p(1, 2, 1)$ and $p(1, 1, 1)$ are both instances of $p(X, Y, X)$ but $p(1, 1, 2)$ is not. As examples with contexts, $p(1, 1, 1)@c1$ is an instance of $p(X, X, X)@C$, but not an instance of $p(X, X, X)@c2$ or $p(X, X, X)$.

5.3.2 Herbrand Interpretation

A Herbrand interpretation $\mathcal{I}$ for a contelog P is a subset of the Herbrand base B_P of P. A ground normal atom A is true under $\mathcal{I}$, denoted as $\mathcal{I} \models A$, iff $A \in \mathcal{I}$. A ground context atom $A@c$ is true under $\mathcal{I}$ iff $A@c \in \mathcal{I}$. Let $r \equiv L_0 \leftarrow L_1, \cdots, L_n$ be any rule in a *c-program* P, where each L_i is a normal or a context atom. An interpretation $\mathcal{I}$ satisfies r, denoted as $\mathcal{I} \models r$, iff there exists a substitution θ such that $L_0\theta \in \mathcal{I}$ whenever $L_i\theta \in \mathcal{I}$, for $1 \leq i \leq n$.

Finally, a Herbrand interpretation $\mathcal{I}$ satisfies P, denoted as $\mathcal{I} \models P$, iff $\mathcal{I}$ satisfies every rule and fact in P. In this case, we may also say that $\mathcal{I}$ is a *model for* P.

Corollary 1. *(Model Intersection Property). Let $\mathcal{M}$ be any non-empty collection of Herbrand models of a* Contelog *program P. Then the intersections $\mathcal{M} = \bigcap_{i\geq 0}\{\mathcal{M}_i \mid \mathcal{M}_i \models P\}$ is also a Herbrand model of P.*

Proof. Suppose $\mathcal{M}$ is not a model of P. Then there exists a ground instance $A \leftarrow B_1, \cdots, B_n$ of a rule r in P that is not true in $\mathcal{M}$. This means that $\mathcal{M}$ includes B_1,B_2,$\cdot$, B_n but not A. Furthermore, B_1,B_2,$\cdot$, B_n are contained in every model $\mathcal{M}_i$ in $\mathcal{M}$. Thus there must exist at least a model $\mathcal{M}_j$ in $\mathcal{M}$ that does not contain A. This in turn implies that r is not true in $\mathcal{M}_j$ and hence $\mathcal{M}_j \not\models P$, which is a contradiction. $\square$

The above result yields a characterization of the intended, declarative semantics of definite programs in *Contelog* , described as follows. If the collection of Herbrand models $\mathcal{M}$ of P considered in the above result includes all such models of P, then $\mathcal{M}$ is the *least model* of P. This model is an abstraction of the world described by the program and the contexts it uses. It is established as the next corollary.

Theorem 1.1. *(The Least Model). The* least Herbrand model *of a c-program P is the intersection of all the Herbrand models of P. That is, $\mathcal{M}_P = \bigcap_{i\geq 0}\{\mathcal{M}_i \mid \mathcal{M}_i \models P\}$.*

Example 8. *We use* Contelog *to formally program the locator example introduced in figure 1.1 of chapter 1. First, we identify the context and model it. For a person coming from east (contextual information), the library will be on the right (contextual information) of the person. Thus, the context contains two dimensions: the direction a person comes from ($from$) and the side a person should go to (to). The dimension "$from$" has one attribute which is the direction name that can have the values "east" or "west". Similarly, the dimension "to" has one attribute which is the side name that have the values "$left$" or "right". Our* Contelog *program P_{bu} has two binary predicates. One is $p(X, Y)$ which refers to a person X coming from a direction Y. The second is $side(X, Z)$ which refers to person X and the side Z on which the library is located. This information can be expressed at the following* Contelog *program.*

The Building Locator program in Contelog

```
#Contexts
c_1 = {from : [east], to : [right]}.
c_2 = {from : [west], to : [left]}.
#Facts
p(john,east).
p(rose,west).
#Rules
p(X, Y)@C ← p(X, Y), from(Y)@C.
side(X, Z)@C ← p(X, Y)@C, to(Z)@C.
```

To save space, restricting to the fact $p(john, east)$, the context universe/lattice, Herbrand universe, and Herbrand base are as follows.

```
$\mathscr{L}_{\mathscr{C}} = \{c_1, c_2, c_3(= c_1 \odot c_2), c_4 = (c_1 \oplus c_2)\}$
$C_U = \{east, west, left, right\}$
$U_P = \{john, east, west, left, right\}$
$B_P = \{p(john, east), p(east, west), p(east, john)...\} \cup \{p(john, east)@c_1, ..., to(east)@c_1, ...\}$
```

5.3.3 Fixpoint Semantics

In this section we introduce a bottom-up, fixpoint method to evaluate programs in *Contelog* and establish its equivalence to the least model. We adapt the *Elementary Production Principle (EPP)* (Abiteboul, Hull, & Vianu, 1995b; Ceri et al., 1989) from standard Datalog to derive new facts which are logical consequences defined by the input facts and rules. Let P be any *c-program* , P^* be the Herbrand instantiation of P, and r be a c-rule in P. Applying the EPP on r is the process of inferring the head of r if a successful *unification* exists for the body of r. INFER is a simple algorithm used in (Abiteboul et al., 1995b; Ceri et al., 1989) that applies EPP concept on the rules and facts in a *Contelog* program. The algorithms EPP, INFER, and other query processing approaches are explained in Chapter 6. This algorithm is repeatedly applied until no more new fact can be inferred. In this case, we say that the least fixpoint is reached. Note that since function symbols are not allowed in *Contelog* , the Herbrand universe of a *c-program* is always finite and hence the fixpoint is always reached in finite time.

The bottom-up evaluation of a *c-program* P can be defined as a mapping $T_P : 2^{B_P} \rightarrow 2^{B_P}$, where 2^{B_P} denotes the subsets of the Harbrand base B_P. The least fixpoint of T_P is obtained at iteration n if $T_P^n = T_P^{n-1}$. That is,

$$T_P{}^n = \begin{cases} \phi & \text{if } n = 0 \\ \\ T_P(T_P^{n-1}) & \text{if } n > 0 \end{cases}$$

and the least fixpoint lfp of T_P is defined as:

$$lm(T_P) = \bigcup_{n \geq 0} T_P^n$$

The following two results ensure that (1) the fixpoint computation terminates in finite time and (2) the least fixpoint includes all logical consequences of the input *c-program* . The proofs can be easily obtained by adopting from the standard case. Moreover, as in the standard Datalog, the number of iterations to compute the least fixpoint is polynomial $O(n^k)$, where n is the number of constants in the Herbrand universe and k is the maximum arity of the predicates in the input *c-program* .

Lemma 1.2. *The T_P operator is both monotone and continuous.*

Theorem 1.3. *The* least fixpoint *of T_P exists and is identical to the least model of P.*

5.4 Currency Exchange - A Complete *Contelog* Example And its Evaluation

The currency is a context-based element. It changes based on the context of the person, namely, the location. This is a basic context-based problem that can manifest more complicated issues such as exchange rate from currency to another, the contextual-translation, and the direction problem. Since *Contelog* is not equipped with functions it cannot calculate or compute numerical equations, however it presents the base for such interesting extension. The goal of this program is to identify the currency of the person based on the context of this person. If this person is in USA then currency is \$, if he/she is in France then it is

Euro, and so on. Consider the following currency contexts for the *c-program* (P_{cu}) which define c_{eu}, c_{ca}, c_{us} for EURO, CAD, and USD, respectively, as follows:

The Currency Contexts for the *c-program* P_{cu}

```
c_eu = {currency:[eur], location:[france]}
c_ca = {currency:[cad], location:[canada]}
c_us = {currency:[usd], location:[usa]}
```

The *Contelog* program P_{cu} defined below has two binary predicates; $person(X, Y)$ which asserts that person X is in location Y, and $loc\_curr(X, Z)$ which asserts that person X should use the currency Z because of his/her location. The program also defines two unary context predicates $location(Y)$ and $currency(Z)$, used during the reasoning process. The first rule r_1 in the *c-program* below identifies different categories of individual persons based on their contexts, and r_2 infers the currency to be used by a person based on his/her category identified by r_1.

The *c-program* P_{cu}

```
person(john,canada).
person(mary,france).
person(ray,usa).
r1: person(X,Y)@C :- person(X,Y), location(Y)@C.
r2: loc_curr(X,Y)@C :- person(X,Z)@C, currency(Y)@C.
```

The fixpoint of the program above $T_{P_{cu}}$ contains the following facts; derived facts are preceded by **.

The facts derived from P_{cu}

```
person(john,canada).
person(mary,france).
person(ray,usa).
**person(john,canada)@c_ca.
**person(mary,france)@c_eu.
**person(ray,usa)@c_us.
**loc_curr(john,cad)@c_ca.
**loc_curr(mary,euro)@c_eu.
```

```
**loc_curr(ray,dollar)@c_us.
```

In the first iteration of the bottom up evaluation, we get all the facts:
$T^0_{P_{cu}}(\phi) = \{person(john, canada),$
$person(mary, france), person(ray, usa)\}.$
In the second iteration, rule r_1 can be applied as some predicates in its body can be unified with the facts obtained. For instance, $person(X, Y)$ is unified with $person(john, canada)$. Passing the substitution to the next subgoal in r_1, it yields $location(canada)@C$. Since $location(canada)$ is a context predicate, it is evaluated against the contexts. As this predicate is contained in one of the contexts, i.e. c_{ca}, we get a hit and a substitution value. This in turns allows inferring $person(john, canada)@c_{ca}$. The same process follows for other possible hits. We thus obtain the following facts.
$T^1_{P_{cu}}(T^0_{P_{cu}}) = \{person(john, canada)@c_{ca},$
$person(mary, france)@c_{eu}, person(ray, usa)@c_{us}\}$
In iteration 3, rule r_2 is triggered since its first predicate $person(X, Z)@C$ is unified with $person(john, canada)@c_{ca}$ which gives the next subgoal $currency(Y)@c_{ca}$ in the rule body to verify against the contexts. This maps Y to cad, the value of the currency dimension in context c_{ca}, using which we infer $loc\_curr(john, cad)@c_{ca}$. Repeating this process for all the other facts and contexts, we obtain the following results.

$T^2_{P_{cu}}(T^1_{P_{cu}}) = \{loc\_curr(john, cad)@c_{ca},$
$loc\_curr(mary, euro)@c_{eu}, loc\_curr(ray, usd)@c_{us}\}.$
In the forth iteration, we note that $T^3_{P_{cu}}(T^2_{P_{cu}}) = T^2_{P_{cu}}$, which means the fixpoint is reached and the evaluation process terminates.

Remark. *Although top-down evaluation approach was not developed for* Contelog *, it is well-understood. Top-down approach is moving from the goal/head of the rule to prove its body (subgoals), it is also called backward chaining. A basic method that applies the top down approach in standard Datalog is called* Query-Subquery method*. In* Contelog *, the same approach can be used to perform the top-down process. This method is sound and complete.*

#	Framework	Multiple Programs	Separate Rules	Local Dynamism	Separate Data	Simplicity	Extension
[1]	*Contelog* : logic-based	✓	✓	✓	✓	✓	✓

Table 5.1: Framework level evaluation for *Contelog*

5.5 Evaluation and Observations for *Contelog* Framework

For the comparison to be fair, we apply the evaluation criteria developed in Chapter 2 to *Contelog* model to assess its merits. Table Table 5.1 provides a summary of results.

(1) *Reuse:* Due to the loose coupling structure between context and *Contelog* programs, it is easy to see to convince ourselves that multiple programs can reuse contexts, even simultaneously, without affecting one another. The local context is the one affected by the *Contelog* program which is an internal copy of the global context used by all other programs.

(2) *Ability for Multiple Reasoners to exist(Separate Rules):* The structure of *Contelog* supports this requirement. Contexts, facts, and rules are three different components in *Contelog* program. They communicate formally through intertwined syntax and semantics. Facts are used to define program specific absolute truths. Contexts are used to define settings and meta-information of the program. Finally, rules are used to use facts and context to achieve certain goal.

(3) *Local Dynamism without Global Side Effect (Local Dynamism):* The context within *c-program* , is dynamic. Rules can be used to infer new context and add it to the local set of contexts, and use it to infer new facts/contexts. This allows the 'transitive closure' within context, which is referred to as 'contextual recursion'. This, however, does not affect the global context with which other *Contelog* programs may be paired.

(4) *Separation of Data from Context (Separate Data):* Facts or absolute truths within a *Contelog* program is different from contextual information. This is well-supported by the structure of *Contelog* program. For instance, in the domain of context-aware systems, temperature and humidity are considered data, while meaning of this data such as 'hot' or 'humid' is a contextual information that should be defined in context

component. This feature is absent in many Ontology-based context-aware systems, where contextual information and factual information cannot be separated.

(5) *Representation Simplicity (Simplicity): Contelog* adopted the syntax of Datalog to maintain simplicity. The context is added after the '@' notation to avoid confusion between predicate attributes and contextual attributes. Also, for simple reading and intuitive understanding so contextual predicates can be read 'the predicate p is true AT context c'.

(6) *Extensibility: Contelog* is a conservative extension of Datalog. That is, it has more features in terms of syntax and semantics, but it inherits all features of Datalog. It can run Datalog programs without the presence of contextual information.

Chapter 6

Query Processing and Optimization Algorithms

In the previous chapter, we introduced *Contelog* formalism and explained its inference engine through *Contelog* semantics. In this chapter, we elaborate on the inference mechanism for evaluating *Contelog* programs. The result of a *Contelog* program is a set of regular/contextual facts inferred from the facts, rules, and contexts in the input program. We consider the bottom-up evaluation approach that uses substitution and unification through which variables in the rules are mapped to constants in facts and contexts.

Given a query, essentially a conjunction of positive atoms, the query processing goal is to find all those tuples that are logical consequences of the program and subsumed by the query. A *basic query* in *Contelog* is simple or annotated atom. Formally, a *c-query* is expressed as follows:

$$?\!-p(X,Y,...)@C. \qquad or \qquad ?\!-p(X,Y,....).$$

where X,Y,..., could be regular variables or constants, and C a context variable or instance.

The query processing uses the facts, rules, and contexts in the *c-program* to determine the output tuples that can be subsumed by the input *c-query* . Unlike *top-down evaluation approaches*, bottom-up approaches are not ”goal oriented." As a result, there are two sources of inefficiency problems recognized and addressed for Datalog programs. The first one is that, tuples derived at an iteration *i* continues to be derived at subsequent iterations. The other source of inefficiency is that if the user wishes to evaluate a program to determine the answers to a given query, the structure of the query and the binding information are not taken into consideration until the evaluation process terminates. At that point, the output tuples that are subsumed by the input query are returned and the rest are discarded. Semi-naive method solved the first problem and the Magic Sets rewriting method solved the second problem successfully for Datalog programs. In our context of programs in *Contelog* , we adopt the ideas used in Datalog and propose solution techniques that solve the aforementioned two problems for efficient evaluation of *Contelog* programs. The results of our numerous experiments indicate significant performance improvement. In what follows we provide details of the proposed processing and optimization methods. The methods extend the standard unification and Elementary Production Principle (EPP) procedures that take into account the presence of contexts. We also illustrate the steps through examples. Performance measurement and optimization is discussed in the next chapter.

6.1 Preliminaries

6.1.1 Pre-processing

Contelog engine executes some pre-processing tasks against the context module and the program to ensure correctness of the syntax. Details of pre-processing tasks are discussed in the technical documentation of the engine (Alsaig, 2017).

6.1.2 Unification Conditions

In order for a non-ground atom to be unified with a fact, there are three conditions that have to be met. (1) Both the atom and the fact must have the same signature. Predicate signature is the predicate name (functor) and the number of attributes (arity) and context, if any. If an atom and a fact do not have similar signature, they can't be unified. (2) A fact unifies with a non-ground atom only. A non-ground atom cannot be unified with another non-ground atom, and (3) A fact unifies with a non-ground atom only if it matches its binding pattern. That is, a predicate $p(X, X)@C$ cannot unify with $p(a, b)@c1$ because the binding of the predicate only accepts similar arguments in its attributes. Therefore, it will only accept similar facts that respect its pattern such as $p(b, b)@c2$. If those three conditions are met for certain non-ground atom and a fact, then the atom becomes unfiable with that particular fact. It then uses the algorithm UCA(fact,non-ground atom) that implements unification of the argument atoms.

Unifiability Check Algorithm (UCA)

```
Input: F: a ground atom, P: a non-ground atom
Output: boolean True for unifiable, False for non-unifiable
Method:

IF IS-FACT(F) and IS-PREDICATE(P):
IF functor(F) EQUALS-TO functor(P):
IF number-of-arguments(F) EQUALS-TO number-of-arguments(P):
IF number-of-contexts(F) EQUALS-TO number-of-contexts(P):
IF binding(F) EQUALS-TO binding(P):
    RETURN TRUE
ELSE
    RETURN FALSE
```

6.1.3 Local Unification

Elementary Production Principle (EPP) is the rule that applies all possible substitutions on atoms to produce all possible ways of unification. This rule is extended to *Contelog* , to which we refer as *local unification*. In essence, it is applied on each subgoal in a rule body independently. For instance, for a *Contelog* atom $L_1 = p(X, X)@C$ and a ground fact $L_2 = p(a, a)@c1$, an acceptable local unification is $\theta = \{X \to a, C \to c1\}$. In this case, we say L_1 subsumes L_2, denoted $L_1 \rhd L_2$, because there exists a substitution θ such that $L_1\theta = L_2$.

The local unification receives two inputs, a non-ground atom A and the set S of ground facts, and performs the following three steps,

(1) Group all ground facts in S that are unifiable with A.

(2) For each unifiable fact, determine substitution θ

(3) Combine all θ's and add it to a set of local unification related to A.

The Local Unification Algorithm (LU)

```
To Call: $LocalUnify(FACTS, CONTEXTS, non-groundpredicate)$
Input: $\Pi^*$: the set of all ground facts and contexts in c-program, $P@C$ a
    non-ground atom.
Output: LU_{P@C}: set of local unification related to $P@C$.
Method:


UNIFIABLE={}
FOR EACH fact in $\Pi^*$:
 IF UCA(fact,P@C):
    APPEND-TO(fact, UNIFIABLE)

FOR EACH fact in UNIFIABLE:
    Determine $\theta_i$
    APPEND-TO($\theta_i, LC_{p@C}$)

RETURN $LU_{P@C}$
```

6.1.4 Global Unification

The global unification for *Contelog* is to apply one local unification for each predicate in a rule body, making sure those unifications do not conflict. The main difference between global and local unification is that global unification considers all variables in a rule body, whereas the local unification considers only the variables pertinent to one predicate. A conflict-free global unification has two main features: (1) no variable is unified twice with different values. (2) no variable is left without substitution. If a global unification has those two features it is accepted for inferencing, otherwise, it is skipped. In technical terms, the global unification for a rule is a set of global $\theta's$ resulted from the Cartesian Product of all local unification for each predicate in a rule body for a particular rule.

The steps to accomplish the global unification is informally presented below

(1) Make a set $Global_\theta = \theta_1 \times \theta_2 \times ... \times \theta_n$ where θ_i corresponds to the local unifications of the i^{th} subgoal of the rule.

(2) For each item in $Global_\theta$ apply the unifcation on all predicates in the body of the rule.

(3) If the unification is conflict-free, add it to global unification (GU) set related to the input c-rule, else, skip and try the next unification.

The Global Unification Algorithm (GU)

```
To Call: $GlobalUnify(LU_{r_i}, BODY_{r_i})$
Input: $LU_{r_i}$ = $\{\theta_1, \theta_2, ..., \theta_n\}$: set of all local unifications related to subgoals of
    $i_{th}$ c-rule, $BODY_{r_i}$ = subgoals of the $i_{th}$ c-rule.
Output: $GU_{r_i}$: set of global unification related to the $i_{th}$ c-rule.
Method:
$\theta_{r_i} = \theta_1 \times \theta_2 \times ... \times \theta_n$
FOR EACH item IN $\theta_{r_i}$:
    SUBSTITUTION(item, $BODY_{r_i}$):
        IF CONFLICT-FREE:
            APPEND-TO(item, $GU_{r_i}$)
RETURN $GU_{r_i}$
```

6.1.5 Inference Algorithm

Inference is the process of computing the logical consequence of a rule in *Contelog* . This process comes after the local and global unification. It is to make sure, first, that each variable in the rule head is substituted. Otherwise, it is considered a violation of the rule safety rules in Section 5.2. Second, it is to add this new fact to the set of facts of a c-program. The steps of inference for a c-program are as follows.

(1) For each substitution in global unification, run the inference check.

(2) If success (that is, all variables have been substituted), then add this fact to the set of inferred facts. Otherwise, move to the next substitution.

INFER

```
To Call: $INFER(GU_{r_i}, r_i)$
Input: $GU_{r_i}$, $r_i$: the $i_th$ rule in c-program
```

```
Output: $IDB$: set if inferred facts.
Method:
FOR EACH item IN $GU_{r_i}$:
    fact = INFER(item, $r_i$):
        IF successful-inference(fact):
            APPEND-TO(fact, $IDB$)
RETURN $IDB$
```

6.2 Goal-Independent Bottom-up Evaluation

This category of bottom-up evaluation does not consider any query from the user in its evaluation process even if it prompts the user to enter query. In fact, usually systems that follow this approach calculate all possible results for given facts and rules as a pre-processing step before running the system. This is to avoid the wasted resources on processing the evaluation with every new query. The evaluation continues only if the user/application changes the facts/rules of the system.

Once this pre-processing is completed, answering any user query is a matter of searching in the already-calculated set of tuples. Usually, the set of results are traversed using the well-known tree-search algorithms, namely the Depth-First Search (DFS) and Breadth-First Search (BFS), to find the matching tuples.

Due to its high cost, this approach is not used in active-reasoning systems that handle frequent changes in the input programs (facts,rules and/or contexts). This approach, however, is useful in passive-reasoning in which it runs only few times, and find answers to user query according to the offline/pre-processed results.

All methods under this category follow similar steps to compute results. The steps are as follows.

(1) PREPROCESS: Run evaluation process on given set of facts/rules/contexts.

(2) QUERY-SEARCH:

- Prompt-user to enter his/her query.
- Apply DFS/BFS algorithms to find relevant tuples.

6.2.1 Naive Evaluation Method

The naive method is the basic method for evaluating *Contelog* programs. It is an iterative process that at each iteration step applies the EPP inference rule to derive tuples, and terminates when no more new tuples is generated. It is called 'naive' because it does not perform any optimization. A drawback of the naive method is that when a tuple is derived in some iteration, it is derived in subsequent iterations. In Datalog, this repetitive derivations is a source of inefficiency which is addressed by the semi-naive evaluation method.

The steps of the naive method for *Contelog* is as follows:

(1) For each rule, for each subgoal in the body, run the LocalUnify algorithm.

(2) For each rule, run the GlobalUnify algorithm.

(3) For each rule run INFER algorithm

(4) Add new inferred facts

(5) Combine existing facts set EDB, with the inferred facts IDB, and run step(1) again until no more new facts are added.

Naive Evaluation Algorithm

```
To Call: $NaiveEV$(EDB, RULES, CONTEXTS)
Input: EDB (set of Facts), RULES (set of c-rules $r_1, r_2, ..., r_n$), CONTEXTS (set of
    contexts)
Output: RESULTS: set of all input/derived facts.
Method:
DO {
    OLD-EDB = EDB
    FOR EACH $r_i$ IN RULES:
        # Local unification
        FOR EACH $body_j$ in $r_i$
            $LU_j$ = LocalUnfiy(EDB U CONTEXTS,$body_j$)

        # Global unification for each rule
        GU_{r_i} = GlobalUnify(LU, BODY(r_i))

    # inference process
    IDB = INFER(GU_{r_i}, r_i)

    # combine with EDB if there are new facts inferred
    IF IDB IS_NEW:
       EDB = EDB U IDB

# Check termination condition
}WHILE (EDB!= OLD-EDB)
RESULTS = EDB
RETURN RESULTS
```

We remark that the naive method is an iterative process in which an atom, once derived, will be derived at every subsequent iteration. This explains one of the two major sources of inefficiency of the method.

6.2.2 Semi-Naive Method

The semi-naive method for *Contelog* programs is a step towards improving the bottom-up evaluation. It simply uses a mechanism in the evaluation process to ensure that the same derivation is not repeated using the same facts, contexts, and rules. The method is described as follows.

(1) For each rule and each body predicate in the rule, run the LocalUnify algorithm on the “un-used" facts.

(2) For each rule, run the GlobalUnify algorithm.

(3) For each rule run the INFER algorithm using only the un-used facts, and book-Keep the ground facts used in the inference and label them as ”used"

(4) Add new inferred facts and label them as ”unused"

(5) Combine existing facts set EDB, with the inferred facts IDB, and run step(1) again until no more new facts are added.

Naive Evaluation Algorithm

```
To Call: $SemiNaiveEV$(EDB, RULES, CONTEXTS)
Input: EDB (set of Facts), RULES (set of c-rules $r_1, r_2, ..., r_n$), CONTEXTS (set of
    contexts)
Output: RESULTS: set of all input/derived facts.
Method:
DO {
    OLD-EDB = EDB
    FOR EACH $r_i$ IN RULES:
        # Local unification consider only unused facts/contexts
        FOR EACH $body_j$ in $r_i$
            $LU_j$ = LocalUnfiy(EDB(unused) U CONTEXTS(unused),$body_j$)

        # Global unification for each rule
        GU_{r_i} = GlobalUnify(LU, BODY(r_i))

        # inference process label used facts as used in the set
        IDB = INFER(GU_{r_i}, r_i)
        EDB(used) = FACTS USED IN INFERENCING
        # combine with EDB if there are new facts inferred
        IF IDB IS_NEW:
            EDB = EDB U IDB
            # label new facts as unused to limit inferencing
            EDB (unused) = IDB


# Check termination condition
}WHILE(EDB!= OLD-EDB)
RESULTS = EDB

RETURN RESULTS
```

Although semi-naive method makes a considerable performance enhancement over the naive, it may suffer from another source of inefficiency during the evaluation process for not considering the structure and binding information of the input query. This can result in deriving ”irrelevant" tuples which will be identified and discarded only when the process terminates. In Datalog, this problem is addressed through the Magic Sets – a rewriting technique which restricts tuple derivations by adding new facts and rules as well as injecting new subgoals in the body of existing rules. Following a similar idea, we propose a program rewriting/transformation technique, called ”Magic Context", to avoid derivations of irrelevant tuples for evaluating *Contelog* programs when there is a given query as well. We next provide details of the proposed rewriting for *Contelog* .

6.3 Magic Context: Goal-Oriented Bottom-up Evaluation

Following a similar idea of the magic sets rewriting technique developed for standard Datalog programs (Ullman, 1989), we introduce the notion of *magic contexts* and propose a rewriting algorithm to avoid derivation of irrelevant tuples in a semi-naive evaluation of queries for *Contelog* programs. The proposed algorithm can also be used as an alternative to the magic sets technique for standard Datalog programs. As *Contelog* is a conservative extension to Datalog, by definition, it works for both Datalog and *Contelog* programs.

As discussed before, the bottom-up naive evaluation of Datalog programs suffers from two sources of inefficiency. First, a tuple derived at some iteration continues to be derived at every subsequent iteration until the process terminates. Second, the structure and binding information of the input query is considered only when the evaluation process terminates. This means, potentially many tuples derived during the evaluation will be discarded at the end for not being relevant to the query. The first problem was addressed using the semi-naive evaluation method, which keeps track of newly generated tuples in the current iteration compared to those derived in the previous iteration. The second source of inefficiency was addressed through the magic sets rewriting technique, which for a given input query, it generates a new program that includes magic facts and associated rules, such that when evaluated, it restricts the derivations to the relevant tuples as much as possible.

Magic sets rewriting is a transformation that considers the predicate and bindings information (constants) in the user query. Two well-known rewriting techniques for Datalog programs are Magic Set Transformation (MST) (Ceri et al., 1989; Ullman, 1990) and Demand Transformation (DT) (Tekle & Liu, 2010, 2011), which follow the same idea but DT shows a better run-time performance (Liang et al., 2009).

6.4 Issues in Extending *Contelog* with Magic Sets Technique

While Magic Context is based on a similar idea of MS in Datalog, it follows a different approach. To show this, we show that a straightforward Magic Sets technique has two drawbacks for *Contelog* programs. First it affects simplicity and expressiveness, explained as follows. To express contexts in Datalog, it is required to annotate or use special predicates to express contexts and contextual predicates in Datalog syntax. This coding of contexts in Datalog destroys the declarative and modular advantages of context structures in *Contelog* for resulting in bulky expressions and loss of separation of concerns principle in programming practices, hence making comprehension, maintenance, reuse, and management of the program too costly. Another problem of coding a *Contelog* program in Datalog is losing the “first class citizenship" status of contexts, as a result of which it is not possible to execute a program under different sets of contexts without redoing context-coding for every set of contexts. The second drawback is the performance which is affected negatively because of the extra layer of coding *Contelog* to Datalog in order to use the MST or DT techniques in Datalog, and finally transform the results back to *Contelog* . At this point, we also need to mention a third drawback, which we elaborate later, that is losing the advantage of concurrent query processing in a single run of *Contelog* programs. Motivated by simplicity, declarative, expressiveness, efficiency, separation of contexts and programs, and attain a higher level of abstraction, we developed the Magic Context Transformation (MCT) method to optimize query evaluation of *Contelog* programs.

The idea of MCT is as follows. We view a query Q over a *Contelog* program P as a context for evaluating P. Following this view, we transform Q into a context represented in *Contelog* and rewrite P accordingly. We refer to this as magic context transformation (or MCT, for short), which essentially uses magic contexts to generate a new *Contelog* program which when evaluated returns the desired results more efficiently. The efficiency advantage obtained by the MCT rewriting provides a simpler alternative solution to the magic set rewriting technique MST for standard Datalog programs.

6.5 Magic Context Transformation

Given a *Contelog* program Π_C and a query Q, we view Q as providing a context Q_C for evaluating Π_C. The intuitive idea of the Magic Context Transformation (MCT) algorithm proposed in this work is to using Q as a context and generate a rewriting of Π_C such that when evaluated, it restricts derivations of tuples to only those relevant to Q. Following this view, in addition to the original contexts of Π_C, the rewritten program Π'_C generated by MCT includes new context(s) and modified rules, defined by the query context Q_C, or the *magic context*, as we call it. We remark that the rewritten program is equivalent to the original program with respect to the query context Q_C.

Intuitively this view works because conceptually context defines an evaluation space which restricts the search for certain facts related to the context. The proposed rewriting modifies the rules in the program accordingly to abide by this restriction. Practically, the proposed rewriting is a realization of a theory of contexts defined in (Guha, 1991), by which a context is viewed as a theory that can have micro-theories that explain different phenomena in different domains. Furthermore, a predicate can be lifted to different theories to be evaluated accordingly with respect to its restrictions and limitations.

6.5.1 Preamble

In this section, we present some terms and concepts used in the proposed MCT algorithm. They are important for understanding the features and functionalities.

Query Context. We use the terms “query context" and “magic context" interchangeably. The following definition is implemented in the MCT algorithm using the function function $Cont(Q)$ that constructs the Magic Context $Q_{\mathscr{C}}$. Table 6.1 illustrates this construction.

Definition 8. *Let $Q = p(\overline{X})@C$ be a query in which C or at least one of the n arguments of p is bound. Let $i_1, \cdots, i_k$ be the positions of the bound arguments listed in $\overline{X}$, where $1 \leq i_k \leq n+1$, and the argument i_{n+1} refers to the context C when it is bound. Then, the context of Q, denoted as $Q_{\mathscr{C}}$, is defined as the set: $\{magic_{i_1} : [b_1], magic_{i_2} : [b_2], ..., magic_{i_k} : [b_k]\}$, where b_i is the ith bound value in Q. When C is a context name c, then $Q_{\mathscr{C}}$ includes $magic_{i_{n+1}} : [c]$.*

Relevant and Irrelevant Rules. Not every rule in a program is relevant to a query in the sense that it is needed to find the answers. This notion is formalized as follows.

Let Π_C be a *Contelog* program, Q be a query, and r be a rule in Π_C. We say that r is *relevant* to Q, if it is used in the derivations of some answer tuple. We refer to the set of all rules in Π_C that are relevant to Q as REL. Any other rule in Π_C is irrelevant to Q, referred to collectively as the set $IRREL$.

Leaf Predicate. A leaf predicate (LP) is a non-ground predicate that can be unified with a regular or contextual fact. We refer to it as leaf to express that this predicate is the first information needed to build subsequent knowledge towards answering a query. For instance, in path-edge program, the edge predicate is a leaf predicate. That is, the basic building block of retrieving all paths is first "knowing the edges".

Magic Predicate. A *magic predicate* (MP) is a non-ground predicate that unifies with query contexts $Q_{\mathscr{C}}$. It acts as the knowledge bridge between the query and the "relevant" tuples to derive all the answer tuples. The set $HEADS$ contains all MPs that are created in the rewriting process.

Magic Rule. Given a query Q, a *magic rule* (MR) defines a magic predicate in $HEADS$ using normal predicates in the program. Such a rule is of the form:

$$MP(\overline{Y})@A :- MP(\overline{X})@A,\ LP.$$

where $\overline{X}, \overline{Y}$ are arguments of the magic predicate with respect to the binding pattern of Q, and A is a context variable that refers to the magic context.

6.5.2 The MCT Algorithm

In this section, we present the MCT rewriting algorithm. We will also describe the steps of the algorithm using the following *Contelog* Program 6.1 as our running example.

Listing 6.1: Contelog program Π_C

```
CONTEXTS := {
    $c_b = \{t : [bird], fea : [can\text{-}fly]\}$,
    $c_a = \{t : [amph], fea : [can\text{-}swim]\}$.
**********************************
$a(parrot, bird)$.
$a(parakeet, parrot)$.
$a(toad, amph)$.
$r_1 : a(X,Y)@C$ :- $a(X,Y)$, $t(Y)@C$.
$r_2 : a(X,Y)@C$ :- $a(X,Z)@C$, $a(Z,Y)$.
$r_3 : f(X,Y)@C$ :- $a(X,Z)@C$, $fea(Y)@C$.
```

The algorithm takes as the input (1) a logic program, Datalog or *Contelog* , which consists of a set of facts and rules, and possibly contexts, and (2) a query $Q(a_1, a_2, \cdots, a_n)@C$ with possibly context. The output of the algorithm is a new logic program with extended and/or modified set of facts, rules, and contexts. The MCT algorithm is presented as follows.

The MCT Rewriting Algorithm

```
Input: A Contelog or Datalog program $\Pi$ and a query $Q$.
Output: A program $\Pi'$ that is equivalent to $\Pi$ with respect to $Q$.
Method:
1- Convert Q to a context: $Q_{\mathscr{C}} = Cont(Q)$.
2- Find the set REL of relevant rules in $\Pi$.
3- Generate the magic predicate corresponding to Q, and identify the set HEADS of
    the relevant head predicates.
4- Identify the Leaf predicates in REL.
5- Generate the set MRs of all magic rules.
6- Add all predicates in HEADS to each rule in REL.
7- Rewrite/Replace the program and return the result $\Pi' = MRs \cup REL$.
```

A detailed explanation of the steps of the proposed algorithm is as follows.

(1) Converting the query into a query context. In the first step, the input query Q with n arguments is converted to a query context $Q_{\mathscr{C}}$. This is done considering only the bound arguments in Q, for each of which, a new dimension is assigned in $Q_{\mathscr{C}}$ with the same bound value for the associated attribute. We consider only bound arguments of the query

Table 6.1: Examples of queries and the corresponding query contexts

Query Q	$Q_{\mathscr{C}} = Cont(Q)$
$p(X, 4)$	$\{magic_2 : [4]\}$
$p(1, X)$	$\{magic_1 : [1]\}$
$a(X, bird)@C$	$\{magic_2 : [bird]\}$
$a(parakeet, X)@C$	$\{magic_1 : [parakeet]\}$
$p(X, Y)@c_b$	$\{magic_{c_3} : [c_b]\}$
$p(1, Y)@c_b$	$\{\{magic_1 : [1]\}$, $\{magic_{c_3} : [c_b]\}\}$

because it is what we need to narrow down the space of reasoning towards only relevant tuples to those bound arguments. If i is the *index* of the bound attribute in the argument list of Q, we name the newly created dimension in $Q_{\mathscr{C}}$ as $magic_i$ or $magic\_i$. Thus, if Q has $k \leq n$ bound attributes, we will have k dimensions in $Q_{\mathscr{C}}$. A *Contelog* query may include a context, which could be a variable or constant. If the context in the query Q is a constant, we will create an additional dimension $magic_{c_{k+1}}$ or equivalently (magic_k+1_c).

The free variables or contexts in Q are ignored, as they do not contribute to query answers. We will then use the function $Cont(Q)$ that converts Q to the query context $Q_{\mathscr{C}}$. This step is explained in Example 9.

Example 9. *Consider the query $Q = p(a, X, Y)$. Since it has one bound attribute, a, we have $Q_{\mathscr{C}} = \{magic\_1 : [a]\}$. For query $Q = p(X, a, b)@c_1$, the second and third arguments are bound and the context c_1 is constant. In this case, we have $Q_{\mathscr{C}} = \{magic\_2 : [a], magic\_3 : [b], magic_{c_4} : [c_1]\}$. Table 6.1 shows $Q_{\mathscr{C}}$'s for different types of queries.*

(2) Finding the rules that are relevant to the query. We proceed in a top-down manner to determine the relevant rules.

(a) *Initialize* Set $REL = \emptyset$.

(b) *Update* Examine each rule r in the program, and update $REL \leftarrow REL \cup \{r\}$ if the functor of the head of rule r is equal to the functor of Q, and it has same number of arguments and contexts. If $REL \neq \emptyset$ then do the following steps.

(c) *Repeat:* Here, $REL \neq \emptyset$. For every $r \in REL$ repeat the following steps.

(d) *Termination* Regard each predicate p in the body of r as a new Q. Apply step (b) repeatedly for every predicate in r.

At this point, every rule that directly or indirectly contributes to the query is found and added to the set REL. See Example 10.

We remark that the above process terminates because there are finitely many rules and facts to consider.

Example 10. *Using our running Program 6.1, we show how the set REL of relevant rules is determined for the input query $Q^2_{C_1} = a(X, bird)@C$.*

Remove irrelevant predicates

```
# For the Contelog program $\Pi_C$:
1. initialize: REL = $\emptyset$
2. update: REL = $\{r_1, r_2\}$, since the head predicate of $Q^2_{C_1}$ matches the
    heads of $r_1$ and $r_2$.
3. Repeat: for each rule $r \in REL$, we examine the subgoals $A$ in the
    body of $r$.If there exists a rule $r_A$ whose head predicate matches
    the predicate of $A$, add $r_A$ to $REL$. This repeats until no more
    rules can be added. In this example, $REL = \{r_1, r_2\}$.
```

Following the steps above, we obtain the set of relevant rules. The set REL of relevant rules is identified by matching the predicate name, number of arguments, and the context if applicable. Next, we identify and remove from REL those rules in the program that are not useful considering the bound arguments of the query.

(3) Generating the magic predicate MP corresponding to the query. Using the query context, we create the corresponding magic predicate. Let $HEADS$ be the collection of such MPs used to restrict the evaluation of the rewritten program given the input query. See Example 11.

Example 11. *Let query be* $Q_{\mathscr{C}} = \{magic_1 : [1]\}$*, then the corresponding MP would be* $magic_1(X)@A$*, where* A *is a free context variable. Similarly, if query is* $Q_{\mathscr{C}} = \{magic_2 : [a], magic_3 : [b]\}$*, then the MPs we have are* $magic_2(X)@A$ *and* $magic_2(X)@A$*. The variable(s)* X *is just a variable now, it is clearly identified when used in the magic-rules introduced later as it is dependant on the binding pattern of the context query and the fact associated with it.*

(4) Finding LP in REL**.** The LP set contains the facts that are used to derive the answer tuples. For ease of illustration, suppose there is only one such fact. Recall that a *Contelog* program may have regular predicates and facts (as in Datalog) as well as contextual ones (as in the so-called context world). The purpose of this step is twofold. First, to know whether the origin of the knowledge is contextual or regular, which in turn is used to determine the dependency among the rules in order to find other relevant information. Second, to know which variable(s) to restrict in the original rules. To find LP facts, we consider the rules in REL and follow a top-down approach to determine a substitution θ until reaching a fact/context, explained as follows. This step is illustrated in Example 12.

Example 12. *Let* $Q : p(b_1, f_2, ..., f_n)@f_{n+1}$ *be a query, where* b *and* f*'s denote bound and free variables, respectively. Let* r_i *be the* i_{th} *rule in* REL *and suppose its head arguments are* $X_1, ..., X_n, C_{n+1}$*. Apply the substitution* $\theta = \{X_1/b_1, ..., X_n/f_n, C_{n+1}/f_{n+1}\}$ *to the head of* r_i*, and then propagate the substitution from the head to the body (left to right subgoals), until we reach a ground fact/context. Two cases may arise, depending on the kind of LP predicate encountered which could be regular or contextual.*

(5) Generating the magic rules. Use the magic predicates MPs and the Leaf predicates, and let DEP refer to the collection of both. Recall that an LP can be either regular or contextual. The main form of MR is the same for both, as follows:

$$MP(\overline{Y})@A\ :-\ MP(\overline{X})@A,\ LP$$

However, differences arise in the bindings $\overline{Y}$ and $\overline{X}$. In case of a regular predicate, the goal is to find information for a missing (free) argument, while for a context predicate, the goal is to find the name of the context for which an LP is found. This step is explained in Example 13.

(1) Case 1: *LP is regular.* Let this LP be $p(X_1, \cdots, X_n)$. Let Q be a query with the head predicate p, and with the sets of positions $B = \{b_1, ..., b_m\}$ and $F = \{f_1, \cdots, f_k\}$ of the bound and free arguments, respectively. There is a magic-rule in the following form for each bound argument in Q:

$$magic_i(\overline{Z})@A \text{ :- } magic_i(b_i)@A, p(\overline{Y}).$$

where, for a rule r_i, $\overline{Y}$ is the set of free variables as they appear in the LP. The variable b_i is free, with a symbol (variable or constant) similar to the symbol in LP that corresponds to the position of bound argument in Q which is being processed. The set $\overline{Z}$ includes the free arguments in $\overline{Y}$ excluding those used in b_i, where i is the position of the bound argument being processed.

If $\overline{Z}$ contains more than one argument, the process of breaking the arguments is repeated considering the new LP as $magic_i(Z)@A$, and terminates when the length of the set of free variables in $\overline{Z}$ becomes 1. For instance, if $p(X_1, X_2, X_3)$ is the LP appearing in a rule r_i and $Q = p(1, X_2, X_3)$, then $\overline{Y} = \{X_1, X_2, X_3\}$, $i = 1$, $b_1 = X_1$, and $\overline{Z} = \{X_2, X_3\}$. Since the length of $\overline{Z}$ is 2, the algorithm generates the following two magic rules.

$$magic_1(X_2, X_3)@A \text{ :- } magic_1(X_1)@A, p(X_1, X_2, X_3).$$

For the second magic rule, LP becomes $magic(X_2, X_3)@A$, and hence we have $\overline{Y} = \{X_2, X_3\}$, $b_1 = X_2$, and $Z = \{X_3\}$.

$$magic_1(X_3)@A \text{ :- } magic_1(X_2)@A, magic_1(X_2, X_3)@A.$$

This process is repeated for each bound argument, and each new rule generated is added to the set DEP. The predicate $magic_i(b_i)@A$ is also added to the set $HEADS$ for later use.

(2) Case 2: *LP is contextual.* This means that the information to be used to build up the required knowledge in the query is in some contexts. In this case, in addition to including the arguments contributing to the rule, the context information has to be collected and added to the rule. The resulting rule(s) will be included in DEP. The rewriting process in this case is exactly the same as in Case (1). The following rule is used to identify the contributing contexts to limit query evaluation by considering them only. The head predicate $magic\text{-}c_{i+1}(C)@A$ of this rule is added to $HEADS$.

$$magic\text{-}c_{i+1}(C)@A \text{ :- } magic\text{-}c_{i+1}(b_i)@A, p(\overline{Y})@C.$$

The rewriting algorithm, generates three sets: REL which contains all rules that are relevant to Q, DEP which contains all new rules required to build up the knowledge pertinent to Q, and $HEADS$ which contains the predicates that will be added to each rule in REL at the final step to restrict the evaluation process.

Example 13. *The steps of the LP computation for the query $Q^2_{C_1}$ over the* Contelog *program Π_C are shown below.*

Find LP for Π_C

```
# For $\Pi_C$,
$Q^2_{C_1} = a(X, bird)@C$
$Q_{\mathscr{C}} = \{magic_2 : [bird]\}$
$REL = \{r_1, r_2\}$
# 1- find LP (top-down unification)
```

```
r1: a(X,bird)@C :- a(X,bird), t(bird)@C.
# in the above case we see that {\em bird} can be a regular fact or
    part of a context, either of which could be considered. We consider
    the former a(X,bird).
#2- apply the dependency rule for regular facts
dpr1 : magic_2(X)@A :- magic_2(Y)@A, a(X,Y).
#3- add dpr1 to the set DEP. add new predicate magic_2(Y)@A to the set
    HEADS
# DEP = {dpr1}
# HEADS = {magic_2(Y)@A}

#If we use contextual fact in step(2). The rule would be
dpr1 :
magic_3_c(C)@A :- magic_3_c(Y)@A, t(Y)@C.
# similarly
# DEP = {dpr1}
# HEADS = {magic_3_c(C)@A}
```

(6) Modify the original program rules by adding the magic predicates to them. In this step, all predicates in the set $HEADS$ are injected at the beginning of each rule in REL.

(7) Rewrite/Replace original program with the new one. The new program is the union $REL \cup DEP$. The steps of the the rewriting performed are as follows.

(1) Copy the facts as provided in the original program.

(2) Add $Q_{\mathscr{C}}$ to the set of contexts of the program.

(3) Write all the rules in the set DEP.

(4) Write all the rules in the set REL.

The final rewritten programs for the input queries are shown below.

Π'_C for query $Q^2_{C_1}$

```
CONTEXTS = {
$c_b = \{t : [bird], fea : [can\text{-}fly]\}$,
$c_a = \{t : [amph], fea : [can\text{-}swim]\}$,
$Q_{\mathscr{C}}$ = {magic_2:[bird]}
*********
$a(parrot, bird)$.
$a(parakeet, parrot)$.
$a(toad, amph)$.
$dpr_1$ : magic_2(Y)@A :- magic_2(X)@A, a(X,Y).
$r_1$ : a(X,Y)@C :- magic_2(Y)@A, a(X,Y), t(Y)@C.
$r_2$ : a(X,Y)@C :- magic_2(Y)@A, a(X,Z)@C, a(Z,Y).
```

The MCT rewriting is based on the positions of the bound argument(s) in the input query. Therefore, for each different position of a bound argument in the query, the rewriting process explained above is repeated, and new set of dependency rules and corresponding magic predicates are added. However, if different queries are to be processed with the same bound argument position, it suffices to recompute only $Q_{\mathscr{C}}$. This makes the proposed rewriting to be flexible and dynamic as the context can be changed and new query can be submitted without repeating the rewriting.

6.5.3 Running Multiple Queries

Answering a query using a rewriting method requires to perform the rewriting rewriting to generate a program that is customized for the query. The idea to support "running multiple queries" in *Contelog* (as well as in Datalog) is to be able to generate a general rewritten program that can handle multiple queries efficiently without requiring to generate a new program for each query. This means the resulting of this general rewriting should be a highly optimized program that can customize itself to a posed query without performing changing the rewritten program.

Naturally existing query processing methods for Datalog are not developed to support contexts, and hence do not have the capability and flexibility of loosely coupled feature of *Contelog* by which contexts can be changed and evaluated without the need to to change the program. Using the highly optimized rewriting method of MCT, *Contelog* can generate a general program that can be evaluated for a query with any bindings and do so efficiently, as shown in the experimental results. To accomplish such a flexibility in Datalog, the program needs to be rewritten for each query with programmer involvement in general. This is because answering more than one query pattern needs different re-writings that need to be appended to each other. Hence, the rewritten program is slower to process (Tekle & Liu, 2011), harder to understand, and less optimized than the original program. *Contelog* being a conservative extension of Datalog, with the use of MCT we can achieve a dynamic universal program for both *Contelog* and Datalog.

In *Contelog* , the MCT rewriting algorithm is applied based on two conditions. (1) There is at least one bound argument in the query, and (2) the predicate name of the query and the position of the bound argument have not been processed before. If there is no bound argument, the rewriting terminates at the step of removing the irrelevant rules. In that case, it uses all the rules in REL to produce the query result. If the value of the bound attribute is changed only and not its position, then MCT rewrites only the context by applying the function $Cont()$ on query and re-contextualizing the query. In this case, it does not perform any rewriting on the rules/facts. This allows a program to evaluate multiple queries (each viewed as a context) within a single evaluation if all the queries use the same position of the bound argument. In Example 14, this point is illustrated, where two different queries are processed. The only extra step required is to compute $Q_{\mathscr{C}}$ twice, once for each new bound argument in the query. However, the rewriting of the program has to be done only once since the position of the bound argument of the query is the same. This implies that evaluation of the rewritten program will provide answers to both queries in the same single run of the program. Example 15 shows a complete universal program for a certain query, and shows the results that are produced with the same set of rules and only by changing the magic-context. Similarly, Example 16 illustrates how the same approach can be used for standard Datalog programs and allows it to have a universal program without repeating the rewriting for each distinctive binding pattern.

Example 14. *Let the query type be* $Q^2_{C_1}$*, and consider two such queries:* $a(X, bird)@C$ *and* $a(X, amph)@C$*. These queries respectively yield the contexts* $c_b = \{t : [bird], fea : [can\text{-}fly]\}$ *and* $c_a = \{t : [amph], fea : [can\text{-}swim]\}$*. Using the Cont function, we get the* *corresponding query contexts as* $q_{c_1} = \{magic_2 : [bird]\}$ *and* $q_{c_2} = \{magic_2 : [amph]\}$. *Using the above information, the rewritten program generated is as follows.*

Π'_C for query $Q^2_{C_1}$

```
a(parrot, bird).
a(parakeet, parrot).
a(toad, amph).
dpr_1 : magic_2(Y)@A :- magic_2(X)@A, a(X,Y).
a(X,Y)@C :- magic_2(Y)@A, a(X,Y), t(Y)@C.
a(X,Y)@C :- magic_2(Y)@A, a(X,Z)@C, a(Z,Y).
```

If we now have a query of the form $q(X, Y, Z)$, a rewriting is required only once for each position, that is, once for $q(a, Y, Z)$, once for $q(X, b, Z)$, and once for $q(X, Y, c)$. When we are able to combine these rewritings, we can answer all these queries without additional rewriting.

The bottom-up evaluation process in *Contelog* is monotonic as negation is not allowed in the rule bodies. Thus, the evaluation process is additive, in the sense that multiple programs (along with their contexts) can be combined and evaluated simultaneously to produce the query results. This capability is formalized as follows.

Definition 9. *Let* $\{\Pi_1, ..., \Pi_n\}$ *be a finite set of* Contelog *programs, and let* $R()$ *denote the function that runs these programs and return the results. Hence,*

$$R(\bigcup_{i=1}^{n} \Pi_i) = R(\Pi_1) \cup R(\Pi_2) \cup ... \cup R(\Pi_n)$$

Using the definition above, we can say that a *universal program* is the union of program rewritings such that for each unique binding of the query, there is a single rewriting present in this union.

Definition 10. *For a* Contelog *program* Π_C *and a query* Q^n_C *with* n *arguments, let* $MCT(\Pi_C, Q^i_C)$ *denote the MCT rewritten program with respect to the query* Q^i_C. *A universal program for* Q^n_C*, denoted as* Π'^n_C*, is the defined as:*

$$\Pi'^n_C = \bigcup_{i=1}^{n} MCT(\Pi_C, Q^i_C)$$

Using the above definitions, the program in Example 15 is a universal program for the running program 6.1 introduced earlier.

Example 15. *Let $f(X,Y)@C$ be the query for which we want to generate the universal program. For this, we need to generate and combine three rewritings, one for X, one for Y, and one for C. Once we combine these three rewritings into a single universal program, we only change the magic-context query to limit the evaluation of the resulting program to different set of facts. This yields the following universal program:*

```
magic_1(Y)@A :- magic_1(X)@A,a(X,Y).
magic_1(Z)@A :- magic_1(X)@A,a(X,Z).
a(X,Y)@C :- magic_1(X)@A,a(X,Y),type(Y)@C.
a(X,Y)@C :- magic_1(X)@A,a(X,Z),a(Z,Y)@C.
f(X,Y)@C :- magic_1(X)@A,a(X,Z)@C,f(Y)@C.
magic_2_c(C)@A :- magic_2_c(Y)@A,f(Y)@C.
a(X,Y)@C :- magic_2_c(C)@A,a(X,Y),type(Y)@C.
a(X,Y)@C :- magic_2_c(C)@A,a(X,Z),a(Z,Y)@C.
f(X,Y)@C :- magic_2_c(C)@A,magic_2_c(Y)@A,a(X,Z)@C,f(Y)@C.
a(X,Y)@C :- magic_3_c(C)@A,a(X,Y),type(Y)@C.
a(X,Y)@C :- magic_3_c(C)@A,a(X,Z),a(Z,Y)@C.
f(X,Y)@C :- magic_3_c(C)@A,a(X,Z)@C,f(Y)@C.
```

Now, for the above program consider the following set of contexts and facts:

```
#CONTEXTS
cbird={type:[bird],f:[canfly]}
camph={type:[amph],f:[canswim]}
#FACTS
a(parrot,bird).
a(frog,amph).
a(parakeet,parrot).
a(parrotlet,parrot).
a(que,parakeet).
a(echo,parrotlet).
a(tods,frog).
```

Consider the query $f(X, canfly)@C$. The magic context for this query is $magic\_context = \{magic_{2c} : [canfly]\}$, which is appended to the set of contexts. The evaluation of the resulting program yields the following tuples:

```
**magic_2_c(cbird)@magic_context.
**a(parrot,bird)@cbird.
**a(parakeet,bird)@cbird.
**a(parrotlet,bird)@cbird.
**f(parrot,canfly)@cbird.
**a(que,bird)@cbird.
**a(echo,bird)@cbird.
**f(parakeet,canfly)@cbird.
**f(parrotlet,canfly)@cbird.
**f(que,canfly)@cbird.
**f(echo,canfly)@cbird.
```

Note that the results are only relevant to all birds as they all can fly. Also note that frogs are not shown in the result because they cannot fly according to the context. Now, if we change the query to $f(parrot, X)@C$, then without needing additional rewriting but only changing the magic-context to $magic\_context = \{magic_1 : [parrot]\}$, we obtain the desired results:

```
**magic_1(bird)@magic_context.
**a(parrot,bird)@cbird.
**f(parrot,canfly)@cbird.
```

The same program can also be used to answer the query $f(X, Y)@cbird$, which would return all the facts that are true at a specific contexts. Using magic-context, we are able to change in the programs above the context and achieve different results without additional rewriting. Consequently, the MCT rewriting method proposed yields highly efficient Contelog *programs without the need for a new rewriting for every new query.*

Example 16. *The same approach used in Example 15 applies for standard Datalog programs. This is illustrated in the following "same generation cousins" program.*

```
sgc(X,Y) :- sib(X,Y).
sgc(X,Y) :- par(X,Z), sgc(Z,Z1), par(Y,Z1).
```

The universal program after running the rewriting method for each binding once, and combining their results is given below:

```
magic_2(X)@A :- magic_2(Y)@A,sib(X,Y).
magic_2(Z1)@A :- magic_2(Y)@A,par(Y,Z1).
sgc(X,Y) :- magic_2(Y)@A,sib(X,Y).
sgc(X,Y) :- magic_2(Y)@A,par(X,Z),sgc(Z,Z1),par(Y,Z1).
magic_1(Y)@A :- magic_1(X)@A,sib(X,Y).
magic_1(Z)@A :- magic_1(X)@A,par(X,Z).
sgc(X,Y) :- magic_1(X)@A,sib(X,Y).
sgc(X,Y) :- magic_1(X)@A,par(X,Z),sgc(Z,Z1),par(Y,Z1).
```

Now, for the above program consider the following set of facts:

```
# FACTS
sib(john,sole).
sib(chole,rams).
par(john,rams).
par(sole,rams).
par(rams,frank).
par(chole,frank).
par(charl,chole).
```

The original complete set of results for this program is as follows:

```
**sgc(john,sole).
**sgc(chole,rams).
**sgc(charl,john).
**sgc(charl,sole).
```

Consider the query $sgc(john, X)$*. The results for this query would include tuples that are relevant to john. The magic-context for this query is* $magic\_context = \{magic\_1 : [john]\}$*, for which the program evaluation is restricted and return the following desired tuples:*

```
**magic_1(sole)@magic_context.
```

```
**magic_1(rams)@magic_context.
**sgc(john,sole).
**magic_1(frank)@magic_context.
```

Similarly, for the same program but the query $sgc(X, john)$, the evaluation result as expected are as follows:

```
**magic_2(rams)@magic_context.
**magic_2(chole)@magic_context.
**magic_2(frank)@magic_context.
**sgc(chole,rams).
**sgc(charl,john).
```

We remark that the ”multiple query" feature in *Contelog* is interesting and advantages in two aspects. First, it is based on an optimized rewriting (minimal) for each binding pattern. Second, with the same input facts and rules and by changing only the context set in the program, we obtain desired tuples true in the contexts specified. Besides, changes in the query context changes the reasoning space restricted to specific facts and rules in the program.

Chapter 7

Contelog: Case Studies

In this chapter, we present several case studies to highlight on the simplicity, expressiveness, and efficiency features of *Contelog* . In the first two case studies, the focus is on inheritance and expressiveness. The third case illustrates simplicity and efficiency of *Contelog* in terms of program size and use of contexts. The last case study demonstrates practical aspects of *Contelog* as an active reasoner (online) in building a context-aware engine as opposed to just being a passive reasoner (offline). More examples and applications are provided in Appendix A.

7.1 Building Locator Program Example

The following is the "building locator" *c-program* P_{bl}, introduced earlier, following by a step-by-step illustration of its fixpoint evaluation results.

The contexts for P_{bl}

c_e = $\{from : [east], to : [right]\}$.

c_w = $\{from : [west], to : [left]\}$.

c_n = $\{from : [north], to : [straight].\}$

The facts and rules for P_{bl}

$f_1 : p(1, east)$.

$f_2 : p(2, west)$.

$f_3 : p(3, north)$.

```
f_4 : p(4, east).
f_5 : p(5, north).
r_1:  p(X,Y)@C  :-  p(X,Y),  from(Y)@C.
r_2:  b(X,Y)@C  :-  p(X,Z)@C, to(Y)@C.
```

The set $T^i_{bl}()$ of tuples derived by the fixpoint evaluation of this program at each step i is shows as follows.

$$T^0_{P_{bl}}(\phi) = \{p(1, east), p(2, west), p(3, north), p(4, east), p(5, north)\}$$
$$T^1_{P_{bl}}(T^0_{P_{bl}}) = T^0_{P_{bl}} \cup \{p(1, east)@c_e, p(2, west)@c_w, p(3, north)@c_n, p(4, east)@c_e, p(5, north)@c_n\}$$
$$T^2_{P_{bl}}(T^1_{P_{bl}}) = T^1_{P_{bl}} \cup \{b(1, right)@c_e, b(2, left)@c_w, b(3, straight)@c_n, b(4, right)@c_e, b(5, straight)@c_n\}$$
$$T^3_{P_{bl}}(T^2_{P_{bl}}) = T^2_{P_{bl}} \quad \text{(fixpoint is reached)}$$

Thus, the result returned when the fixpoint evaluation of P_{bl} terminates is the following set of tuples: $\{p(1, east), p(2, west), b(3, north), p(1, east)@c_e,$
$p(2, west)@c_w, p(3, north)@c_n, p(4, east)@c_e, p(5, north)@c_n$
$b(1, right)@c_e, b(2, left)@c_w, b(3, straight)@c_n, b(4, right)@c_e, b(5, straight)@c_n\}$.

7.1.1 Magic Context: Query Processing for Building Locator program P_{bl}

We next consider evaluation of the *c-program* P_{bl} for the following three c-queries:

- $Q^1_{\mathscr{C}} = b(1, X)@C$: asks for the position (left, right, straight) of the library for a person "1" approaching the building from X (east or west) X in context C.
- $Q^2_{\mathscr{C}} = b(X, straight)@C$: asks about the people directed to "straight" with the name of the person and the context as variables.
- $Q^{1,3}_{\mathscr{C}} = b(3, Y)@c_n$: asks about person "3" in context c_n regardless of the direction.

For the c-query $Q^1_{\mathscr{C}} = b(1, X)@C$**,** the contexts and rules of the rewriting of P_{bl} as well as the evaluation results are as follows. Note that for this case and also the other c-queries below, the set of facts used in the evaluation of the program is the same as defined initially, and the output shown are the derived tuples that are relevant to the corresponding query.

```
$c_e$ = $\{from : [east], to : [right]\}$.
$c_w$ = $\{from : [west], to : [left]\}$.
$c_n$ = $\{from : [north], to : [straight]\}$.
magic_context={magic_1:[3].}
```

```
magic_1(Y)@A :- magic_1(X)@A, p(X,Y).
p(X,Y)@C :- magic_1(X)@A, p(X,Y), from(Y)@C.
b(X,Y)@C :- magic_1(X)@A, p(X,Z)@C, to(Y)@C.
```

```
**magic_1(east)@magic_context.
**p(1,east)@ce.
**b(1,right)@ce.
```

For the c-query $Q^2_{\mathscr{C}} = b(X, straight)@C$**,** the contexts and rules generated by the rewriting of P_{bl} as well as the output tuples returned are as follows.

```
# other contexts are unchanged
# ''c" in the dimension name indicates that this information comes from context.
magic_context = {magic_2_c:[straight]}
```

```
magic_2_c(C)@A :- magic_2_c(Y)@A, to(Y)@C.
p(X,Y)@C :- magic_2_c(C)@A, p(X,Y), from(Y)@C.
b(X,Y)@C :- magic_2_c(C)@A, magic_2_c(Y)@A, p(X,Z)@C, to(Y)@C.
```

```
**magic_2_c(cn)@magic_context.
**p(3,north)@cn.
**p(5,north)@cn.
**b(3,straight)@cn.
**b(5,straight)@cn.
```

For the c-query $Q_{\mathscr{C}}^{1,3} = b(3, Y)@c_n$, the contexts and rules generated by the rewriting of P_{bl} as well as the output tuples returned are as follows.

```
# other contexts are unchanged
# ''c" in the dimension name indicates that this information comes from context.
# Two dimensions are used: one points to first argument magic_1 and the
    otherpoints to the context.
magic_context = {magic_1:[3],magic_3_c:[cn]}
```

```
# to limit the reasoning, two MP predicates are used, one for each dimension
    (bound argument).
magic_1(Y)@A :- magic_1(X)@A, p(X,Y).
p(X,Y)@C :- magic_3_c(C)@A, magic_1(X)@A, p(X,Y), from(Y)@C.
b(X,Y)@C :- magic_3_c(C)@A, magic_1(X)@A, p(X,Z)@C, to(Y)@C.
```

```
**magic_1(north)@magic_context.
**p(3,north)@cn.
**b(3,straight)@cn.
```

The above program is a simple navigation example that can show case the expressiveness and simplicity of *Contelog* to solve context-based reasoning problems. The same settings can be taken to a larger scale scenarios to solve smart-rooms, smart-gates, and other context-based decision problems. In addition, through “universal programs" explained in Section 6.5.3, multiple and concurrent queries can be processed and answered simultaneously without increasing the complexity of the program.

7.2 Access Control Program Example

This example illustrates an application of *Contelog* to support file system access control in the contexts of users. The dimensions in the context include *types* of users and the access privileges associated with each type. A user may have more than one type and hence he/she may inherit more than one privilege based on the contexts. In this example, the context defines the role of the users in a computer system. Therefore, a context in this application has three dimensions: (1) *t*ype: describes the type of the users. (2) *p*riv: describes the privilege of a user of a particular type, and (3) *c*om: describes the privilege common in all contexts. There are two main contexts from the users' perspective: the *a*dmin context c_a and *r*egular context c_r. The greatest lower bound (meet) of the contexts lattice is c_g and their least upper bound (join) defines the super-user context c_s. The context and other information of the access control program P_{ac} in *Contelog* are shown below.

Access control program P_{ac}

```
# Contexts
$c_a = \{type(admin), priv(admin\text{-}priv), com(guest)\}.$
$c_r = \{type(regular), priv(regular\text{-}priv), com(guest)\}.$
# Note also the implied contexts: $c_g = c_a \odot c_r$ and $c_s = c_a \oplus c_r$.

# Facts
$f_1: u(john, admin).$
$f_2: u(john, user).$
$f_3: u(rose, guest).$

# Rules
$r_1: u(X,Y)@C$ :- $u(X,Y), type(Y)@C.$ # to select context according to type
$r_2: u(X,Y)@C$ :- $u(X,Y)@C, u(X,Z)@W.$ # if one user has more than one context
$r_3: priv(X,Y)@C$ :- $u(X,Z)@C, priv(Y)@C.$ # to infer privilege based on context
$r_4: superpriv(X,Y)@M$ :- $priv(X,Z)@C, priv(X,Y)@W.$
# according to formula #11 in Table 1, it derives superpriv(X,Y) as the join of $C$
    and $W$.
$r_5: guest(X,Y)@M$:- $u(X,Y), com(Y)@C, com(Y)@W$
# according to formula #8 in Table 1, this derives guest(X,Y) as the meet of $C$
    and $W$.
```

The bottom-up evaluation of P_{ac} is described as follows:

$$T^0_{P_{ac}}(\phi) = \{u(john, admin), u(john, user)$$
$$, u(rose, guest)\}$$

Using rule r_1 and contexts c_a and c_r, the evaluation derives the contexts of users. Also using r_5, we derive $guest(rose, guest)@cg(c_a \odot c_r)$, and hence:

$$\begin{aligned} T^1_{P_{ac}}(T^0_{P_{ac}}) = T^0_{P_{ac}} \cup \\ & \{u(john, admin)@c_a, u(john, user)@c_r, \\ & guest(rose, guest)@c_g\} \end{aligned}$$

Since users are inferred in contexts, the rules r_2 and r_3 are triggered,

$$\begin{aligned} T^2_{P_{ac}}(T^1_{P_{ac}}) = T^1_{P_{ac}} \cup \\ & \{priv(john, admin\text{-}priv), \\ & priv(admin, regular\text{-}priv)\} \end{aligned}$$

Rule r_3 infers privileges of users, which in turn triggers firing rule r_4. We thus have:

$$\begin{aligned} T^3_{P_{ac}}(T^2_{P_{ac}}) = T^2_{P_{ac}} \cup \\ & \{superpriv(john, admin\text{-}priv)@c_s \\ & where\ (c_s = c_a \oplus c_r)\} \end{aligned}$$

$T^4_{P_{ac}}(T^3_{P_{ac}}) = T^3_{P_{ac}}$ (fixpoint is reached)

The least fixpoint evaluation of P_{ac} returns the collection of the tuples mentioned above:

$$\begin{aligned} & \{u(john, admin), u(john, user), u(rose, guest), \\ & u(john, admin)@c_a, u(john, user)@c_r, priv(john, adminpriv), \\ & priv(admin, employeepriv), guest(rose, guest)@c_g, \\ & superpriv(john, adminpriv)@c_s\} \end{aligned}$$

7.2.1 Magic Context: Query Processing for Access Control Example

Consider the query $Q^2_{\mathscr{C}} = priv(X, admin)@C$, which asks for all employees with the privilege "admin". The initial contexts and the one defined by the query are shown below, followed by the rewritten *c-program* generated by the proposed MCT algorithm.

New Program For c-query $Q^2_{\mathscr{C}} :: priv(X, admin)@C$

```
# Contexts
ca = {type:[admin],priv:[adminpriv],com:[guest].}
cr = {type:[regular],priv:[regularpriv],com:[guest].}
magic_context = {magic_2:[admin].}
```

```
# Facts
u(john,admin).
u(john,regular).
u(rose,guest).
u(roy,admin).
u(roy,regular).
# Rules - irrelevant rules are discarded by the algorithm.
u(X,Y)@C :- magic_2(Y)@A,u(X,Y), type(Y)@C.
u(X,Y)@C :- magic_2(Y)@A,u(X,Y)@C, u(X,Z)@W.
priv(X,Y)@C :- magic_2(Y)@A, u(X,Z)@C, priv(Y)@C.
```

```
# Results
**u(john,admin)@ca.
**u(roy,admin)@ca.
```

User authentication is a mandatory component for a wide range of applications. This authentication is based on user roles and privileges which may change at run-time depending on the user's context. For instance, a user in building A can access certain files, and a user who has been promoted to a position X can have access to certain data. All these changes need to be secured, verified, and assured to maintain security and safety of the data being shared among the staff/users of a system. As illustrated in the above example *Contelog* is a declarative, simple, and yet powerful framework to support such applications efficiently.

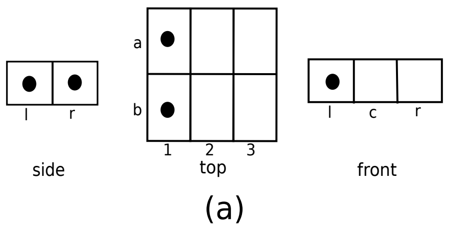
7.3 Simple Magic Box Example Program

This example is used in different literature (Akman & Surav, 1996; Benerecetti, Bouquet, & Ghidini, 2000) to bring out the importance and motivate contextual reasoning. We use this example here to illustrate (1) practicality, (2) flexibility, (3) simplicity, and (4) expressive power of *Contelog* programming. The example describes a box that is divided from top into 2×3 matrix, as shown in Figure A.9(b). It considers only three faces of the box, namely the *top, front, and side*. Each face of the box has its own view which is analogous to view points. Figure A.9(a) illustrates the actual view with respect to each those faces of the box. The following list establishes some notation to describe the box:

- l: means left square
- c: means center square
- r: means right square

The top face is described as a matrix, where a_1 refers to the top left corner, and b_1 refers to the bottom left corner, and so on. To understand the relative positioning of the squares with respect to three faces, say with respect to b_1, we see from the top it is b_1, from front it is l, and from the side it is r. The rule in the game is that the box contains only two balls and someone can see the balls from any face in any square unless it is blocked by the second ball as it is the case in the front side.The idea is to figure out the position of the balls on the top view of the box by only knowing the position of the balls on the side and the front faces.

In (Benerecetti et al., 2000), authors used a propositional logic-based framework, called MCS, to model and solve this problem. A program they define express all possible situations of the balls and boxes as a list of rules. They define two rules for each square in the face stating where the wall would be from the top view. That is, for the left square of the side face, the rules define two cases: whether the ball 'exist' or it 'does not exist'. A snippet of MCS example for the side face is shown below:



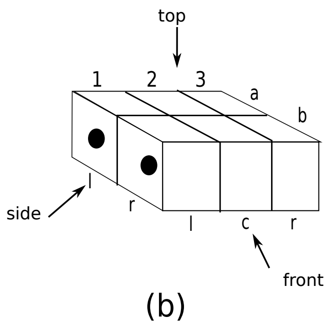


Figure 7.1: Illustration of the magic box example

MCS Magic Box V(1) Example

(1)$ist(side, l) \iff ist(Top, “(a_1 \vee a_2 \vee a_3)”)$

(2)$ist(side, \neg l) \iff ist(Top, “\neg(a_1 \vee a_2 \vee a_3)”)$

(3)$ist(side, r) \iff ist(Top, “(b_1 \vee b_2 \vee b_3)”)$

(4)$ist(side, \neg r) \iff ist(Top, “\neg(b_1 \vee b_2 \vee b_3)”)$

The balls are presented as a set of facts “$\{l, r\}$". The above rules have “hard coded” all possible situations, and can deal with any number of balls within the same structure/number of boxes. However, if the number of boxes changes, which we will examine in the next example, the number of rules increases exponentially.

The goal of this example is to illustrate contextual information and reasoning to solve the puzzle. The information in this example is the three faces (regarded as view points), namely, top, front, and side. Since top is inferred, only front and side are explicitly expressed in the following description.

Magic Box Example

```
side = {'l':['ball'], 'r':['ball']}.
front = {'l':['ball']}.
top = {}.
```

A context here describes the location of a ball, or to be more specific, which squares contain balls. For instance, in the context `front`, dimension 'l'describes the left square of the face, and its attribute is 'ball', that asserts it contains a ball. The other positions are not specified since they contain no balls. The rules that define each square of the top face are laid out below.

Magic Box Example Program in

```
f1 : s(side).
r1: a1(X)@top :- l(X)@C, l(X)@W, C!=W,s(C).
r2: a2(X)@top :- l(X)@C, c(X)@W, C!=W,s(C).
r3: a3(X)@top :- l(X)@C, r(X)@W, C!=W,s(C).
r4: b1(X)@top :- r(X)@C, l(X)@W, C!=W,s(C).
r5: b2(X)@top :- r(X)@C, c(X)@W, C!=W,s(C).
r6: b3(X)@top :- r(X)@C, r(X)@W, C!=W, s(C).
```

Rule r_1 is used to infer whether or not there is a ball in the square $a1$ in the top face. It asserts: if there is a ball in the left of the side face and there is one in the left of the front face, then a ball is located in a_1. Other rules are similar but differ in positions of the squares. A rule is defined for each square in the top face, and hence there are six rules. The inequality $C! = W$ is to look at different faces. Also, the fact $s(side)$ is to restrict the context variable C to the side face. This will ensure that the context variables in the rules refer to desirable contexts. The following is the result of executing this program which shows the squares in the top face that contain balls.

Results of Magic Box Example 2-balls

```
{s(side), a1(ball)@top, b1(ball)@top}
```

Now, let us increase the context $front = \{l : [ball], c : [ball]\}$, which means we are adding one more ball. The result of the fixpoint evaluation of the modified program is as follows:

Results of Magic Box Example 3-balls

```
# note that the results context show the location of the ball in the top view
    context, i.e., in square-a1, square-b1, and square-b2.
{s(side), a1(ball)@top, b1(ball)@top, b2(ball)@top}
```

7.3.1 Magic Box Example - Scaled Up Version

In this section, how *Contelog* help to provide a solution to the magic box example. We will also illustrate the scalability advantage of *Contelog* over the MCS solution as the number of boxes increases in the range 6 to 12, assuming that the rules of the game remains the same. Refer to Figure A.10 for a scaled-up version of the problem by a factor of 2.

Note that if we use the same concept introduced in (Benerecetti et al., 2000), we need eight rules to express the side face, since each square in the face requires two rules, i.e., for existing and non-existing cases. The top left square of the side face is expressed by the following two rules:

MCS code for the Magic Box Example Version 2 (V2)

$$(1)\, ist(side_t, l) \iff ist(Top, ``(a_1 \vee a_2 \vee a_3)")$$
$$(2)\, ist(side_t, \neg l) \iff ist(Top, ``\neg(a_1 \vee a_2 \vee a_3)")$$

This also increases the size of the solution to the problem by a factor of 2, which is expected for limited expressive disadvantage of the propositional logic. However, using *Contelog* , we only need to make a few changes in the program and the context modules, explained as follows. First, we add two more contexts, $side_b$ and $front_b$. This is needed because a new layer in the box has been added, and *Contelog* uses contexts to describe boxes. If we have three layers, then three types of context should be defined for each fact. Also, some facts are added to restrict the unification process within the rule. For instance, the facts $s(side_t)$ and $s(side_b)$ are added so that the predicate $s(C)$ in the rules helps in restricting the domain of variable C to the contexts $side_t$ and $side_b$. The context and program modules are shown below.

The facts and rules of the Magic Box Example V2

```
# Contexts
side_t={r:[ball]}.
side_b={r:[ball]}.
front_t={l:[ball]}.
front_b={c:[ball]}.
top = {}.
```

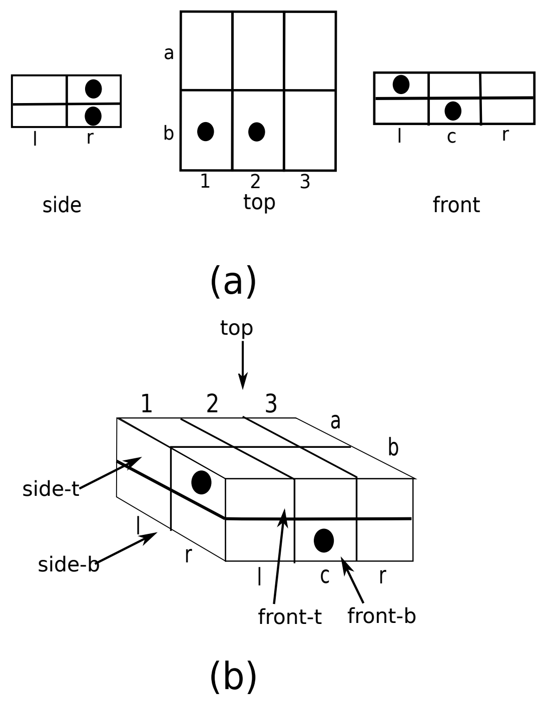


Figure 7.2: Illustration of the scaled-up magic box example

Magic Box Example V2 i *Contelog*

$f_1 : s(side_t).$

$f_2 : s(side_b).$

$f_3 : f(front_t).$

$f_4 : f(front_b).$

r_1: $a1(X)@top$:- $l(X)@C, l(X)@W, C! = W, s(C), f(W).$ `# s(C) and f(W) are added to`

r_2: $a2(X)@top$:- $l(X)@C, c(X)@W, C! = W, s(C), f(W).$ `# restrict the contexts to be`

r_3: $a3(X)@top$:- $l(X)@C, r(X)@W, C! = W, s(C), f(W).$ `# unified with the variable.`

r_4: $b1(X)@top$:- $(X)@C, l(X)@W, C! = W, s(C), f(W).$ `# letter s and f are used for`

r_5: $b2(X)@top$:- $r(X)@C, c(X)@W, C! = W, s(C), f(W).$ `# the side and front contexts,`

r_6: $b3(X)@top$:- $r(X)@C, r(X)@W, C! = W, s(C), f(W).$ `# respectively.`

Results of the Magic Box Example V2

`# Results`

$\{s(side_t),\ s(side_b),\ f(front_t),\ f(front_b),\ b1(ball)@top,\ b2(ball)@top\}$

We remark that using MCS, the number of rules required to express the above example compared to the number of rules required to solve the simple scenario shown in "scaled-up magic box A.11" in (Benerecetti et al., 2000) indicate the problem of exponential growth by MCS solution, whereas when using *Contelog* to solve the same problem, we simply add two context declarations without changing the rules. This will then show scalability advantage of *Contelog* over alternative solutions to handle large data sets.

7.3.2 Magic Context: Query Processing for Scaled-up Version of the Box Example

We now illustrate the MCT rewriting technique for the Magic Box problem when a c-query provided. This example clearly shows suitability and effectiveness advantage of MCT Magic with magic-contexts for producing programs of sizes similar to the original program. in comparison to other rewriting methods.
Consider the query $Q^1_{\mathscr{C}} = b1(ball)@C$, which asks to find the balls in box (b1) regardless of the context. The context and rule modules of the resulting *c-program* is provided below together with the output tuples.

New Program For c-query $Q^1_{\mathscr{C}} = b1(ball)@C$

```
# Context: same as the contexts in the original program.
sidet={r:[ball]}.
sideb={r:[ball]}.
frontt={l:[ball]}.
frontb={c:[ball]}.
top={box:[noball]}. # an initialization value
magic_context = {magic_1:[ball]}
```

REWRITTEN PROGRAM:

```
# Facts
s(sidet).
s(sideb).
f(frontt).
```

```
f(frontb).
# Rules - There is only one rule relevant to the query; the rest are discarded by
    MCT.
b1(X)@top :- magic_1(X)@A,r(X)@C,c(X)@W,C!=W,s(C),f(W).
```

```
# Results
**b1(ball)@top.
```

The significance of solving the magic-box problem using *Contelog* is twofold. First, it shows how intuitive it is to solve a well-known problem considered in related literature. Second, it shows how simple it is to take the problem to the next level, scale it up by increasing the number of layers and/or balls, while the size of the resulting *c-program* literally remain minimal. Another remark to add here is that the examples presented in this chapter demonstrate capability of *Contelog* to handle different application scenarios with ease due to its modular and loosely coupled features, allowing to evaluate programs with different contexts efficiently. In the following section we further illustrate suitability of *Contelog* in different applications.

7.4 Context Aware Application - Active Reasoner

Contelog inference engine is a passive-reasoner due to two main reasons. The first reason is the fact that it is monotonic, as negation is not allowed in *Contelog* . This is because active reasoners maybe faced with changes in the input facts, which in turn affects the derivations of output tuples being dependent on the order of facts/rules fired. The second reason is that its evaluation process is bottom-up. That means the program evaluation must complete and terminate to know what facts to be inferred. This could be prohibitive in terms of running time for real-life applications with high volume of data, when instant response is expected by users in an active fashion. Therefore, in general, logic-based knowledge bases may be considered as or believed not to be suitable for context-aware applications and systems. The *Contelog* experience suggests the belief may not be true anymore, argued as follows.

Due to its loose-coupling and magic-context features, *Contelog* can be used as an active context-aware system. The loose-coupling between facts/rules and contexts makes it

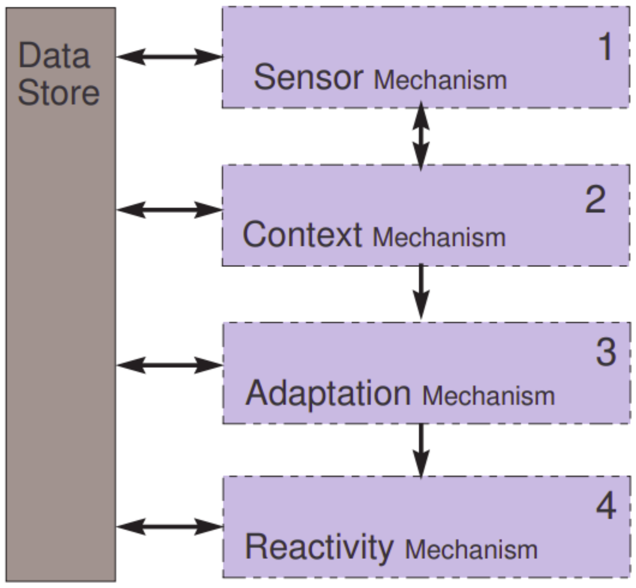


Figure 7.3: Context-aware General Structure (Alagar et al., 2014a)

possible for the application to easily add new contexts and generate new facts/actions without changing the program. Thus, monotonicity is maintained, yet conflicting facts can be present and true in different contexts. This indicates a major advantage of *Contelog* over non-contextual, logic-based frameworks and engines. The MCT query optimization method developed for *Contelog* provides significant performance improvements, in particular when the selectivity ratio of the input query is high. That is, when the number of desired answer tuples to the number of all tuples is small. Finally, the features of supporting multiple queries, combined programs, and local dynamism of *Contelog* discussed earlier in Chapters 6, 3 allow collaboration of multiple *Contelog* engine instances to arrive at single goal for an application. Figure 7.3 is the general structure of context-aware systems adopted from (Alagar et al., 2014a) that we will use in our case study. In Section 7.4.1, we briefly introduce the components of a context-aware system for *Contelog* . In Section 7.4.2, we show the components of *Contelog* based context-aware system that performs “tilt detection and adaptation". In the remaining subsections we discuss the implementation aspects of the tilt detection systems.

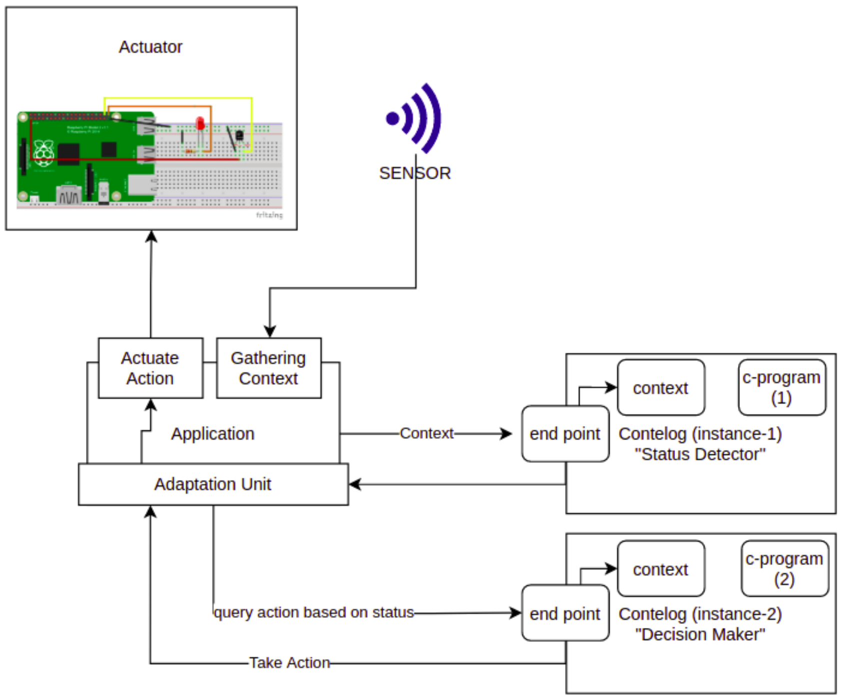


Figure 7.4: Context-aware main components with *Contelog*

7.4.1 Context-Aware System Components

Following the architecture proposed in (Alagar et al., 2014a) for context-aware systems, we use *Contelog* to present the components of such system. As any context-aware system, there are main components that were developed/assembled to accomplish the task. Those components are as follows (illustrated in Figure A.11):

(1) Sensor: This is the device responsible for monitoring the contextual changes in an environment.

(2) Actuator: This unit is the electronic circuit that carries out the adaptation actions, such as open gate, start the alarm, etc.

(3) *Contelog* Engine Instances: One or more instances of *Contelog* that infers decisions according to the input context, and facts/rules of the system.

(4) Application (Control Unit): This unit orchestrates the actions/information among sensors, actuators, and *Contelog* instances.

(5) Application (Adaptation Unit): This unit decides the adaptation actions based on the incoming data from *Contelog* programs.

7.4.2 Contelog-based Context-aware System: Tilt Detector

This context-aware system detects inclination in surfaces using a Mercury-based Sensors to read the tilting, and sends it to the control unit to take the proper action. Those type of systems are very critical in several applications (Fraden, 2004). Some details are as follow.

- Roll Sensing: Inclination management using sensors provides a roll over or tip over warning for applications like construction equipment and lift vehicles that operate in rugged terrains.
- Automotive uses: Sensing inclination has been used by Automobile manufacturers for lighting controls (for example, trunk lid lights), ride control (horizontal and vertical inclination), and anti-lock braking systems.
- Fall alarms: Work performed in confined space (such as a welder inside a tank) raises special safety concerns. Such context-aware systems are used to sound an alarm if a worker falls over.
- Bombs: A slight tilt can trigger a bomb. Context-aware systems are used to sense and trigger safety action in case of this inclination.

Although the system idea is simple, it has far more crucial applications and usages. This motivated us to use *Contelog* , a logic-based framework to monitor such changes in contexts and infer the recommended actions to take. The beauty of using *Contelog* is that it enjoys a sound and complete proof procedure with inferred results being verifiable as well as explainable. The latter is not elaborated on here but is important when the application include large number of rules and facts.

7.4.3 Tilt Detector: The Electronic Circuit

We have implemented the circuit using Raspberry Pi Model B Microcomputer. The complete circuit is provided in Figure A.12. It consists of four main components explained as follows:

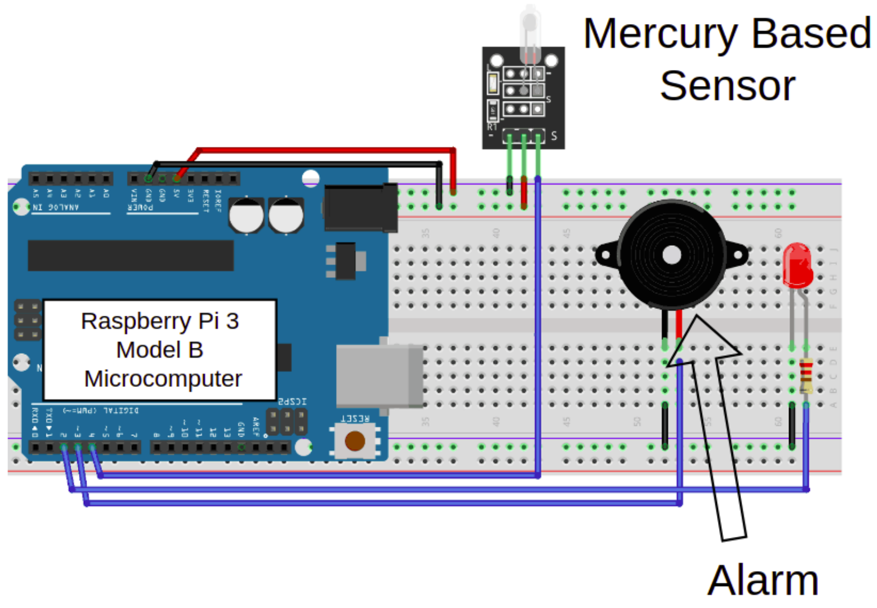


Figure 7.5: Circuit for Tilt Detector

- The Microcomputer: It represents the controller where the logic of the application, controlling, and adaptations are performed. The application is programmed using python. Full implementation of the controlling unit is provided in Appendix B.
- Alarm: A device that produces sound when instructed by the controller, which is basically when the tilt sensor is tilted.
- LED: This is to indicate another danger, used to trouble shoot in case a signal is coming from the sensor but still the alarm does not go-off.
- Sensor: As explained earlier, Mercury-based tilt sensor to sense any inclination in surfaces.

7.4.4 Tilt Detector: The System Logic and Sequence

The controller constantly reads signals from the sensor. In case, a tilt/change is read, it communicates with the first *Contelog* instance, to which we refer as “status detector". Based on the “status detector” results, the controller communicate with the second *Contelog* instance, that we call “decision maker”, in order to make the proper action. The communication between the control unit, and the two instances of *Contelog* is through a RESTful

API that interfaces through web services. After receiving the results from the "decision maker", there are two actions that can be made, either to set or release the alarm. This whole process is depicted in Figure A.11.

7.4.5 Tilt Detector: Contelog Instance-1, Status Detector

This program takes context from the controller unit as an input-context. As it receives the context, it is triggered to run evaluation and infer results. Those results are communicated back to the controller. The c-program for this is given below, which recommends the action to be taken on the alarm. However, the controller will not take this action until it verifies the status of the alarm from *Contelog* Instance-2.

Basically, Status Detector consists of four contexts $\{c_1, c_2, c_3, c_4\}$ all having the same context schema. Each context defines a certain setting and the recommended action in the case of this setting. The context consists of two dimensions "position" and "alarm". Position dimension can be "tilt", "notilt", "nuetral", which are the three settings of the sensor. Tilt means it is inclined, notilt means horizontal, and neutral means it is stable for a long time. The context-aware system will treat neutral and notilt with the same reaction. The alarm dimension can be "set" to recommend for setting the alarm off or "released" to turn the alarm off. For instance, if a sensor is in context "c1" (defined below), then it is on an inclined surface, and the alarm needs to be on. The context "input" is the input to the status detector *Contelog* program. It changes/re-entered by the controller, and every time it does, the program re-evaluatesthe context against the facts and rules.

The fact is $sensor(1)@input$, which links the sensor name (in case we have more than one sensor) to the input context related to it. In our case we have one input context and one sensor.

The program has one rule that infers the recommended actions "recommend(X,Y)@C" based on the predicates:

- sensor(S)@C: unifies with the fact sensor(1)@input.
- position(X)@C: unifies with the position of the input context.
- position(X)@W: unifies with the position of program contexts (c1-c4).
- alarm(Y)@W: recommends the action based on the contexts from (c1-c4).

```
# Contexts
c1={position:[tilt],alarm:[set]}
c2={position:[notilt],alarm:[set]}
c3={position:[notilt],alarm:[released]}
c4={position:[nuetral],alarm:[released]}
input={position:[tilt],alarm:[set]}
# Facts to link the sensor to the input context
sensor(1)@input.
# Rules
recommend(X,Y)@C :- sensor(S)@C,position(X)@C, position(X)@W,alarm(Y)@W.
```

7.4.6 Tilt Detector: Contelog Instance-2, Decision Maker

This program takes context from the controller unit as an input-context. As it receives the context, it is triggered to run evaluation and infer results. Those results are communicated back to the controller. The goal of this program is to direct the action to be taken after confirming the status of the alarm. That is , if it is "ON" and the action to take is "ON", this c-program will not take any action to avoid wasting resources. However, it takes action only if the action to be taken is different from the status of the alarm.

Basically, this Decision Maker program consists of three contexts all having the same context schema. Each context defines a certain setting and the case, and action to be taken. Status dimension can be "set" or "released", which indicates the status of alarm within this context. The "action" dimension can also be "set" or "released". Finally, the case dimension can be "tilt", "notilt", or "nuetral", which indicates the context of the sensor. The context "input" is the input to the decision maker *Contelog* program as an input from the status detector *Contelog* program through the control unit. It changes/re-entered by the controller, and every time it does, the program re-evaluates the context against the facts and rules.

The fact is $sensor(1)@input$, which links the sensor name (in case we have more than one sensor) to the input context related to it. In our case we have one input context, and one sensor.

The program has one rule that infers the action to take "take_action(X)@W" based on the predicates

- sensor(S)@C: unifies with the fact sensor(1)@input.
- recommended(X,Y)@C: unifies with the input context coming from the controller passed by "status detector" *Contelog* program.
- status(Y)@W: unifies with the position of program contexts (release1,release2,setoff).
- action(E)@W: unifies with the action dimension in the contexts (release1,release2,setoff).
- case(Y)@W: unifies with the case dimension in contexts (release1,release2,setoff).

```
# Contexts
release1={status:[set],action:[release],case:[nottilt]}
release2={status:[set],action:[release],case:[nuetral]}
setoff={status:[released],action:[set],case:[tilt]}
input={recommend:[tilt,released]}
# Facts to link the sensor to the input context
sensor(1)@input.
# Rules
take_action(X)@W :-
    sensor(S)@C,recommend(X,Y)@C,status(Y)@W,action(E)@W,case(X)@W.
```

The above system is fully implemented and is made available in the Book of Examples provided in Appendix B, along with all other case studies discussed in this chapter. The Book of Examples is made world available through the web (Alsaig, 2017) in particular to researchers and practitioners in database and context-aware systems. We welcome any comments on the examples provided. More comment more on the significance of "active *Contelog* for context-aware reasoning systems" in Section 9.2.

Chapter 8

Performance Evaluation

In this chapter, we evaluate and report the performance of *Contelog* . We briefly present the complexity. We then study and compare its scalability and efficiency. The data sets we used for performance evaluation are borrowed from two well-known case studies presented in (Liang et al., 2009) and (Brass & Stephan, 2017). We compare *Contelog* against several different implementations of Datalog including DLV, Iris, and XSB. The results of our numerous experiments show that the semi-naive method improves the performance over the naive method, then show semi-naive with the magic context rewriting significantly improves the query processing performance. These improvement results are comparable to those reported in the Datalog related literature.

8.1 Algorithmic Complexity

The complexity of evaluating Datalog programs has been studied thoroughly (Ceri et al., 1989; Grau et al., 2020; Greco & Molinaro, 2015; Liang et al., 2009; Tekle & Liu, 2011). The study considers three different complexity measures: data size in the number of input tuples, program size in the number of rules, and the combined data-program complexity. A summary of these results is shown in Table 8.1. *Contelog* complexity can be looked at from theoretical and practical perspectives. Since *Contelog* is a conservative extension of Datalog, and, as in Datalog, uses the Herbrand structures and models as the semantics basis, its complexity class remains the same as Datalog. In essence, the Herbrand universe of a *Contelog* program is made up of the union of the constants in the program and the constants and context names in the context lattice. The Herbrand base of a program is the collection of all ground atoms with and without context annotations. However, because in real world application the number of contexts in comparison to the number of facts is negligible,i.e. very small, the increase of the number of constants/ground predicates in the Herbrand universe, and Herbrand base, respectively, is by a constant factor. Therefore, *Contelog* has the same order of magnitude as Datalog,i.e. it terminates in polynomial time.

From the practical side, we can measure and compare their performance using measures such as execution time and program size. In the next section, we present the performance results of our experiments.

8.2 Scope of Comparison

In this section, we compare the semi-naive and magic context implementations of *Contelog* with the implementations of Datalog, shown in Table 8.2. It should be mentioned that we found not implementation of Datalog or its extensions for context based reasoning. We compare our *Contelog* implementation with the following systems/engines and the features listed:

(A) *Contelog* : semi-naive, indexing, and Magic Contexts.

(B) SDT: bottom-up with Subsumptive Demand Transformation as proposed in (Tekle & Liu, 2011).

(C) XSB (Sagonas, Swift, & Warren, 1999) with magic sets.

(D) OntoBroker (Decker, Erdmann, Fensel, & Studer, 1999): Ontology-based bottom up evaluation with magic sets.

(E) Iris (Bishop & Fischer, 2008): optimized implementation of Datalog with magic sets, tabling, and optimized indexing techniques.

(F) DLV (Alviano et al., 2010): bottom-up evaluation with optimized magic sets and premises reordering techniques.

Program Type	Data Complexity	Program Complexity	Combined Complexity
General	PTIME-complt	EXPTIME-complt	EXPTIME-complt
Linear	NLOGSPACE-complt	PSPACE-complt	PSPACE-complt

Table 8.1: Summary of the Complexity of Datalog programs

Systems	Semi-naive	indexing	tabling	MST/DT	Contexts	MCT
Contelog	✓	✓	×	×	✓	✓
SDT	✓	✓	✓	DT	×	×
XSB	✓	✓	×	MST	×	×
Ontobroker	✓	✓	✓	Optimized MST	×	×
Iris	✓	✓	✓	Optimized MST	×	×
DLV	✓	✓	✓	Optimized MST	×	×

Table 8.2: Features For Comparing Evaluation Approaches

For (C), (D), (E), and (F), we managed to get the corresponding original implementations. For (B), the original implementation (Tekle & Liu, 2011) was not available, however, following the description in their paper, we developed a running system and used it in our comparison with *Contelog* and other engines listed above.

8.3 Performance Evaluation of *Contelog*

Before we compare *Contelog* to other systems, we evaluate and report its performance with respect to implementations of different evaluation methods of naive, semi-naive, and MCT, we developed in this research. We use the "building locator" and “magic box" programs for this comparison. In the experiments, we used input data sets with the number of tuples varying from 500 to 10,000,000 tuples, as indicated in a figure legend. That is, for the building locator example, we consider 500 to 10,000,000 people coming from different directions to locate the library building, and for the magic box example, the range of boxes we consider in these experiments is from 500 to 10,000,000. Figure 8.1 shows the evaluation time in seconds for each input size. As expected, it can be seen that the performance of semi-naive evaluation method is far better than the naive method. The results also show that the performance is further improved when using the MCT rewriting method introduced in Chapter 6. It is important to note, however, that memory utilization has almost doubled when using semi-naive. This is due to the use of indexing and storage structures to keep track of newly derived tuples. The figure illustrates that *Contelog* with MCT shows

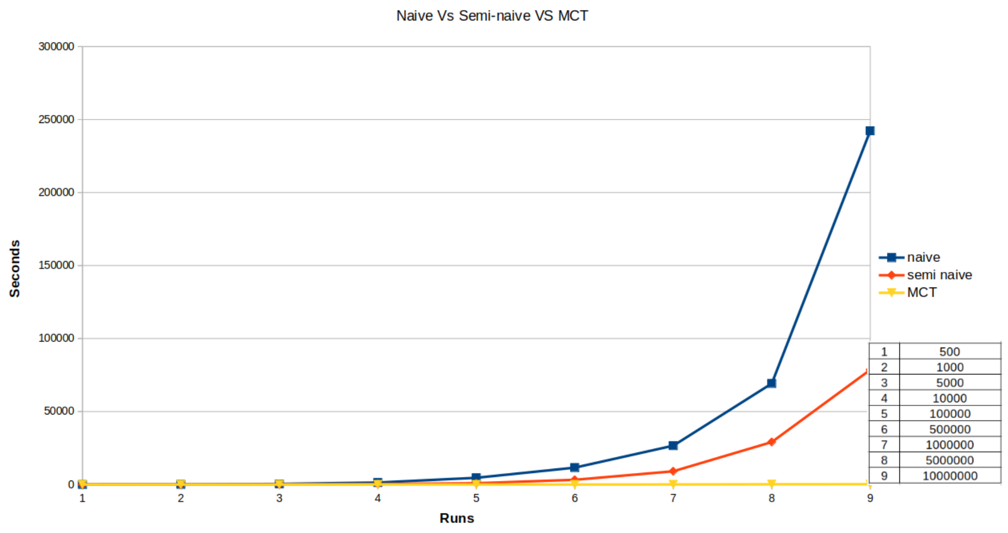


Figure 8.1: Performance of *Contelog* on large number of facts

a great advantage in performance over the semi-naive without MCT. Since *Contelog* with semi-naive and MCT is the most efficient when compared to other *Contelog* versions, we consider this implementation and features when comparing to other systems.

8.4 Experiments

We follow the approach from (Liang et al., 2009) to ensure that our comparisons are correct and fair. Also, the same tests were run on the same machine to build upon one reference point to ensure fairness of the results. The goal is to compare the performance of the MCT method against previous approaches in different settings, such as large sets, cyclic and acyclic data, large joins, and recursive rules. Through the experiments and results, we will illustrate performance superiority of the MCT rewriting method compared to previous ones.

The data sets of tuples used as inputs in our performance evaluation experiments are randomly generated and the same sets are used for evaluating all the methods mentioned. These data sets contain 50000, 100000, 250000, 500000, and 1000000 tuples. The parameter *timeout* is the time we set to terminate an evaluation process if it runs for too long. The parameter *memory-out* shows the system ran out of main memory and the current process cannot continue. We set the timeout limit to 3600 seconds and the memory limit to 8 GB.

For each test settings we perform 10 runs and report the average in the results. Also, for each experiments, the following tests were conducted:

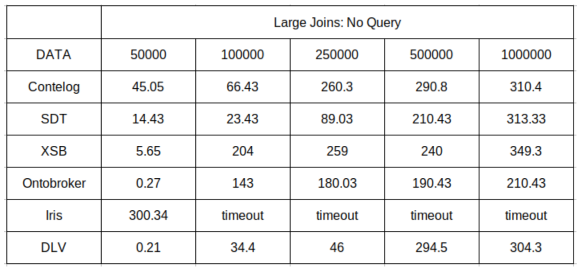
- No query: this means that no query was considered. Hence, rewriting technique is irrelevant, and the performance depends on semi-naive and the level of optimization on tabling, dynamic indexing, and premises re-ordering.
- Query with two arguments: The reason why we make it on two different categories is because the binding pattern highly affects the quality of the rewriting for the previous methods, but not for *Contelog* . Those tests showcase this.
- Recursive Data: We consider both cyclic and acyclic data in our experiments. This is to insure that the rewriting method is able to produce the answer tuples and efficiently retrieve relevant tuples from cyclic data.
- Context-based programs: We used three different binding patterns to carry out the tests in this category. The query in this category is in the form $Q(X,Y)@C$. The first binds the first argument, the second query binds the second, and the third binds the context. Because the other engines considered do not support contexts, we have defined context as regular facts and made some changes to the program to correctly use those contexts.

8.4.1 Large Joins Rules

In order to evaluate the performance of the various rewriting and evaluation methods on large joins, we use the following program that includes non-recursive rules, each of which uses two binary predicates. The base relations are c2, c3, c4, d1, and d2, and the derived relations are a1, b1, and b2. For the query, we considered the following binding combinations: Q(f,f), Q(f,b), Q(b,f), where f means free argument and b means bound.

Π_{large}

```
r1 : a1(X,Y) :- b1(X,Z), b2(Z,Y).
r2 : b1(X,Y) :- c1(X,Z), c2(Z,Y).
r3 : b2(X,Y) :- c3(X,Z), c4(Z,Y).
r4 : c1(X,Y) :- d1(X,Z), d2(Z,Y).
```



	Large Joins: No Query				
DATA	50000	100000	250000	500000	1000000
Contelog	45.05	66.43	260.3	290.8	310.4
SDT	14.43	23.43	89.03	210.43	313.33
XSB	5.65	204	259	240	349.3
Ontobroker	0.27	143	180.03	190.43	210.43
Iris	300.34	timeout	timeout	timeout	timeout
DLV	0.21	34.4	46	294.5	304.3

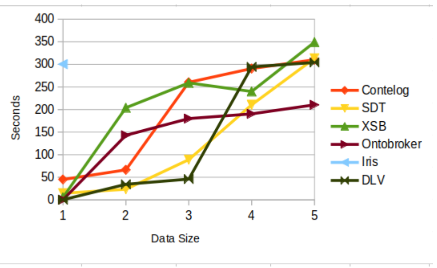
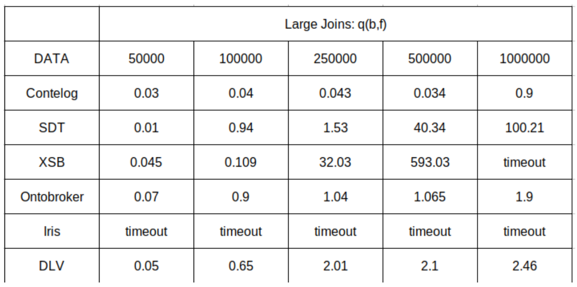


Figure 8.2: Large Joins, No Query Results



	Large Joins: q(b,f)				
DATA	50000	100000	250000	500000	1000000
Contelog	0.03	0.04	0.043	0.034	0.9
SDT	0.01	0.94	1.53	40.34	100.21
XSB	0.045	0.109	32.03	593.03	timeout
Ontobroker	0.07	0.9	1.04	1.065	1.9
Iris	timeout	timeout	timeout	timeout	timeout
DLV	0.05	0.65	2.01	2.1	2.46

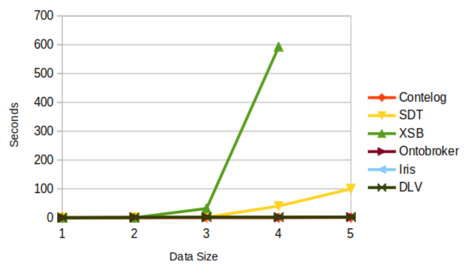
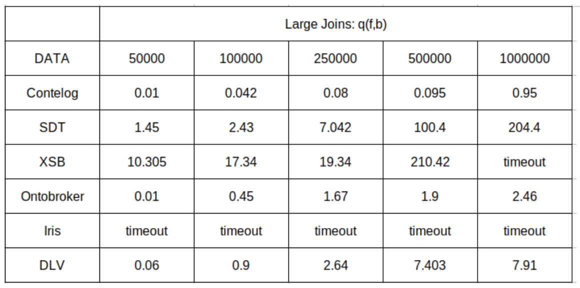


Figure 8.3: Large Joins With First Argument Bound

Results of Large Joins - No query. The summary of results of this set of experiments is shown in Figure 8.2. As can be seen, the results show that Ontobroker and DLV performed the best. This is because they are highly optimized to handle bottom-up evaluations. At the level of the program, they apply re-ordering of premises, and minimizing. Also, they have incorporated dynamic indexing, and apply tabling techniques in addition to heuristic algorithms to find common results and avoid repetitions. The performance of *Contelog* is not as good since it is using simple indexing techniques in addition to semi-naive approach. This can be addressed through more sophisticated indexing to be devised and employed for *Contelog* .

Results of Large Joins - First Argument Bound. The summary of results of this set of experiments is shown in Figure 8.3. As indicated by the results, due to the rewriting techniques of MCT, *Contelog* competes well and even beats Ontobroker and DLV, noting that they have better indexing and tabling techniques. The reason why *Contelog* performs much better in these experiments is because the MCT rewriting technique generates programs whose sizes are almost half of those produced by other systems, making *Contelog* perform faster scans through facts and rules to produce the results. Also, *Contelog* removes



	Large Joins: q(f,b)				
DATA	50000	100000	250000	500000	1000000
Contelog	0.01	0.042	0.08	0.095	0.95
SDT	1.45	2.43	7.042	100.4	204.4
XSB	10.305	17.34	19.34	210.42	timeout
Ontobroker	0.01	0.45	1.67	1.9	2.46
Iris	timeout	timeout	timeout	timeout	timeout
DLV	0.06	0.9	2.64	7.403	7.91

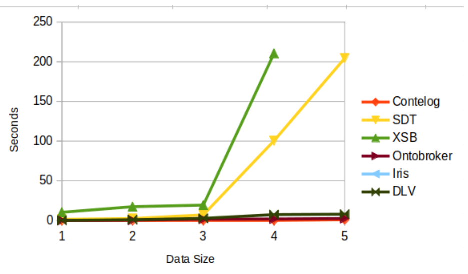


Figure 8.4: Large Joins with First Argument Bound

irrelevant rules and hence, many facts are not even considered during query processing for not being relevant.

Results of Large Joins - Second Argument Bound. The summary of results of this set of set of experiments is shown in Figure 8.4. These experiments show similar findings as in the previous one. However, a major advantage of MCT is that binding patterns do not change or affect the rewriting technique and the program size. Hence, *Contelog* show consistent performance while this is not true for all other engines. For instance, the results for Ontobroker in the last column (for 1,000,000 tuples) almost doubled when compared to the previous results only because of the binding change in the query.

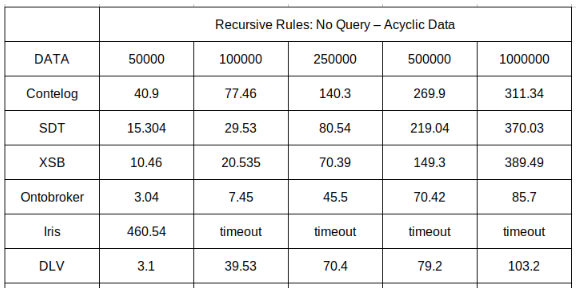
8.4.2 Recursive Rules

For the category of recursive tests, the programs are the "edge-path" and the "same generation cousins" programs provided below. The data selected for this example can be both acyclic and cyclic. For instance, the dataset p(1,2), p(2,3), p(1,3) is cyclic, and the reasoner will infer all possible combination of p(X,Y).

Listing:

```
# edge-path program $\Pi^1_{recursion}$
$r_1$ : p(X,Y) :- e(X,Y).
$r_2$ : p(X,Y) :- p(X,Z), e(Z,Y).

# same generation cousins program $\Pi^2_{recursion}$
$r_1$: sgc(X,Y) :- sib(X,Y).
$r_2$: sgc(X,Y) :- par(X,Z), sgc(Z,Z1), par(Y,Z1).
```



	Recursive Rules: No Query – Acyclic Data				
DATA	50000	100000	250000	500000	1000000
Contelog	40.9	77.46	140.3	269.9	311.34
SDT	15.304	29.53	80.54	219.04	370.03
XSB	10.46	20.535	70.39	149.3	389.49
Ontobroker	3.04	7.45	45.5	70.42	85.7
Iris	460.54	timeout	timeout	timeout	timeout
DLV	3.1	39.53	70.4	79.2	103.2

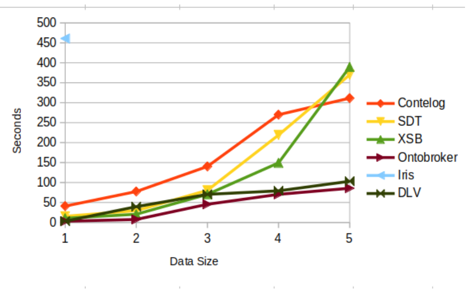
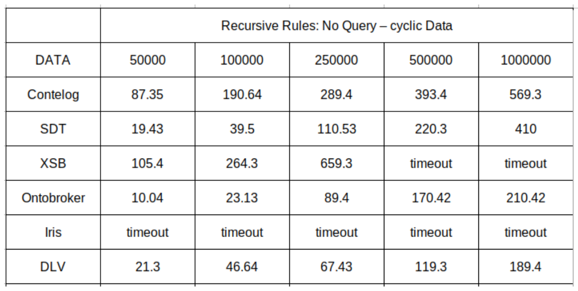


Figure 8.5: Recursive Rules, No Query Results - Acyclic



	Recursive Rules: No Query – cyclic Data				
DATA	50000	100000	250000	500000	1000000
Contelog	87.35	190.64	289.4	393.4	569.3
SDT	19.43	39.5	110.53	220.3	410
XSB	105.4	264.3	659.3	timeout	timeout
Ontobroker	10.04	23.13	89.4	170.42	210.42
Iris	timeout	timeout	timeout	timeout	timeout
DLV	21.3	46.64	67.43	119.3	189.4

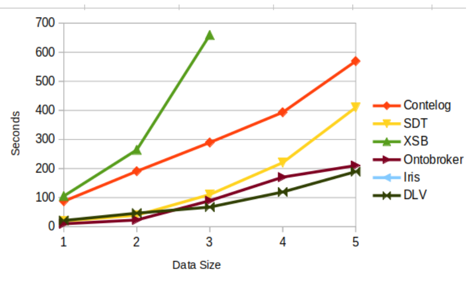
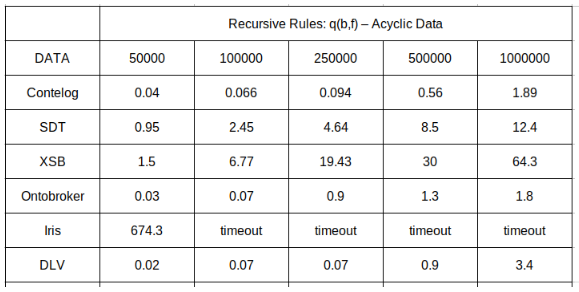


Figure 8.6: Recursive Rules No Query Results - Cyclic

Results of Recursive Rules - No Query - Acyclic. The summary of results of this set of experiments is shown in Figure 8.5. This test and second one are the heaviest tests in our experiments. This is because recursive rules take up large memory and time to compute. Similar to large joins, Ontobroker and DLV show best performance due to their indexing and tabling techniques. Iris and XSB show least performance in this set of tests.

Results of Recursive Rules - No Query - Cyclic. The summary of results of this set of experiments is shown in Figure 8.6. Similar to the previous test, Ontobroker and DLV show best performance due to their indexing and tabling techniques. Although *Contelog* is not highly optimized when there is no input query, it has consistently passed all the tests with a steady increase.

Results of Recursive Rules - First Argument Bound - Acyclic. The summary of results of this set of experiments is shown in Figure 8.7. *Contelog* ranked top in this test. As explained earlier, it is an advantage of the MCT rewriting technique that it does not consider binding patterns in rules as it deals with individual bound argument of the query and restricts each rule to the magic context.



	Recursive Rules: q(b,f) – Acyclic Data				
DATA	50000	100000	250000	500000	1000000
Contelog	0.04	0.066	0.094	0.56	1.89
SDT	0.95	2.45	4.64	8.5	12.4
XSB	1.5	6.77	19.43	30	64.3
Ontobroker	0.03	0.07	0.9	1.3	1.8
Iris	674.3	timeout	timeout	timeout	timeout
DLV	0.02	0.07	0.07	0.9	3.4

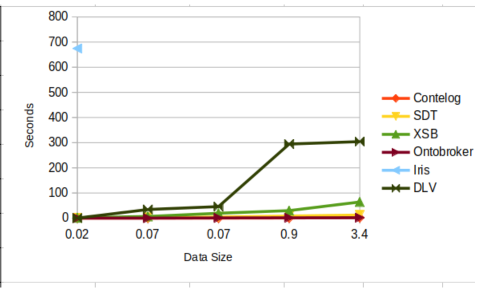
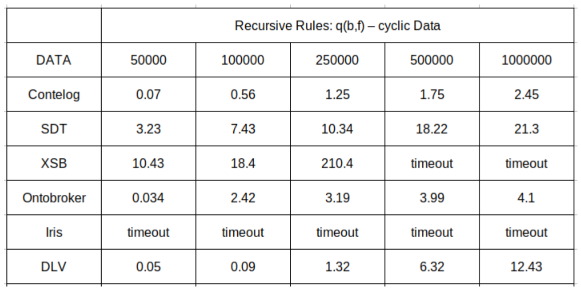


Figure 8.7: Recursive Rules First Argument Bound - Acyclic



	Recursive Rules: q(b,f) – cyclic Data				
DATA	50000	100000	250000	500000	1000000
Contelog	0.07	0.56	1.25	1.75	2.45
SDT	3.23	7.43	10.34	18.22	21.3
XSB	10.43	18.4	210.4	timeout	timeout
Ontobroker	0.034	2.42	3.19	3.99	4.1
Iris	timeout	timeout	timeout	timeout	timeout
DLV	0.05	0.09	1.32	6.32	12.43

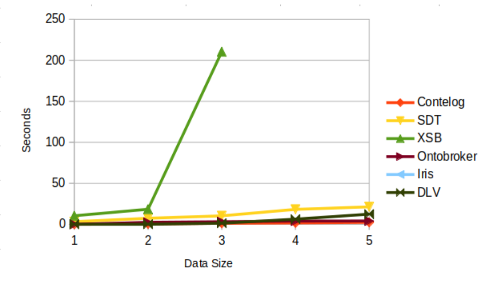
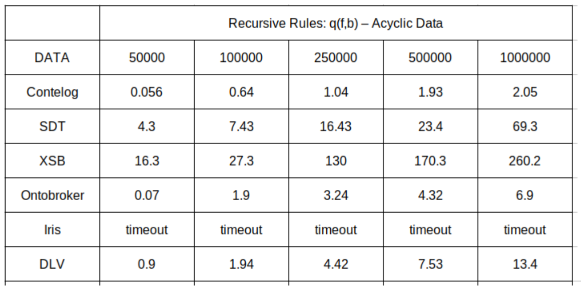


Figure 8.8: Recursive Rules First Argument Bound - Cyclic



	Recursive Rules: q(f,b) – Acyclic Data				
DATA	50000	100000	250000	500000	1000000
Contelog	0.056	0.64	1.04	1.93	2.05
SDT	4.3	7.43	16.43	23.4	69.3
XSB	16.3	27.3	130	170.3	260.2
Ontobroker	0.07	1.9	3.24	4.32	6.9
Iris	timeout	timeout	timeout	timeout	timeout
DLV	0.9	1.94	4.42	7.53	13.4

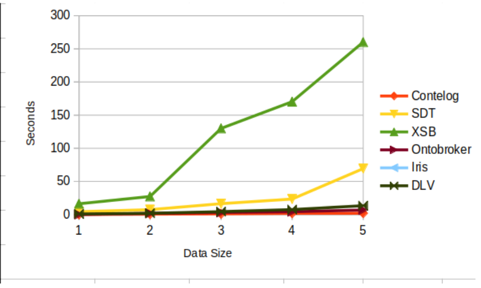
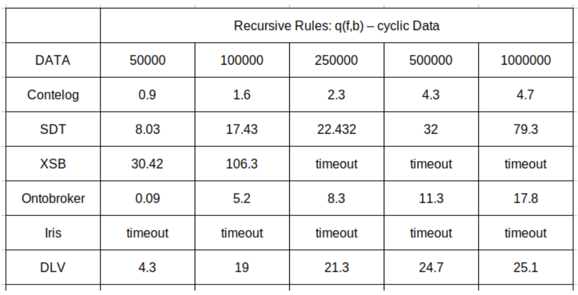


Figure 8.9: Recursive Rules Second Argument Bound - Acyclic

Results of Recursive Rules - First Argument Bound - Cyclic. The summary of results of this set of experiments is shown in Figure 8.8. Here too, *Contelog* ranks top. XSB could not pass the test. All engines including *Contelog* show reduced performance by a constant factor when compared to the Acyclic tests.

Results of Recursive Rules - Second Argument Bound - Acyclic. The summary of results of this set of experiments is shown in Figure 8.9. The performance of *Contelog* is similar to the case with first argument bound. However, the other engines performance have been severely impacted.



	Recursive Rules: q(f,b) – cyclic Data				
DATA	50000	100000	250000	500000	1000000
Contelog	0.9	1.6	2.3	4.3	4.7
SDT	8.03	17.43	22.432	32	79.3
XSB	30.42	106.3	timeout	timeout	timeout
Ontobroker	0.09	5.2	8.3	11.3	17.8
Iris	timeout	timeout	timeout	timeout	timeout
DLV	4.3	19	21.3	24.7	25.1

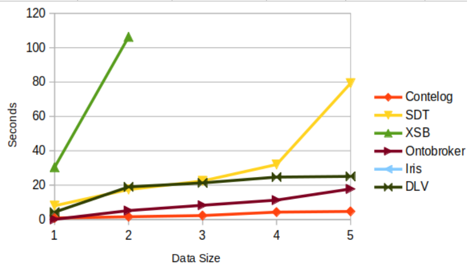


Figure 8.10: Recursive Rules Second Argument Bound - Cyclic

Results of Recursive Rules - Second Argument Bound - Cyclic. The summary of results of this set of experiments is shown in Figure 8.10. *Contelog* shows consistent results in comparison to the cyclic test with the first argument bound. However, all other engines rewriting have been highly impacted and resulted in reduced performance.

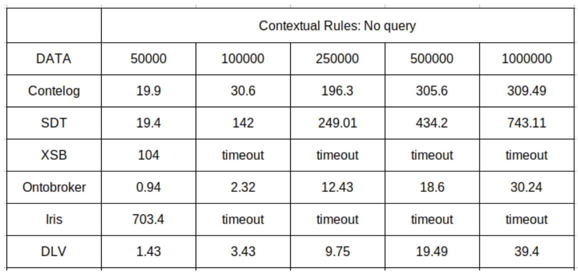
8.4.3 Contextual Rules

For this category of tests, we used the animal-inheritance program ($\Pi_{context}$), shown below, which includes recursion and dependency on contexts. In this program, a combination of the derived predicates $a(X,Y)@C$ and $f(X,Y)@C$ are used with the query bindings $Q(f,b)@f$, $Q(b,f)@f$, and $Q(f,f)@b$.

$\Pi_{context}$

$r_1 : a(X,Y)@C$:- $a(X,Y)$, $t(Y)@C$.
$r_2 : a(X,Y)@C$:- $a(X,Z)@C$, $a(Z,Y)$.
$r_3 : f(X,Y)@C$:- $a(X,Z)@C$, $fea(Y)@C$.

We remark that except for *Contelog* , none of the other evaluation methods employed or implemented in our experiments supported context representation and reasoning. One of our objectives in the experiments was to study the performance of *Contelog* with large number of tuples and/or large number of contexts. So, we generated and used two sets of data, one in which the number of tuples was fixed to 1000 but with dynamic set of contexts ranging from 50,000 to 10^6, and the other in which the number of context was fixed to 1000 but with dynamic set of tuples ranging from 50,000 to 10^6.



	Contextual Rules: No query				
DATA	50000	100000	250000	500000	1000000
Contelog	19.9	30.6	196.3	305.6	309.49
SDT	19.4	142	249.01	434.2	743.11
XSB	104	timeout	timeout	timeout	timeout
Ontobroker	0.94	2.32	12.43	18.6	30.24
Iris	703.4	timeout	timeout	timeout	timeout
DLV	1.43	3.43	9.75	19.49	39.4

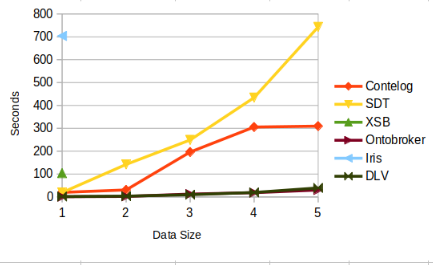
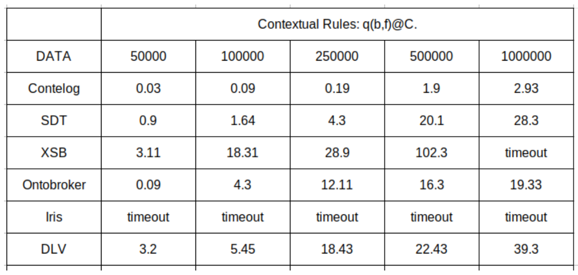


Figure 8.11: Contextual Rules - No Query



	Contextual Rules: q(b,f)@C.				
DATA	50000	100000	250000	500000	1000000
Contelog	0.03	0.09	0.19	1.9	2.93
SDT	0.9	1.64	4.3	20.1	28.3
XSB	3.11	18.31	28.9	102.3	timeout
Ontobroker	0.09	4.3	12.11	16.3	19.33
Iris	timeout	timeout	timeout	timeout	timeout
DLV	3.2	5.45	18.43	22.43	39.3

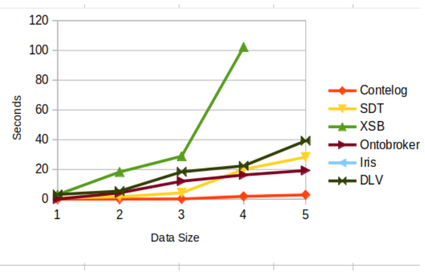
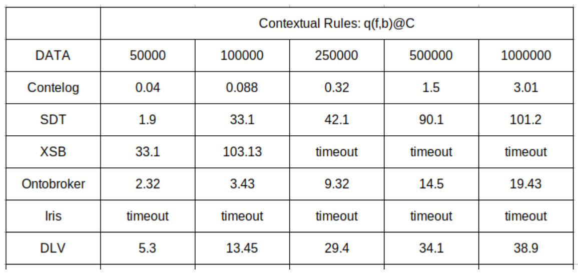


Figure 8.12: Contextual Rules - First Argument Bound



	Contextual Rules: q(f,b)@C				
DATA	50000	100000	250000	500000	1000000
Contelog	0.04	0.088	0.32	1.5	3.01
SDT	1.9	33.1	42.1	90.1	101.2
XSB	33.1	103.13	timeout	timeout	timeout
Ontobroker	2.32	3.43	9.32	14.5	19.43
Iris	timeout	timeout	timeout	timeout	timeout
DLV	5.3	13.45	29.4	34.1	38.9

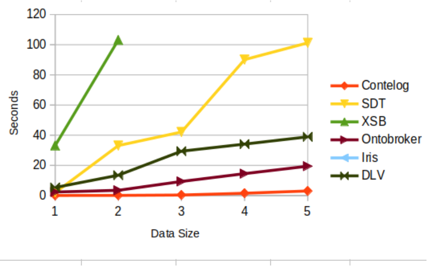
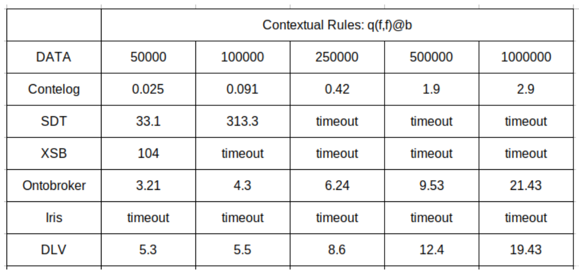


Figure 8.13: Contextual Rules - Second Argument Bound

Contextual Rules - No Query. The summary of results of this set of experiments is shown in Figure 8.11. Although *Contelog* is the only engine that have contexts, it did not rank the top since there was no query. We have dealt with contexts in other programs as arguments and facts. There are other ways that we can implement this by using function symbols, but we noticed adverse impacts on the performance on other engines when using function symbols, so we decided not to use function symbols.

Contextual Rules - First Argument Bound. The summary of results of this set of experiments is shown in Figure 8.12. In this test as all previous tests with queries, *Contelog* ranks at the top.



	Contextual Rules: q(f,f)@b				
DATA	50000	100000	250000	500000	1000000
Contelog	0.025	0.091	0.42	1.9	2.9
SDT	33.1	313.3	timeout	timeout	timeout
XSB	104	timeout	timeout	timeout	timeout
Ontobroker	3.21	4.3	6.24	9.53	21.43
Iris	timeout	timeout	timeout	timeout	timeout
DLV	5.3	5.5	8.6	12.4	19.43

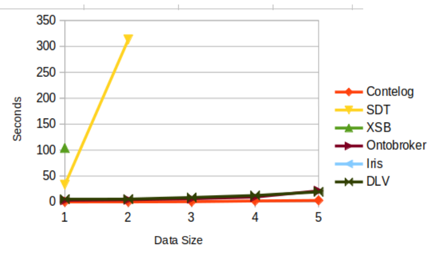


Figure 8.14: Contextual Rules - Third (context) Argument Bound

Contextual Rules - Second Argument Bound. The summary of results of this set of experiments is shown in Figure 8.13. *Contelog* in this test shows consistent results, unlike other engines.

Contextual Rules - Third (Context) Argument Bound. The summary of results of this testset is shown in Figure 8.14. We referred to the context as a third argument, because this is how we used it in engines that don't support context. *Contelog* ranks at the top, while performance of DLV and Ontobroker are adversely impacted by the addition of the third argument.

8.5 Summary of Experiments

We summarise our findings and observations as follows. First, in evaluation of all programs that contained no-query, which means generating all the tuples, DLV and Ontobroker ranked as the top. This is because both systems use the optimization techniques such as indexing, tabling, and re-ordering, to enhance the bottom-up evaluation. *Contelog* did not perform at its best as it is not equipped with such techniques. Second, for all query-dependant and rewritten programs using magic-context, *Contelog* ranked the top. DLV and Ontobroker ranked second and other systems fell far behind. Third, number of arguments and the binding patterns show high impact on the quality and performance of the rewritten program for all reviewed systems, but not for *Contelog* . In fact, *Contelog* shows steady performance for all binding patterns. Finally, *Contelog* ranked as the top for all query-dependant programs and ranked low for programs with no-query. This indicates opportunity for *Contelog* for considerable improvement by optimizing the semi-naive techniques to compete with DLV and Ontobroker. However, even with current implementation of *Contelog* , it shows high potential and in fact best performance among the other systems in query-dependant contextual and non-contextual programs.

Chapter 9

Conclusion and Future Work

In this chapter, we summarise the significant contributions of this thesis. We highlight what *Contelog* can achieve and how it can be further improved. We comment on how *Contelog* provides a theory for representing and reasoning with contexts in an efficient and practical manner. Finally, we present a list of topics for future work, in particular those that can be pursued to further improve *Contelog* towards using it as a back-end reasoner to support real-world, context-aware applications and as a context-aware knowledge base system.

9.1 Summary of Contributions

In this thesis, we introduced *Contelog* , a full-fledged logic framework and established its model theory and fixpoint semantics. The framework is fully implemented as a running prototype system, and its performance is evaluated using a different programs with varying number of tuples.

Contelog is a conservative extension of Datalog with context, in which contexts are represented and reasoned with as first class citizens, and enjoys powerful query processing and optimization techniques. A key advantage feature of *Contelog* over other proposals for context representation and reasoning is the "loosely-coupled" connection between the context world and problem solving world (also called the reasoning world). Through this feature, we achieve additional advantages in context-based reasoning at both theoretical and practical levels. At the theory level, *Contelog* provides separation between the problems of context representation and contextual reasoning. We provide a formal representation of context by introducing a theory for context lattice and an associated calculus, using which it performs desired reasoning over the context world in divers applications.

The entire contextual information can be modeled using context entity alone. However, as in real life, context by itself does not mean much unless it is linked to events in a world where it is needed and used. The proposed formalism developed for *Contelog* extends the syntax and semantics of Datalog. Therefore, *Contelog* with a high degree of freedom, can reason within a context, outside a context, or use contextual information in problem solving world. Therefore, *Contelog* achieves the transcendence property of context described by McCarthy. At practical level, the loose coupling between context and reasoning worlds provides opportunities for parallel reasoning, where multiple applications (reasoners) can use the same context(s) world without affecting each other. This independence is an advantage over ontology-based reasoners. Also, any optimization technique, theoretically or practically, will not affect the context world in the reasoning process. This design allows for future improvements, yet keeps the conceptual structure of the framework intact. In summary, *Contelog* theoretically and practically implements essential features of context-aware and context-based reasoners as suggested in major related literature, e.g., (Akman & Surav, 1996; Giunchiglia, 1993; Guha, 1991; McCarthy, 1963). Below, we highlight what *Contelog* has achieved.

- *Contelog* implements the function is-true $ist(C, P)$ introduced in (McCarthy, 1993) and the viewpoint function $in(‘A', vp)$ introduced in (Attardi & Simi, 1995) as an atomic predicate $p@c$.

- *Contelog* achieves simplicity and declarative semantics by conservatively extending Datalog.

- Being a conservative extension of Datalog, *Contelog* can also process standard Datalog programs efficiently without context notation.

- The theoretical concepts of entering and exiting contexts introduced in (Guha, 1991) has been practically implemented in *Contelog* . The concepts “enter", “exit",“roll-up",“roll-down",“push", and “pull", that appeared in different works in the literature are all realized in *Contelog* through the rules in problem solving context. A *Contelog* rule can infer something in a context, thus achieving the effects of “enter"/“push"/“roll-down". Also, it can infer something in problem solving context using contextual information, achieving the effect of “exit"/“pop"/“roll-up".

- *Contelog* actually achieves the “separation of concerns" and the “loose coupling" features introduced in (Akman & Surav, 1996; McCarthy, 1963) by separating the program components from the contexts module.
- It allows coexistence of possible “conflicting knowledge" introduced in (Akman & Surav, 1996; Attardi & Simi, 1995) by being able to reason in different contexts.
- *Contelog* assumes that contexts are “complete" for the purpose of reasoning, although they are not globally complete. The context theory assumes existence of a complete lattice of contexts for the problem solving world. It contains all possible contexts that can be constructed using the available contextual information. The underlying lattice provides the sense of “necessary completeness" and “closed world assumption" considered in many applications. Nevertheless, it does not violate the fact that context is a rich concept that may not be globally complete.
- It fulfills the evaluation aspects proposed in (Benerecetti et al., 2000).
- *Contelog* is formal, expressive, generic, yet flexible and simple framework. We illustrated these features in the magic box example and its improved version.
- *Contelog* is equipped with a novel and efficient rewriting method, called the Magic Context Transformation (MCT), that generates programs and can achieve the same efficiency provided by Magic Sets for standard Datalog programs. Also, because the size of the rewriting generated by MCT is small, it can process any query efficiently using contexts without requiring rewriting.
- Due to the efficiency of the MCT rewriting technique, *Contelog* can be used as an active reasoner as shown in Chapter 7. This makes *Contelog* , to the best of our knowledge, the first rule-based logic framework, with sound and complete proof procedures for context-aware reasoning.

- Using the separation of concerns between contexts and application logic, multiple programs with different facts can use the same contexts, or multiple contexts can be used by the same program. This makes *Contelog* suitable for providing opportunities for concurrent or parallel context-aware computations.

9.2 Future Work

Contelog is in its infancy and can be further improved along the following directions. The current implementation lacks indexing and tabling mechanisms (also called memoing in XSB) that can help speed up query processing even further. Our current implementation of *Contelog* uses a simple tokenizing method and does not benefit from sophisticated run-time optimization methods, such as indexing or multi-threading, for better memory and CPU utilization. We have not developed top-down query processing technique for *Contelog* . One reason is the proposed MCT rewriting together with the semi-naive evaluation methods achieved the efficiency and goal-oriented nature of top-down query processing. Despite this developing a top-down proof procedure may prove to be useful. One way to do this is using a meta-programming approach and build an interpreter in XSB for *Contelog* programs and associated contexts. This requires extension of unification to context variable or names in the rules and facts. We expect the performance of a desired top-down solution to be comparable to the MCT with semi-naive evaluation. This expectation remains to be assessed. Below, we list some of the venues that can be pursued towards real rule-based, context-aware, knowledge-base system.

(1) *Performance:*

- Following the ideas presented in (Ferreira & Rocha, 2005), investigate parallel computing in the implementation of the *Contelog* engine to further boost-up its performance.
- Implement optimized tabling and indexing techniques to enhance the performance of fetching data (facts/contexts,) as done for engines such as XSB, YAP, DLV, and Push, as noted in (Brass & Stephan, 2017).

(2) *Expressiveness:*

- Enhance the expressiveness of *Contelog* by allowing function symbols in contexts.
- Allow more data types, such as 'string', 'lists', and 'dictionaries' to describe data in contexts.
- Extend *Contelog* with negation to allow negated facts that can induce stratified programs and result in the development of stratified evaluation.
- Extend with uncertainty. Many real world applications require to represent and deal with uncertain information. It would be interesting to study extensions of *Contelog* to support applications with different notions and theories of uncertainty, such as probability, possibility, and fuzzy. Such applications might consider including a contextual probabilistic predicate $ispt(C, P, Pr)$. Such an extended *Contelog* might be expressive in describing expert systems, e.g., MYCIN (Riguzzi, 2013)
- Currently *Contelog* works under the closed world assumption. It would be interesting to extend it for open-world assumption.

(3) *Reasoning:*

- Implement top-down query processing techniques for *Contelog* . This would allow tracing, drawing the proof-tree/derivation tree corresponding a given query. While the process is a one-step-at-a-time fashion, it is suitable top provide explanation and details of tuple derivations.

(4) *Integration:*

- Provide integration through APIs or endpoints for applications and systems to connect to *Contelog* and benefit from it.
- Integrate *Contelog* with Big Data management and reasoning. Big Data sources will provide one or more global data sets, each containing all possible related contexts. A public application can use the *Contelog* reasoner to choose relevant data sets to its purposes.
- Investigate the design and implementation of a software tool to convert ontology or XML files to our context representation, and convert the source code to *Contelog* for reasoning.

(5) *Applications:*

- Investigate how to couple *Contelog* with current context-aware AI applications such as autonomous vehicles, ubiquitous computing, and Internet of Things (IoT) systems used to build SMART cities.

Appendix A

Book of Examples

A.1 The Direction Example

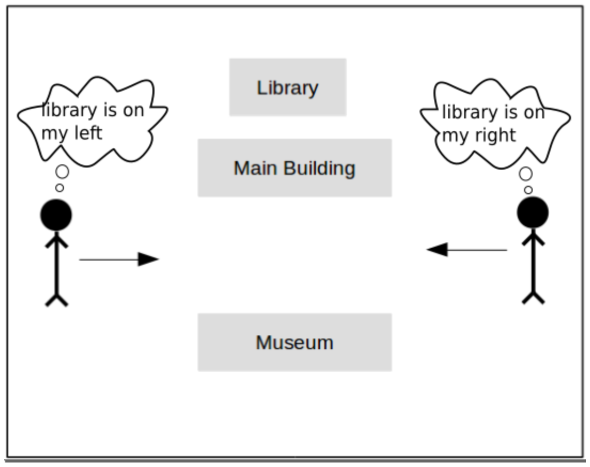


Figure A.1: Illustration of the building locator example

A.1.1 Goal

the goal of this program is to show the ability of contelog to perform context based reasoning by giving the person the correct direction based on his/her context.

A.1.2 Overview

This program direct the person to where the library is based on the side he/she coming from. For simplicity, people can only come from two sides, namely east/west.

A.1.3 Program Details

the program consist of two contexts and two regular predicates. The contexts are:

(1) a context that defines a person coming from the east, and should be directed to the right to reach the library.

(2) a context that defines a person coming from the west, and should be directed to the left to reach the library.

Thus, these two contexts contain two dimensions “from” and “to”, each of which has one attribute that defines the direction. Specifically, from dimension has the attribute “Ordinal Direction” with the domain of values [east,west], and to dimension has the attribute “Side” with the domain of values [right,left].

The program has two regular binary predicates: per(X,Y) where X is the name of the person, and Y is the ordinal direction he/she is coming from, and lib(X,Z) where X is the name of the person and Z is the direction of the library based on context.

A.1.4 Code

A.1.4.1 Context

```
cw= from:[west],to:[left]}
ce= from:[east],to:[right]}
```

A.1.4.2 Program

```
#FACTS:
per(1,east).
per(2,west).
per(3,north).
#RULES:
per(X,Y)@C:-per(X,Y),from(Y)@C.
lib(X,Y)@C:-per(X,Z)@C,to(Y)@C.

#--- RESULTS ---#
```

```
per(1,east).
per(2,west).
per(3,north).
**per(1,east)@ce. #the starred facts are the inferred ones.
**per(2,west)@cw.
**lib(1,right)@ce.
**lib(2,left)@cw.
```

A.2 Animal Classifier

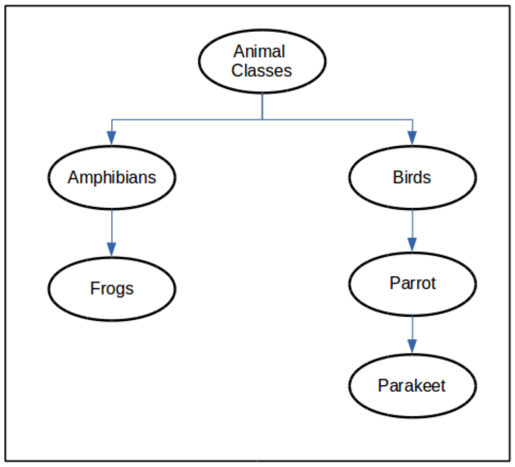


Figure A.2: illustration of inheritance example

A.2.1 Goal

the goal of this program is to show that contelog inherit the recursive reasoning from Datalog and it can reason using recursive rules.

A.2.2 Overview

The program classifies the animals and their descendants and give their features based on their context.

A.2.3 Program Details

The program contains two regular predicates and two contexts. The context defines the animal type and its features through the dimensions “type” and “f” respectively. The “type” dimension has an attribute "name of type" which has the domain of values [bird,amphibians]. The “f” dimension has an attribute "feature of animal" which has the domain of values [canfly,canswim].

The program contains two regular binary predicates: animal(X,Y) and feature(X,Z). “X” is the name of the animal, “Y” is the type of animal, “Z” is the feature of the animal.

A.2.4 Code

```
# CONTEXT
cb= {type:[bird],f:[canfly]}
ca= {type:[amphibian],f:[canswim]}


# FACTS:
animal(parrot,bird).
animal(frog,amphibian).
animal(parakeet,parrot).
animal(tods,frog).
# RULES:
animal(X,Y)@C:-animal(X,Y),type(Y)@C.
animal(X,Y)@C:-animal(X,Z), animal(Z,Y)@C.
feature(X,Y)@C:-animal(X,Z)@C,f(Y)@C.

# --- RESULTS --- #
animal(parrot,bird).
animal(frog,amphibian).
animal(parakeet,parrot).
animal(tods,frog).
**animal(parrot,bird)@cb.
**animal(frog,amphibian)@ca.
**animal(parakeet,bird)@cb.
```

```
**animal(tods,amphibian)@ca.
**feature(parrot,canfly)@cb.
**feature(frog,canswim)@ca.
**feature(parakeet,canfly)@cb.
**feature(tods,canswim)@ca.
```

A.3 Animal Classifier (enriched)

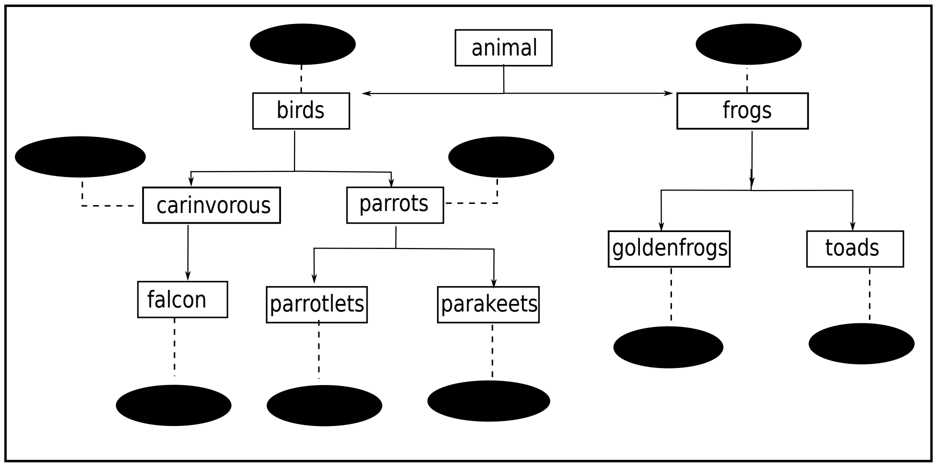


Figure A.3: illustration of inheritance example

A.3.1 Goal

the goal of this program is to show that contelog inherit the recursive reasoning from Datalog and it can reason using recursive rules in a more sophisticated scenario.

A.3.2 Overview

The program classifies the animals and their descendants and give their features based on their context. It also assigns features to childs based on inheritance. However, there are some specific features to childs that parents don't share.

A.3.3 Code

```
# CONTEXT
cbird= {type:[bird],f:[canfly],level:[a]}
camph= {type:[amphibian],f:[canswim],level:[a]}
cparrot={type:[bird],f:[cantalk],name:[parrot]}
cparakeet={type:[parrot],f:[small],name:[parakeet]}
ctod={type:[frog],f:[big],name:[toad]}
cyfrogs={type:[frog],f:[poisonous],name:[yellowfrog]}
cplet={type:[parrot],f:[bigbeaks],name:[parrotlet]}
cfalcons={type:[bird],f:[carnivorous],name:[falcon]}

# FACTS:
animal(parrot,bird).
animal(parakeet,parrot).
animal(parrotlet,parrot).
animal(falcon,bird).
animal(frog,amphibian).
animal(toad,frog).
animal(yellowfrog,frog).
# RULES:
animal(X,Y)@C:-animal(X,Y),type(Y)@C,level(a)@C.
animal(X,Y)@C:-animal(X,Y),type(Y)@C,name(X)@C.
animal(X,Y)@C:-animal(X,Z), animal(Z,Y)@C.
feature(X,Y)@C:-animal(X,Z)@C,f(Y)@C.

# --- RESULTS ---#
animal(parrot,bird).
animal(frog,amphibian).
animal(parakeet,parrot).
animal(tods,frog).
**animal(parrot,bird)@cb.
**animal(frog,amphibian)@ca.
**animal(parakeet,bird)@cb.
**animal(tods,amphibian)@ca.
**feature(parrot,canfly)@cb.
```

```
**feature(frog,canswim)@ca.
**feature(parakeet,canfly)@cb.
**feature(tods,canswim)@ca.
```

A.4 User Access Controller

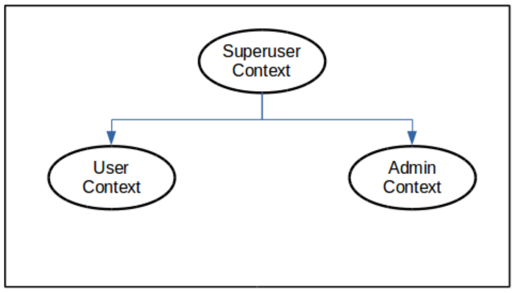


Figure A.4: illustration of abstract context

A.4.1 Goal

show the ability of contelog to perform operations (join/meet) on the contexts in the program.

A.4.2 Overview

The program assignes privileges to users based on their context and the roles/privileges defined in them. If a user has an admin and viewer roles he can have both of their privileges and can be identified as super user.

A.4.3 Program Details

Two contexts are included in this program. The viewer context 'cv' and the admin context 'ca'. Both contexts has the following schema: role:[name of role], priv:[name of priv]. The attribute name of role has the domain of values [viewer,admin] while the priv has the domain [canedit, canview].

The program contains two regular binary predicates: user(X,Y) and priv(X,Z), where:

- X: name of user

- Y: role of user

- Z: priv of user

```
# CONTEXT
cv= {role:[viewer],priv:[canview]}
ca= {role:[admin],priv:[canedit]}

# FACTS:
user(john,admin).
user(mike,viewer).
user(john,viewer).

# RULES:
# The user X of role Y is at context C if the role(Y) is in C
user(X,Y)@C:-user(X,Y),role(Y)@C.
# The user X of role Y is at the join of contexts C and W if the same user has
    two different roles at two different contexts
user(X,Y)@C+W:-user(X,Y)@C,user(X,Z)@W,Y!=Z.
# The privilege Y for is assigned for the user X based on the context defined
    from rule 1
priv(X,Y)@C:-user(X,Z)@C,priv(Y)@C.
# The privilege Y from the join of contexts W and C is assigned to user X
priv(X,Y)@C+W:-priv(X,Z)@C,priv(X,Y)@W,Z!=Y.


# --- RESULTS --- #
user(john,admin).
user(mike,viewer).
user(john,viewer).
**user(john,admin)@ca.
**user(mike,viewer)@cv.
**user(john,viewer)@cv.
**user(john,admin)@ca+cv.
**user(john,viewer)@cv+ca.
**priv(john,canedit)@ca.
```

```
**priv(mike,canview)@cv.
**priv(john,canview)@cv.
**priv(john,canview)@ca+cv.
**priv(john,canedit)@cv+ca.
```

A.5 Diagnosis Program

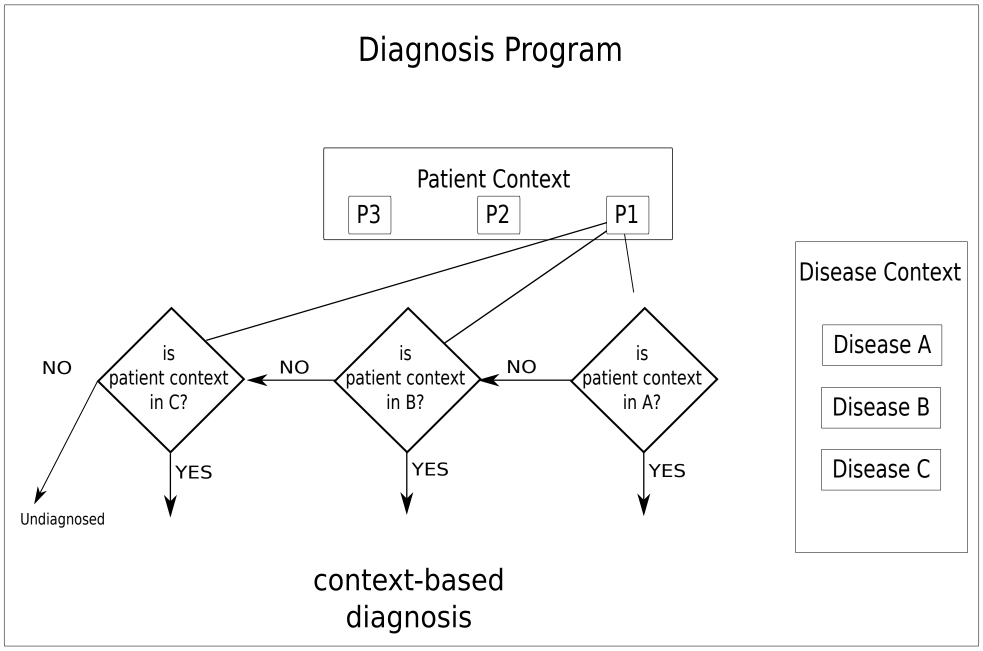


Figure A.5: Illustration of the diagnosis program

A.5.1 Goal

To show the exprissive power of contelog using the inequalities (built in predicate) and containment relation checker in the rules body.

A.5.2 Overview

Using containment/inclusion concept, the patient can be diagnosed if all symptomps and vital (patients' context) signs are included in the disease context.

A.5.3 Program Details

The context of this program is richer than previous ones. There are two different categories of context: patients' context, and disease context. Patient context has the following schema: $fever : [degree], bs : [degree], bp : [degree], symp : [nameofsymptom]$. It can also be any subschema of this schema. Example: $fever : [degree], symp : [nameofsymptom]$ is a patient context.

The disease context has the following schema:

$$fever : [degree], bs : [degree], bp : [degree], symp : [nameofsymptom].$$

the domain of 'degree' attribute is [low,normal,high]. The domain of 'name of symptom' attribute is [any possible symptoms].

The program consist of two regular unary predicates: $patient(X)$ and $diagnosis(Y)$. X is the name of patient and Y is the name of the diagnosis (disease).

A.5.4 Code

```
# CONTEXT
p1={fever:[high],symp:[headache]}
p2={bs:[high],symp:[dehydration]}
p3={bp:[high],symp:[chestpain]}
meningitis={bs:[normal],bp:[normal],fever:[high],symp:[headache]}
diebeties={bs:[high],bp:[normal],fever:[normal],symp:[dehydration]}
heart=\{bs:[normal],bp:[high],fever:[normal],symp:[chestpain]}

# FACTS:
# patients are assigned to a specific context already.
patient(john)@p1.
patient(rod)@p2.
patient(derek)@p3.

# RULES:
# if patient context is included in the disease context he/she is assigned a
    potential diagnosis.
```

```
diagnosis(X)@W:-patient(X)@C,symp(Y)@W,C<W,C!=W.

# --- RESULTS --- #
patient(john)@p1.
patient(rod)@p2.
patient(derek)@p3.
**diagnosis(john)@meningitis.
**diagnosis(rod)@diebeties.
**diagnosis(derek)@heart.
```

A.6 Database Access Controller

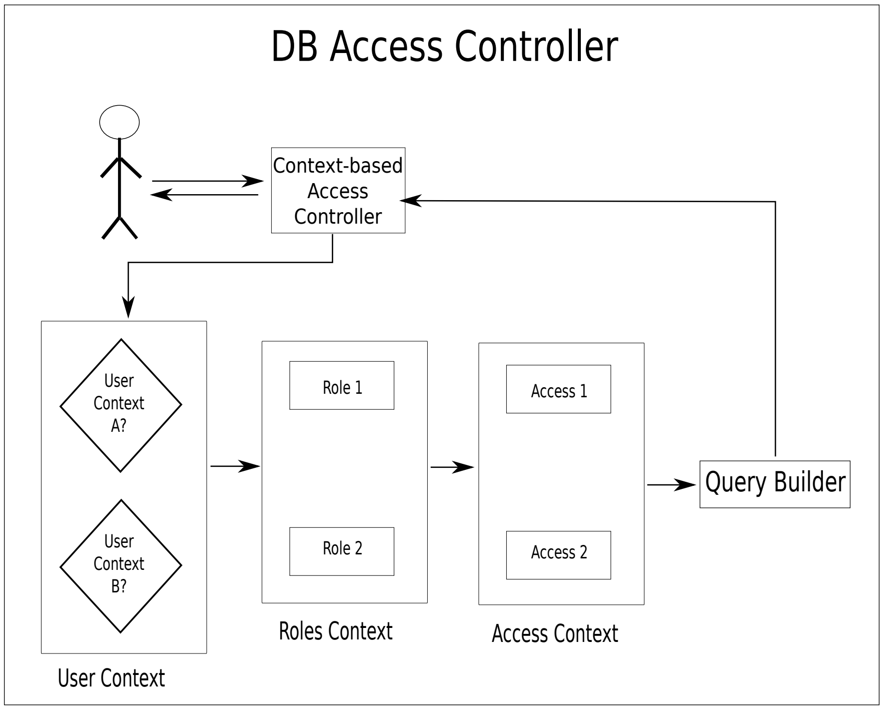


Figure A.6: Illustration of the db access controller program

A.6.1 Goal

manage database access based on the context of the user in a more complex settings.

A.6.2 Overview

The program controls the access of the user considering different contexts. It is different if a user location is within the premise or if it is a night or morning. Different context roles are attached to different locations/status. Upon this information the user is assigned different contexts and thus given an access to the database upon that.

A.6.3 Program Details

the program has two regular predicates: query(X,Y,Z,E,O,T) and p(X). the query predicate is constructed if a person has access based on his/her context.

- X is the name of the person
- Y is the first column in the table of the query
- Z is the second and so on for all the rest.

The context of the program has 3 categories.

(1) the user context: which defines the settings of the user through the location, role and time dimensions. The schema of this context is location:[loc], role:[role name], 'time':[daynight]

(2) the role context: which defines the privilege allowed for a user at a specific settings. The schema is similar to user context with the following addition priv:[privilege name].

(3) the privilege context: which defines the access information that the user allowed to see with the a specific role context. The schema is 'access':[col,col,col,col,col].

Where the domains for attributes are:

- loc=[inhospital,outhospital]
- role name=[user,dr,unknown]
- time=[morning,night]
- priv=[allpriv,viewpriv,userpriv]
- col=[name,address,phone,dob,history]

A.6.4 Code

```
# CONTEXT
allpriv={access:[name,address,phone,dob,history]}
userpriv={access:[name,address,phone,dob,none]}
drpriv={access:[name,phone,dob,history,none]}
nursepriv={access:[name,history,none,none,none]}
viewpriv={access:[name,none,none,none,none]}
rc1={location:[in_hospital],role:[dr],time:[morning],priv:[allpriv]}
rc2={location:[out_hospital],role:[dr],time:[morning,night],priv:[viewpriv]}
rc3={location:[in_hospital],role:[user],time:[morning],priv:[userpriv]}
rc4={location:[out_hospital],role:[user],time:[morning],priv:[userpriv]}
uc1={location:[out_hospital],role:[unknown],time:[morning]}
uc2={location:[out_hospital],role:[dr],time:[morning]}
uc3={location:[in_hospital],role:[dr],time:[morning]}

# FACTS
p(john)@uc2.
p(derek)@uc3.

# RULES:
p(X)@C:-p(X)@M,M<W,priv(C)@W.
query(O,X,Y,Z,E,T)@C:-p(O)@C,access(X,Y,Z,E,T)@C.


# --- RESULTS --- #
p(john)@uc2.
p(derek)@uc3.
**p(john)@viewpriv.
**p(derek)@allpriv.
**query(john,name,none,none,none,none)@viewpriv.
**query(derek,name,address,phone,dob,history)@allpriv.
```

A.7 context based path finder

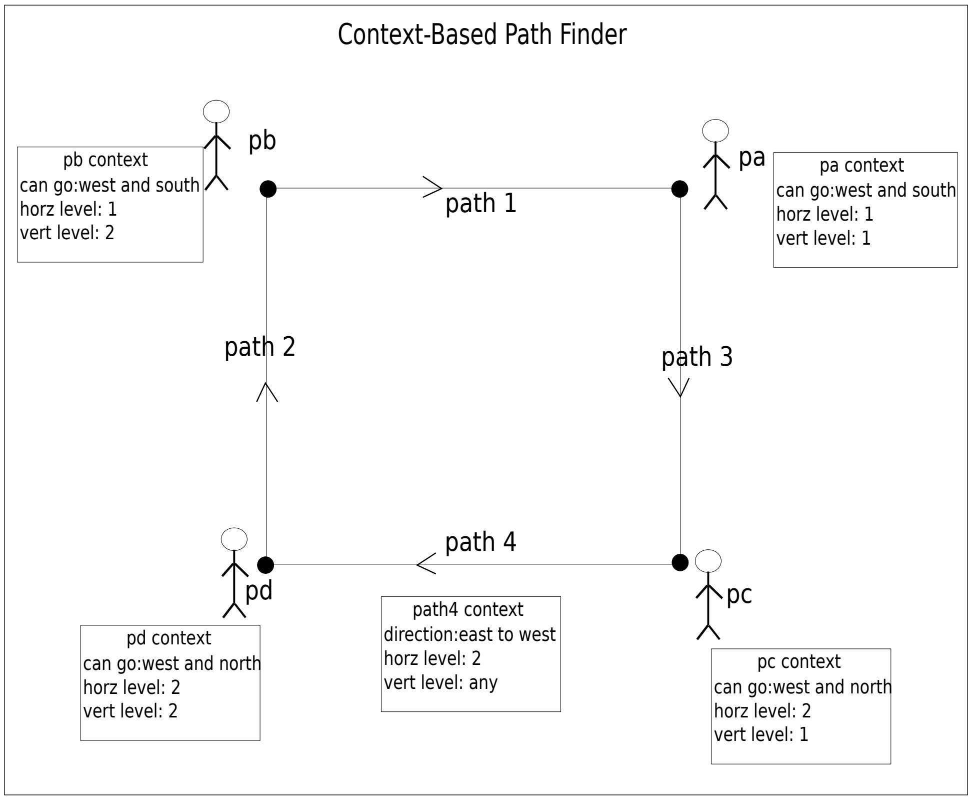


Figure A.7: Illustration of the path finder program

A.7.1 Goal

Add context-sensitivity to a traditional datalog problem and see the differences and the ability of contelog to express and solve that.

A.7.2 Overview

Every person has a context that allows him to take specific directions based on the context of the roads. If a person context matches or lined up with the road, a person can move from node to another through that road. If not, however, there is no path.

A.7.3 Program Details

The program has two types of contexts: the road context, and the person context. The road context has the following schema: dir:[ordinal,ordinal],level:[level number],vlevel:[level number]. The person context has the following schema: cango:[ordinal,ordinal],plevel:[level number], pvlevel:[level number] where: ordinal values domain = [e,w,s,n] level, plevel, vlevel and pvlevel = [natural number set (N)]

The road context as described as three dimensions: dir, level and vlevel. 'dir' is to describe the direction from->to. so the first argument should be the ordinal direction of the source and the second argument should be the ordinal direction of the target. This describes the road direction. level and plevel are to determine the coordinates of the line in the map.

the person context has also three dimensions cango, plevel and pvlevel. plevel and pvleve are just to set the current coordinates of the person in the map. However, cango is to determine which direction can a particular person go.

The idea of the rules of the program is to see the cango predicate in the person context, and see if the roads that he has access to line up with the settings of the person and his directions.

A.7.4 Code

```
# CONTEXT
rb1={dir:[w,e],level:[1]}
rb2={dir:[s,n],vlevel:[2]}
rb3={dir:[n,s],vlevel:[1]}
rb4={dir:[e,w],level:[2]}
pa={cango:[e,s],plevel:[1],pvlevel:[1]}
pb={cango:[s,w],plevel:[1],pvlevel:[2]}
pc={cango:[e,n],plevel:[2],pvlevel:[1]}
pd={cango:[n,w],plevel:[2],pvlevel:[2]}

# FACTS:
p(john,0,0,0)@pb.

```

```
# RULES:
p(X,W,M,C)@M:-p(X,D,F,U)@W,cango(Z,E)@W,dir(E,T)@C,
cango(T,Z)@M,level(Y)@C,plevel(Y)@W.
p(X,W,M,C)@M:-p(X,D,F,U)@W,cango(Z,E)@W,dir(Z,T)@C
,cango(E,T)@M,level(Y)@C,plevel(Y)@W.
p(X,W,M,C)@M:-p(X,D,F,U)@W,cango(Z,E)@W,dir(T,E)@C,
cango(Z,T)@M,pvlevel(O)@M,vlevel(O)@C.
p(X,W,M,C)@M:-p(X,D,F,U)@W,cango(Z,E)@W,dir(T,Z)@C,
cango(T,E)@M,pvlevel(O)@M,vlevel(O)@C.

# --- RESULTS --- #
p(john,0,0,0)@pa.
**p(john,pa,pc,rb3)@pc.
**p(john,pc,pd,rb4)@pd.
**p(john,pd,pb,rb2)@pb.
**p(john,pb,pa,rb1)@pa.
```

A.8 The Navigator

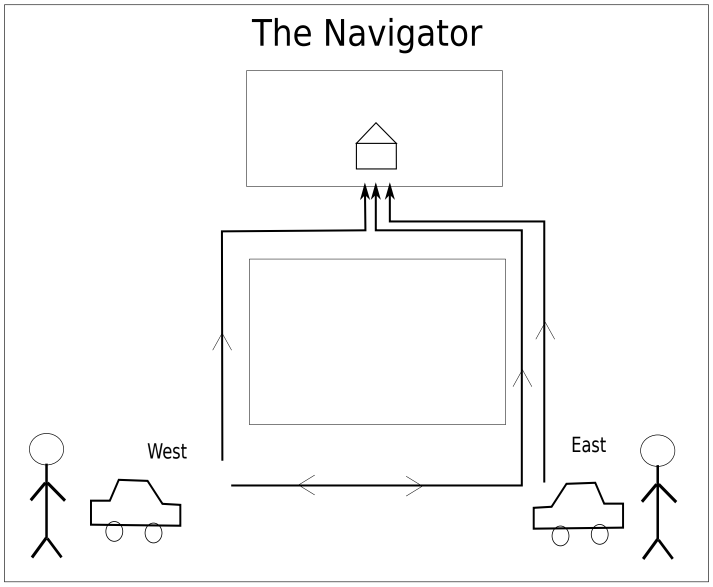


Figure A.8: Illustration of the navigator program

A.8.1 Goal

To show more complex program with more than categories of context and multiple predicates.

A.8.2 Overview

the program navigates the person based on the street context in the map. If there is a blockage, if it is one direction, are all factors that affect the navigation result. The person has also a context that might affect the navigation, if he is onfoot or by car.

A.8.3 Program Details

the program has 7 regular predicates, all of them are unary predicates. predicates are defined as follows:

- p(X) person name ‘X’
- ped(X) pedistrian name ‘X’
- route(X) a route for a person ‘X’
- drive(X) person ‘X’ is a driver (coming by car).
- better(X) a better route for person ‘X’
- best(X) the best route for person ‘X’.

As for contexts, the program uses two categories of contexts: street context and person context. The street context defines the street direction, status and traffic. The person context defines the method by which the person is coming, and the direction he/she is coming from.

A.8.4 Code

```
# CONTEXT
r1= {oneway:[n,e],status:[notblocked],traffic:[low]}
r2= {oneway:[n,w],status:[notblocked],traffic:[high]}
r3= {oneway:[w,n],status:[notblocked],traffic:[no]}
```

```
r4= {bidirection:[yes],status:[notblocked],traffic:[no]}
onfoot1= {method:[onfoot],ordinal:[e]}
onfoot2= {method:[onfoot],ordinal:[w]}
bycar3= {method:[bycar],ordinal:[e]}
bycar4= {method:[bycar],ordinal:[w]}

# FACTS:
# the symbol $ means a user defined predicate. The following four instructions
    makes a category cperson for some contexts
$cperson(onfoot1).
$cperson(onfoot2).
$cperson(bycar3).
$cperson(bycar4).
# the road category of contexts
$croad(r1).
$croad(r2).
$croad(r3).
$croad(r4).
p(ammar)@onfoot1.
p(ahmad)@bycar3.
p(amr)@bycar4.

# RULES
# if a person in context C comes onfoot then hes a pedistrian
ped(X)@C:-p(X)@C,method(onfoot)@C,$cperson(C).
# there is a route for a pedistrian if the road is not blocked.
route(X)@W:-ped(X)@C,status(notblocked)@W,$croad(W).
# a person is determined as a driver if he is in bycar context.
drive(X)@C:-p(X)@C,method(bycar)@C,$cperson(C).
# there is a route for the driver if the direction of the street matches the
    direction the driver wants to go to. and if the street is not blocked
route(X)@C:-drive(X)@W,ordinal(Y)@W,oneway(Y,ANY)@C,$croad(C),status(notblocked)@C.
route(X)@C:-drive(X)@W,ordinal(Y)@W,oneway(ANY,Y)@C,$croad(C),status(notblocked)@C.
# a better route is determined if there is a low traffic in the street context.
better(X)@C:-route(X)@C,traffic(low)@C,drive(X)@W.
```

```
# a best route is determined if there is no traffic at all in the street context.
best(X)@C:-route(X)@C,traffic(no)@C,drive(X)@W.


# --- RESULTS --- #
$cperson(onfoot1).
$cperson(onfoot2).
$cperson(bycar3).
$cperson(bycar4).
$croad(r1).
$croad(r2).
$croad(r3).
$croad(r4).
p(ammar)@onfoot1.
p(ahmad)@bycar3.
p(amr)@bycar4.
**ped(ammar)@onfoot1.
**drive(ahmad)@bycar3.
**drive(amr)@bycar4.
**route(ammar)@r1.
**route(ammar)@r2.
**route(ammar)@r3.
**route(ammar)@r4.
**route(amr)@r3.
**route(ahmad)@r1.
**route(amr)@r2.
**better(ahmad)@r1.
**best(amr)@r3.
```

A.9 Money Exchanger

A.9.1 Goal

show how simple and compact it is to right a useful real context based program. Also, to show how results get richer using the same rule only with richer contexts.

A.9.2 Overview

this program shows the currency of the country based on the location/context of the person.

A.9.3 Program Details

context contains the dimensions ‘currency’, and ‘location’. They have the attributes ‘name of currency’ and ‘name of location’ respectively.

the program uses the predicates: person(X,Y) and percontext(X,Y,Z) where X is name of the person, Y is the location of the person and Z is the currency used in that location.

A.9.4 Code

```
# CONTEXT
c1={currency:[euro],location:[france]}
c2={currency:[dollar],location:[usa]}
c3={currency:[cad],location:[canada]}

# FACTS:
person(ammar,canada).
person(zaki,france).

# RULES:
percontext(X,Y,Z)@C:-person(X,Y),location(Y)@C,currency(Z)@C.
# --- RESULTS ---#
person(ammar,canada).
person(zaki,france).
**percontext(ammar,canada,cad)@c3.
```

```
**percontext(zaki,france,euro)@c1.
```

A.10 The Translator

A.10.1 Goal

show the ability of contelog to use recursion, context operators, and user defined relations in one program.

A.10.2 Overview

program has some translations in context. These context based translations can be both translated to a common language through which a cross translation can be performed

A.10.3 Program Details

A.10.4 Code

```
# CONTEXT
cf1={meaning:[door,dar]}
cf2={meaning:[sky,asaman]}
ca1={meaning:[door,bab]}
ca2={meaning:[sky,samaa]}

# FACTS:

word(door).
word(sky).
$arabic(ca1).
$arabic(ca2).
$farsi(cf1).
$farsi(cf2).

# RULES:

```

```
english_arabic(X,Y)@C:-word(X),meaning(X,Y)@C,$arabic(C).
english_farsi(X,Y)@C:-word(X),meaning(X,Y)@C,$farsi(C).
arabic_farsi(Y,Z)@C+W:-english_arabic(X,Y)@C,english_farsi(X,Z)@W.
all_translations(X,Y)@C:-word(X),meaning(X,Y)@C.
across_translation(X,Y)@C+W:-all_translations(Z,X)@C,all_translations(Z,Y)@W,C!=W.

# --- RESULTS --- #
word(door).
word(sky).
$arabic(ca1).
$arabic(ca2).
$farsi(cf1).
$farsi(cf2).
**english_arabic(door,bab)@ca1.
**english_arabic(sky,samaa)@ca2.
**english_farsi(door,dar)@cf1.
**english_farsi(sky,asaman)@cf2.
**all_translations(door,dar)@cf1.
**all_translations(door,bab)@ca1.
**all_translations(sky,asaman)@cf2.
**all_translations(sky,samaa)@ca2.
**arabic_farsi(bab,dar)@ca1+cf1.
**arabic_farsi(samaa,asaman)@ca2+cf2.
**across_translation(dar,bab)@cf1+ca1.
**across_translation(bab,dar)@ca1+cf1.
**across_translation(asaman,samaa)@cf2+ca2.
**across_translation(samaa,asaman)@ca2+cf2.
```

A.11 Simple Magic Box Example

This example is used several times in the literature (Akman & Surav, 1996; Benerecetti et al., 2000) to bring out the importance of contextual reasoning. For our work, this example illustrates (1) practicality, (2) flexibility, (3) simplicity, and (4) expressive power. The example describes a box that is divided from top into 2×3 matrix in Figure A.9(b). It considers only three faces of the box, namely the *top, front, and side.* Each face of the box has its own view which is analogous to view points. Figure A.9(a) illustrates the actual view w.r.t. each those faces of the box. The following list establishes some notation to describe the box:

- l: means left square
- c: means center square
- r: means right square

The top face is described as a matrix, where a_1 means the top left corner of the top face in means the bottom left corner, and so on. To understand the relative positioning of the squares with respect to three faces, say with respect to b_1, we see from the top it is b_1, from front it is l, and from the side it is r. The rule in the game is that the box contains only two balls and someone can see the balls from any face in any square unless it is blocked by the second ball as it is the case in the front side.The idea is to figure out the position of the balls on the top view of the box by only knowing the position of the balls on the side and the front faces.

In (Benerecetti et al., 2000), they have used a framework named MCS that is based on propositional logic to model and solve this problem. Their program expresses all possible situations of the balls and boxes as a list of rules where they define two rules for each square in the face stating where would be the ball from the top view. That is, for the left square of the side face, the rules define if the ball 'exist', and if the ball 'does not exist' cases. A snippet of MCS example for the side face is shown below:

MCS Magic Box V(1) Example

```
(1)$ist(side, l) \iff ist(Top, \text{“}(a_1 \vee a_2 \vee a_3)\text{”})$
(2)$ist(side, \neg l) \iff ist(Top, \text{“}\neg(a_1 \vee a_2 \vee a_3)\text{”})$
(3)$ist(side, r) \iff ist(Top, \text{“}(b_1 \vee b_2 \vee b_3)\text{”})$
(4)$ist(side, \neg r) \iff ist(Top, \text{“}\neg(b_1 \vee b_2 \vee b_3)\text{”})$
```

where the balls are given as set of facts “$\{l, r\}$”. As the above set of rules considers “hard code” all possible situations, they can deal with any number of balls within the same

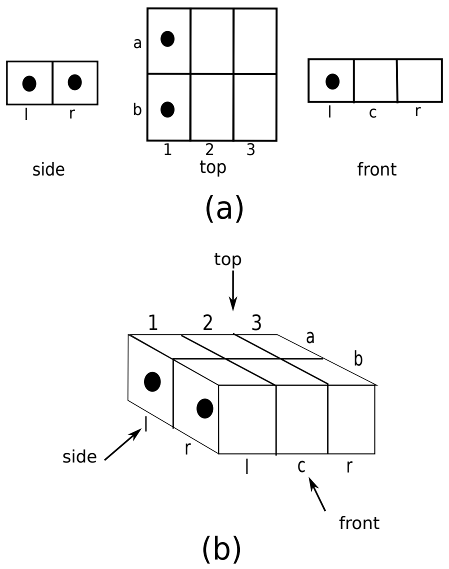


Figure A.9: Illustration of the magic box example

structure/number of boxes. However, if the number of boxes changes, which we will examine in the next example, the number of rules increases exponentially.

The goal of the example is to exercise the context to solve the puzzle by a contextual reasoner. The contextual information in this example is the three faces (view points). Namely, top, front, and side. Since top is inferred, only front and side are modeled in the following description.

Context of Magic Box Example

```
side = {'l':['ball'], 'r':['ball']}
front = {'l':['ball']}
top = {}
```

Where the content of the context only describes where the ball is located, specifically in which square. For instance, in front context, '*l*' is a dimension that describes the left square of the face, and its attribute is 'ball' which means it contains a ball. The other positions are not mentioned because they contain no balls. Now that contexts have been defined, the rules that defines each square of the top face is laid out below.

c-program for Magic Box Example

```
f1 : s(side).
```

```
r₁: a1(X)@top← l(X)@C,l(X)@W,C!=W,s(C).
r₂: a2(X)@top← l(X)@C,c(X)@W,C!=W,s(C).
r₃: a3(X)@top← l(X)@C,r(X)@W,C!=W,s(C).
r₄: b1(X)@top← r(X)@C,l(X)@W,C!=W,s(C).
r₅: b2(X)@top← r(X)@C,c(X)@W,C!=W,s(C).
r₆: b3(X)@top← r(X)@C,r(X)@W,C!=W,s(C).
```

Rule r_1 is to infer whether or not there is a ball in the square $a1$ in the top face. It asserts "if there is a ball in the left of the side face, and there is a ball in the left of the front face, then ball is located in a_1." Other rules are similar but differ in the positions of the squares. A rule is defined for each square in the top face, and so there is a total of six rules. The inequality $C! = W$ is to look at different faces. Also, the fact $s(side)$ is to restrict the context variable 'C' to only side face. This will ensure that the context variables in the rules refer to desirable contexts. The result of executing this program shows the squares in the top face at which the balls are, as shown below.

Results of Magic Box Example 2-balls

```
{s(side),a1(ball)@top,b1(ball)@top}
```

Here, we change the front context to $front = \{l : [ball], c : [ball]\}$, which means increasing one more ball. The results in this case are as follows:

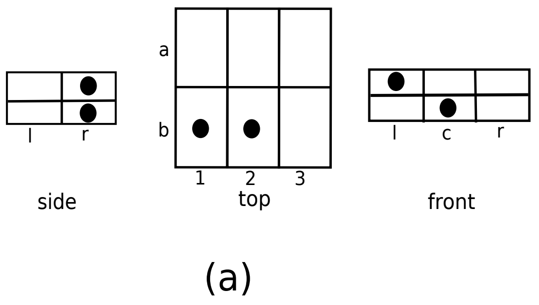
Results of Magic Box Example 3-balls

```
{s(side),a1(ball)@top,b1(ball)@top,b2(ball)@top}
```

A.11.1 Improved Magic Box Example

To show the scalability, and expressiveness of *Contelog* , let us consider this example again, but we increase the number of boxes to twelve instead of six. This means increasing the problem by a factor of '2' as shown in Figure A.10, assuming that the rules of the game remains the same.

First, if the same concept introduced in (Benerecetti et al., 2000) is used, the side face will require eight rules to express it, as each square in the face requires two rules, i.e. the



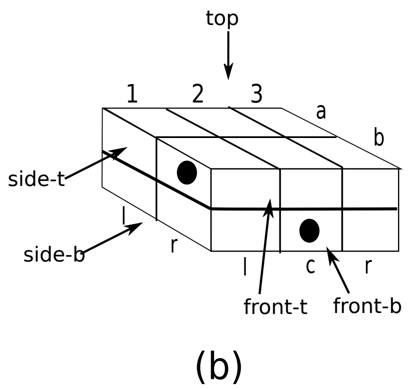


Figure A.10: Illustration of the improved magic box example

assertion and the negations. The top left square of the side face is expressed with two rules as follows,

MCS Magic Box V(2) Example

```
(1)$ist(side_t, l) \iff ist(Top, \text{“}(a_1 \vee a_2 \vee a_3)\text{”})$
(2)$ist(side_t, \neg l) \iff ist(Top, \text{“}\neg(a_1 \vee a_2 \vee a_3)\text{”})$
```

It is clear that the solution to the problem has also been increased by a factor of ‘2’, which is intuitive as one of the propositional logic disadvantage is the lack of expressiveness, and hence the rules are expressing all possible cases.

However, in *Contelog* only few changes in the code and context are required. First, we added two more contexts:$side_b$, and $front_b$. This addition is because new layer in the box has been added, and *Contelog* uses context to describe boxes. If we have three layers then three types of context for each face should be defined. Also, some facts are added to restrict the unification within the rule. For instance, the facts $s(side_t)$, and $s(side_b)$ are added so that the predicate $s(C)$ in the rules helps in restricting the substitution of the variable C to the contexts, $side_t$ and $side_b$. The context and code are shown below.

Context of Magic Box Version(2) Example

$side_t$ `= {r : [ball]}`

$side_b$ `= {r : [ball]}`

$front_t$ `= {l : [ball]}`

$front_b$ `= {c : [ball]}`

`top = {}`

c-program for Magic Box Version(2) Example

$f_1 : s(side_t).$

$f_2 : s(side_b).$

$f_3 : f(front_t).$

$f_4 : f(front_b).$

r_1: $a1(X)@top \leftarrow l(X)@C, l(X)@W, C! = W, s(C), f(W).$ `# s(C), and f(W) are added to`

r_2: $a2(X)@top \leftarrow l(X)@C, c(X)@W, C! = W, s(C), f(W).$ `# restrict the contexts to be`

r_3: $a3(X)@top \leftarrow l(X)@C, r(X)@W, C! = W, s(C), f(W).$ `# unified with the variable.`

r_4: $b1(X)@top \leftarrow (X)@C, l(X)@W, C! = W, s(C), f(W).$ `#‘s’ only considers ‘side’`

r_5: $b2(X)@top \leftarrow r(X)@C, c(X)@W, C! = W, s(C), f(W).$ `#contexts, while ‘f’`

r_6: $b3(X)@top \leftarrow r(X)@C, r(X)@W, C! = W, s(C), f(W).$ `#considers ‘front’`

The results become as follows.

Results of Magic Box Version(2) Example

{$s(side_t), s(side_b), f(front_t)$

$, f(front_b), b1(ball)@top, b2(ball)@top$}

A.12 Contelog-based Context-aware System: Tilt Detector

In this section we present context-aware system based on *Contelog* engine. As any context-aware system, there are main components that were developed/assembled to accomplish the experiment. Those components are as follows (illustrated in Figure A.11):

(1) Sensor: this is the device responsible for monitoring the contextual changes in an environment.

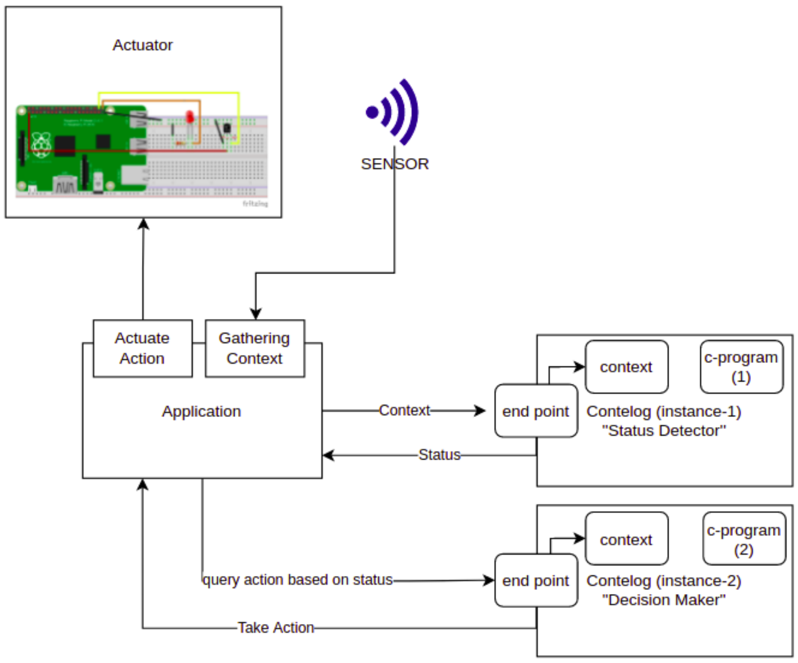


Figure A.11: Context-aware main components with *Contelog*

(2) Actuator: this unit is the electronic circuit that carries out the action in reality such as open gate, start the alarm,... etc.

(3) Contelog Engine Instances: one or more instance of *Contelog* that infer the decision according to the input context, and facts/rules of the system.

(4) Application (Control Unit): this unit orchestrates the actions/information among sensors, actuators, and *Contelog* instances.

This context-aware system detects inclination in surfaces using a Mercury-based Sensors to read the tilting, and sends it to the control unit to take the proper action. Those type of systems are very critical in several applications (Fraden, 2004). Some of these are the following.

- Roll Sensing: Inclination management using sensors provides a rollover or tip over warning for applications like construction equipment, and lift vehicles that operate in rugged terrains.

- Automotive uses: Sensing inclination has been used by Automobile manufacturers for lighting controls (for example, trunk lid lights), ride control (horizontal and vertical

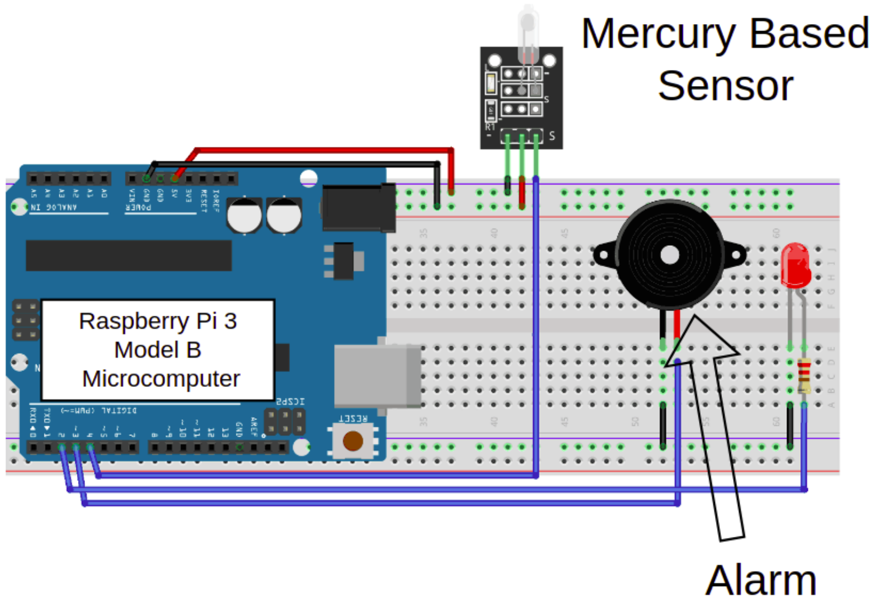


Figure A.12: Circuit for Tilt Detector

inclination), and anti-lock braking systems.

- Fall alarms: Work performed in confined space (such as a welder inside a tank) raises special safety concerns. Such context-aware systems are used to sound an alarm if a worker falls over.

- Bombs: A slight tilt can trigger a bomb. Context-aware systems are used to sense and trigger safety action in the case of this inclination.

Although the system idea is simple, it has far more crucial applications and usages. This motivated us to use *Contelog* a Formal-based, logic-based method to monitor such changes in context and infer the recommended actions to take. The beauty of using *Contelog* is that it is sound and complete, hence, the inferred results are formally verified.

A.12.1 Tilt Detector: The Electronic Circuit

We have implemented the circuit using Raspberry Pi Model B Microcomputer. The complete circuit is provided in Figure A.12. The circuit consists of four main components explained as follows:

- The Microcomputer: It represents the controller, the logic of the application and the controlling is performed here. The application is programmed using python. Full implementation of the controlling unit is provided in Appendix B.

- Alarm: A device that produces sound when given instruction to do so by controller. Basically when the tilt sensor is tilted.
- LED: This is to indicate another danger, used to trouble shoot in case a signal is coming from the sensor, but still the alarm does not go-off.
- Sensor: As explained earlier, Mercury-based tilt sensor to sense any inclination in surfaces.

A.12.2 Tilt Detector: The System Logic and Sequence

The controller constantly reads signals from the sensor. In case, a tilt/change is read, it communicates with the first *Contelog* instance, we refer to as, “status detector”. Based on the “status detector” results, the controller communicate with the second *Contelog* instance, that we call “decision maker”, in order to make the proper action. The communication between the control unit, and the two instances of *Contelog* is through a RESTful API that interfaces through web services. After receiving the results from the “decision maker”, there are two actions that can be made, either set/release the alarm. Set to start the alarm, and release to stop it. This whole process is depicted in Figure A.11.

A.12.3 Tilt Detector: Contelog Instance-1, Status Detector

This program takes context from the controller unit as an input-context. As it receives the context, it is triggered to run evaluation and infer results. Those results are communicated back to the controller. This c-program is given below. The goal of this program is to recommend the action to be taken on the alarm. However, the controller won’t take this action until it verifies the status of the alarm from Contelog Instance-2.

Basically, Status Detector consists of four contexts $\{c_1, c_2, c_3, c_4\}$ all having the same context schema. Each context defines a certain setting and the recommended action in the case of this setting. The context consists of two dimensions “position” and “alarm”. Position dimension can be “tilt”, “notilt”, “nuetral” which are the three settings of the sensor. Tilt means it is inclined, no tilt means horizontal, nuetral means it is stable for a long time. The context-aware system will treat nuetral and notilt with the same reaction. The alarm dimension can be "set" to recommend for setting the alarm off or "released" to turn the alarm off. For instance, if a sensor is in context "c1" (defined below), then it is on an inclined surface, and the alarm needs to be on. The context "input" is the input to the status detector *Contelog* program. It changes/re-entered by the controller, and every time it does, the program re-evaluate the context against the facts and rules.

The fact is *sensor*(1)@*input* which links the sensor name (in case we have more than one sensor) to the input context related to it. In our case we have one input context, and one sensor.

The program has one rule that infers the recommended actions "recommend(X,Y)@C" based on the predicates:

- sensor(S)@C: unifies with the fact sensor(1)@input.
- position(X)@C: unifies with the position of the input context.
- position(X)@W: unifies with the position of program contexts (c1-c4).
- alarm(Y)@W: recommends the action based on the contexts from (c1-c4).

```
# CONTEXT
c1={position:[tilt],alarm:[set]}
c2={position:[notilt],alarm:[set]}
c3={position:[notilt],alarm:[released]}
c4={position:[nuetral],alarm:[released]}
input={position:[tilt],alarm:[set]}
# Facts to link the sensor to the input context
sensor(1)@input.
# Rules
recommend(X,Y)@C:-sensor(S)@C,position(X)@C, position(X)@W,alarm(Y)@W.
```

A.12.4 Tilt Detector: Contelog Instance-2, Decision Maker

This program takes context from the controller unit as an input-context. As it receives the context, it is triggered to run evaluation and infer results. Those results are communicated back to the controller. The goal of this program is to direct the action to be taken after confirming the status of the alarm. That is , if it is “ON” and the action to take is “ON”, this c-program won’t take any action to avoid wasting resources. However, it only takes action if the action to be taken is different than the status of the alarm.

Basically, Decision Maker program consists of three contexts all having the same context schema. Each context defines a certain setting and the case, and action to be taken. Status dimension can be “set” or “released” which indicates the status of alarm within this context. The “action” dimension can also be “set” or “released”. Finally, the case dimension can be “tilt”, “notilt”, or “nuetral” which indicates the context of the sensor. The context “input” is the input to the decision maker *Contelog* program as an input from the status detector *Contelog* program through the control unit. It changes/re-entered by the controller, and every time it does, the program re-evaluates the context against the facts and rules.

The fact is *sensor*(1)@*input* which links the sensor name (in case we have more than one sensor) to the input context related to it. In our case we have one input context, and one sensor.

The program has one rule that infers the action to take “take_action(X)@W” based on the predicates

- sensor(S)@C: unifies with the fact sensor(1)@input.
- recommended(X,Y)@C: unifies with the input context coming from the controller passed by “status detector” *Contelog* program.
- status(Y)@W: unifies with the position of program contexts (release1,release2,setoff).
- action(E)@W: unifies with the action dimension in the contexts (release1,release2,setoff).
- case(Y)@W: unifies with the case dimension in contexts (release1,release2,setoff).

```
# CONTEXT
release1={status:[set],action:[release],case:[nottilt]}
release2={status:[set],action:[release],case:[nuetral]}
setoff={status:[released],action:[set],case:[tilt]}
input={recommend:[tilt,released]}
# Facts to link the sensor to the input context
sensor(1)@input.
```

```
# Rules
take_action(X)@W:-sensor(S)@C,recommend(X,Y)@C,status(Y)@W,action(E)@W,case(X)@W.
```

Appendix B

Contelog Tilt Detector Implementation

In this appendix, we explain the controller implementation for tilt detector application that was developed using *Contelog* . The program, context and hardware components are verbosely explained in Section 7.4.2.

B.1 The Controller

```python
# those libraries are the important libraries to communicate with Raspberry Pi 3
    hardware.
import RPi.GPIO as GPIO
import time
import requests
import json
from gpiozero import Button


def contact_contelog(alarm_status,program_name,input_context):
    #defining the url to conect to Contelog endpoint
    api_url= "https://e8c6-142-119-80-31.ngrok.io/contelogapp/system/api/"
    # set the type of encoding
    headers = {'content-type': 'application/json'}
    # this information is the login info to use Contelog endpoint
    # the program_name variable is the a program saved in the database of Contelog, it will be retrieved upon the reception of this request. The variable input_context is the input information sensed from the sensor and passed to Contelog.
    data = {'data':{'username': 'tilt_app', 'password': 'Abc@1234', 'program': program_name, 'context':input_context}}
    # those parameters are used to allow the api to be used in this session - it is only for authentication purposes.
    params = {'sessionKey': '9ebbd0b25760557393a43064a92bae539d962103', 'format': 'json', 'platformId': 1}

    input_data= {"code":"anycode.","context": "\n input={action:[" + action + "],alarm:[" + btn_status + "]}", "runoption":"Naive Method", "name_of_file":"ammar_example"}

    response = requests.post(api_url, data=json.dumps(data), headers=headers)
    print(response)

    return {'success': False, 'errorMsg': "ERROR", 'result': str(response.content)}


def setAlarmStatus(newstatus):
    """ This function to set the alarm status """
    global ALARM_STATUS
    ALARM_STATUS= newstatus
    return ALARM_STATUS

def getAlarmStatus():
    """ This function to get the alarm status """
    return ALARM_STATUS

def pushalarm_func(channel):
    """ this function is the main function that runs the infinite while loop that
        keeps monitoring the sensor and orchestrate between the two Contelog
        systems """
    ALARM_STATUS= getAlarmStatus()
    print(ALARM_STATUS)

    if ALARM_STATUS == "release":
        setAlarmStatus("set")
    else:
        setAlarmStatus("release")
        triggerPIN = 21
        tiltPIN=26
        togglePIN = 4
        setAlarmStatus("release")
        # the following instructions are to set the pins on the Raspberry Pi 3.
        GPIO.setmode(GPIO.BCM)
        GPIO.setup(triggerPIN,GPIO.OUT)
        GPIO.setup(tiltPIN,GPIO.IN)
        GPIO.setup(togglePIN,GPIO.IN,pull_up_down=GPIO.PUD_UP)
        # this is to control the response time and the monitoring time of the
            sensor
        GPIO.add_event_detect(togglePIN,GPIO.RISING,
        callback=pushalarm_func,bouncetime=300)
        print(" set ")
        buzzer = GPIO.PWM(triggerPIN,1000)
    while True:
        # this is to check if signal sent to from the mercury sensor is on or off.
        i = GPIO.input(togglePIN)
        if GPIO.input(tiltPIN):
            GPIO.output(triggerPIN,False)
            buzzer.stop()
            print("No Tilt Detected ")
        while GPIO.input(tiltPIN):
            # this is to keep monitoring the signal
            time.sleep(0.2)
        else:
            GPIO.output(triggerPIN,True)
            buzzer.start(10)
            print("ALERT ALERT ALERT")
            print("TILT DETECTED BE CAREFUL!")
            print("ALARM status is ", ALARM_STATUS)
            # send the request to the first system
            action = contact_contelog(ALARM_STATUS, "status detector", "tilt")
            # send the request to the second system according to the first system
            decision = contact_contelog(action, "decision maker", "tilt")
            print(decision)
            setAlarmStatus(decision)
```

References

Abiteboul, S., Hull, R., & Vianu, V. (1995a). *Foundations of databases: the logical level.* Addison-Wesley Longman Publishing Co., Inc.

Abiteboul, S., Hull, R., & Vianu, V. (1995b). *Foundations of databases: the logical level.* Addison-Wesley Longman Publishing Co., Inc.

Akman, V., & Surav, M. (1996). Steps toward formalizing context. *AI magazine*, *17*(3), 55.

Akman, V., & Surav, M. (1997). The use of situation theory in context modeling. *Computational Intelligence: An International Journal*, *13*(3), 427–438.

Alagar, V., Mohammad, M., Wan, K., & Hnaide, S. A. (2014a). A framework for developing context-aware systems. *EAI Endorsed Transactions on Context-aware Systems and Applications*, *1*(1).

Alagar, V., Mohammad, M., Wan, K., & Hnaide, S. A. (2014b, 9). A framework for developing context-aware systems. *EAI Endorsed Transactions on Context-aware Systems and Applications*, *14*(1). doi: 10.4108/casa.1.1.e2

Alagar, V., & Wan, K. (2008, October). Context based enforcement of authorization for privacy and security. In *Proc. of the first ifip wg 11.6 working conference on policies & research in identity management (idman 2007), lncs-ifip publications* (Vol. 261, p. 25-38).

Alsaig, A. (2017). *Contelog engine and documentation: A prototype environment for reasoning with contexts.* http://www.contelog.com. Retrieved from `http://www.contelog.com`

Alsaig, A., Alagar, V., & Nematollaah, S. (2020). Contelog: A declarative language for modeling and reasoning with contextual knowledge. *Knowledge-Based Systems*, *207*, 106403. doi: https://doi.org/10.1016/j.knosys.2020.106403

Alsaig, A., Alagar, V., & Shiri, N. (2019a). Declarative approach to model checking for context-aware applications. In P. C. Vinh & A. Rakib (Eds.), *Context-aware systems and applications, and nature of computation and communication.* Cham: Springer International Publishing.

Alsaig, A., Alagar, V., & Shiri, N. (2019b). Formal context representation and calculus for context-aware computing. In P. Cong Vinh & V. Alagar (Eds.), *Context-aware systems and applications, and nature of computation and communication* (pp. 3–13). Cham: Springer International Publishing.

Alsaig, A., Mohammad, M., & Alsaig, A. (2015, 04). Enhancing wearable systems by introducing context-awareness and fca. In (p. 264-271). doi: 10.1007/978-3-319-29236-6_26

Alvaro, P., Marczak, W., Conway, N., Hellerstein, J., Maier, D., & Sears, R. (2011). Dedalus: Datalog in time and space. *Datalog Reloaded*, 262–281.

Alviano, M., Faber, W., Leone, N., Perri, S., Pfeifer, G., & Terracina, G. (2010). The disjunctive datalog system dlv. In *International datalog 2.0 workshop* (pp. 282–301).

Apt, K., & Emden, M. (1982, 07). Contributions to the theory of logic programming. *J. ACM*, *29*, 841-862. doi: 10.1145/322326.322339

Attardi, G., & Simi, M. (1995). A formalization of viewpoints. *Fundamenta informaticae*, *23*(2, 3, 4), 149–173.

Barwise, J. (1989). *The situation in logic* (Vol. 17). Center for the Study of Language (CSLI).

Bazire, M., & Brézillon, P. (2005). Understanding context before using it. In *International and interdisciplinary conference on modeling and using context* (pp. 29–40).

Benerecetti, M., Bouquet, P., & Ghidini, C. (2000). Contextual reasoning distilled. *Journal of Theoretical and Ex-perimental Artificial Intelligence*, *I2*(3), 279-305.

Bishop, B., & Fischer, F. (2008). Iris-integrated rule inference system. In *International workshop on advancing reasoning on the web: Scalability and commonsense (area 2008).*

Borgo, S., Cesta, A., Orlandini, A., & Umbrico, A. (2019). Knowledge-based adaptive agents for manufacturing domains. *Engineering with Computers*, *35*(3), 755–779.

Bouquet, P., Giunchiglia, F., van Harmelen, F., Serafini, L., & Stuckenschmidt, H. (2004). Contextualizing ontologies. *Web Semantics: Science, Services and Agents on the World Wide Web*, *1*(4), 325 - 343. Retrieved from `//www.sciencedirect.com/science/article/pii/S1570826804000125` (International Semantic Web Conference 2003) doi: http://dx.doi.org/10.1016/j.websem.2004.07.001

Brass, S., & Stephan, H. (2017). Experiences with some benchmarks for deductive databases and implementations of bottom-up evaluation. *arXiv preprint arXiv:1701.00627*.

Brewka, G., & Eiter, T. (2007). Equilibria in heterogeneous nonmonotonic multi-context systems. In *Aaai* (Vol. 7, pp. 385–390).

Brézillon, P. (1996). Context in human-machine problem solving: A survey. *LIP*, *6*(1996), 029.

Brézillon, P. (1999). Context in problem solving: a survey. *The Knowledge Engineering Review*, *14*(1), 47–80.

Brezillon, P., & Abu-Hakima, S. (1995). Using knowledge in its context: Report on the ijcai-93 workshop. *AI magazine*, *16*(1), 87.

Brèzillon, P., & Gonzalez, A. I. (2014). *Context in computing: A cross-disciplinary approach to modeling real world.* Springer-Verlag, Berlin.

Buchanan, B. G., Shortliffe, E. H., et al. (1984). *Rule-based expert systems* (Vol. 3). Addison-wesley Reading, MA.

Buvač, S., & Mason, I. A. (1993). Propositional logic of context. In *Aaai* (pp. 412–419).

Carminati, B., Ferrari, E., & Perego, A. (2006). Rule-based access control for social networks. In *On the move to meaningful internet systems 2006: Meersman, robert and tari, zahir and herrero, pilar (eds.)* (pp. 1734–1744). Berlin, Heidelberg: Springer Verlag. Retrieved from `http://dx.doi.org/10.1007/11915072_80` doi: 10.1007/11915072_80

Carnap, R. (1947). *Meaning and necessity.* Chicago University Press, Enlarged Edition, 1956.

Ceri, S., Gottlob, G., & Tanca, L. (1989). What you always wanted to know about datalog (and never dared to ask). *IEEE Transactions on Knowledge and Data Engineering*, *1*(1), 146–166.

Chen, Z., Lin, F., Liu, H., Liu, Y., Ma, W.-Y., & Wenyin, L. (2002, Sep 01). User intention modeling in web applications using data mining. *World Wide Web*, *5*(3), 181–191. Retrieved from `https://doi.org/10.1023/A:1020980528899` doi: 10.1023/A:1020980528899

Clancey, W. J. (1983). The epistemology of a rule-based expert system—a framework for explanation. *Artificial intelligence*, *20*(3), 215–251.

Clark, H. H., & Carlson, T. B. (1981). Context for comprehension. In *Attention and performance* (pp. 313–330). Lawrence Erlbaum Associates, Hillside, NJ.

Costa, H., Furtado, B., Pires, D., Macedo, L., & Cardoso, A. (2012). Context and intention-awareness in pois recommender systems. In *6th acm conf. on recommender systems, 4th workshop on context-aware recommender systems, recsys* (Vol. 12, p. 5).

Decker, S., Erdmann, M., Fensel, D., & Studer, R. (1999). Ontobroker: Ontology based access to distributed and semi-structured information. In *Database semantics* (pp. 351–369). Springer.

Dey, A. K. (2001). Understanding and using context. *Personal and ubiquitous computing*, *5*(1), 4–7.

Dey, A. K., Abowd, G. D., & Salber, D. (2000). A context-based infrastructure for smart environments. In *Managing interactions in smart environments* (pp. 114–128). Springer.

Dey, A. K., Abowd, G. D., & Salber, D. (2001). A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction*, *16*, 97-161.

Dowley, D., Wall, R., & Peters, S. (1981). *Introduction to montague semantics.* Reidel Publishing Company.

Ejigu, D., Scuturici, M., & Brunie, L. (2007). An ontology-based approach to context modeling and reasoning in pervasive computing. In *Pervasive computing and communications workshops, 2007. percom workshops' 07. fifth annual ieee international conference on* (pp. 14–19).

et al., C. B. (2009). A survey of context modelling and reasoning techniques. *Pervasive and Mobile Computing.*

Ferreira, M., & Rocha, R. (2005). Coupling optyap with a database system. In *Iadis ac.*

for the Study of Language, C., for the Study of Language, I. U. C., Information, & Barwise, J. (1989). *Situations and small worlds.*

Fraden, J. (2004). *Handbook of modern sensors: physics, designs, and applications* (Vol. 3). Springer.

Gao, Q., & Dong, A. (2017). A conditional context-awareness ontology based personalized recommendation approach for e-reading. In *Robotic computing (irc), ieee international conference on* (pp. 365–370).

García, K., & Brézillon, P. (2015). A contextual model of turns for group work. In *International and interdisciplinary conference on modeling and using context* (pp. 243–256).

García, K., & Brézillon, P. (2017). Contextual graphs for modeling group interaction. In *International and interdisciplinary conference on modeling and using context* (pp. 151–164).

Giunchiglia, F. (1993). Contextual reasoning. *Epistemologia, special issue on I Linguaggi e le Macchine*, *16*, 345–364.

Grätzer, S. (1971). *Lattice theory: First concepts and distributive lattices.* W. H. Freeman, San Francisco.

Grau, B. C., Horrocks, I., Kaminski, M., Kostylev, E. V., & Motik, B. (2020). Limit datalog: A declarative query language for data analysis. *ACM SIGMOD Record*, *48*(4), 6–17.

Greco, S., & Molinaro, C. (2015).

Gu, T., Wang, X. H., Pung, H. K., & Zhang, D. Q. (2004). An ontology-based context model in intelligent environments. In *Proceedings of communication networks and distributed systems modeling and simulation conference* (Vol. 2004, pp. 270–275).

Guarino, N., et al. (1998). Formal ontology and information systems. In *Proceedings of fois* (Vol. 98, pp. 81–97).

Guha, R. V. (1991). *Contexts: a formalization and some applications* (Vol. 101). Stanford University Stanford, CA.

Halpin, T. (1998). Object-role modeling (orm/niam). In *Handbook on architectures of information systems* (pp. 81–103). Springer.

Held, A., Buchholz, S., & Schill, A. (2002a). Modeling of context information for pervasive computing applications. *Proceedings of SCI*, 167–180.

Held, A., Buchholz, S., & Schill, A. (2002b). Modeling of context information for pervasive computing applications. *Proceedings of SCI*, 167–180.

Held, A., Buchholz, S., & Schill, A. (2002c). Modeling of context information for pervasive computing applications. *Proceedings of SCI*, 167–180.

Henricksen, K., & Indulska, J. (2006). Developing context-aware pervasive computing applications: Models and approach. *Pervasive and mobile computing*, *2*(1), 37–64.

Interdisciplinary, & Series, I. C. (1997-). *Modeling and using context.* Lecture Notes in Artificial Intelligence (LNAI), Springer.

Johansson, N., & Löfgren, A. (2009). *Designing for extensibility: An action research study of maximizing extensibility by means of design principles* (B.S. thesis).

Jon, B., & John, P. (1983). *Situation and attitudes.* Cambridge, MA: Mit press.

Kass, A., Leake, D., & Owens, C. (1986). Swale: A program that explains. *Explanation patterns: Understanding mechanically and creatively*, *1*(1), 232–254.

Kinneavy, J. L. (1971). *A theory of discourse: The aims of discourse.* ERIC.

Kintsch, W., & van Dijk, T. (1978). Cognitive psychology and discourse: Recalling and summarizing stories. *Current Trends in Text Linguistics, de Gruyter, Berlin*, 61–80.

Knappmeyer, M., Kiani, S. L., Frà, C., Moltchanov, B., & Baker, N. (2010). Contextml: A light-weight context representation and context management schema. In *Wireless pervasive computing (iswpc), 2010 5th ieee international symposium on* (pp. 367–372).

Kofod-Petersen, A., & Mikalsen, M. (2005). Context: Representation and reasoning. *Special issue of the Revue d'Intelligence Artificielle on" Applying Context-Management*.

Korkea-Aho, M. (2000). Context-aware applications survey. *Department of Computer Science, Helsinki University of Technology*.

Lamperti, G., & Zanella, M. (2003). Rule-based diagnosis. In *Diagnosis of active systems: Principles and techniques* (pp. 193–233). Dordrecht: Springer Netherlands.

Lamport, L. (1980). Sometime is sometimes not never: On the temporal logic of programs. In *Proceedings of the 7th acm sigplan-sigact symposium on principles of programming languages* (pp. 174–185).

Leake, D. B. (2014). *Evaluating explanations: A content theory.* Psychology Press.

Lee, K.-C., Kim, J.-H., Lee, J.-H., & Lee, K.-M. (2007). Implementation of ontology based context-awareness framework for ubiquitous environment. In *Multimedia and ubiquitous engineering, 2007. mue'07. international conference on* (pp. 278–282).

Liang, S., Fodor, P., Wan, H., & Kifer, M. (2009). Openrulebench: An analysis of the performance of rule engines. In *Proceedings of the 18th international conference on world wide web* (pp. 601–610).

Liu, R., Zhang, X., Webb, J., & Li, S. (2015). Context-specific intention awareness through web query in robotic caregiving. In *Robotics and automation (icra), 2015 ieee international conference on* (pp. 1962–1967).

Lloyd, J. W. (1987). Foundations of logic programming. In *Symbolic computation.*

Loyola, W. (2007). Comparison of approaches toward formalising context: Implementation characteristics and capacities. *Electronic Journal of Knowledge Management*, *5*(2), 203–214.

McCarthy, J. (1963). *Situations, actions, and causal laws* (Tech. Rep.). DTIC Document.

McCarthy, J. (1993). *Notes on formalizing context* (Tech. Rep.). Stanford University.

McCarthy, J., & Buvac, S. (1997). Formalizing context (expanded notes).

Mechkour, S. (2007). Overview of situation theory and its application in modeling context. In *Seminar paper.*

Merton, R. K. (1973). *The sociology of science: Theoretical and empirical investigations.* University of Chicago press.

Moffett, J. (1968). *Teaching the universe of discourse.* ERIC.

Moldovan, D., Clark, C., & Harabagiu, S. (2005). Temporal context representation and reasoning. In *International joint conference on artificial intelligence* (Vol. 19, p. 1099).

Mooney, R., & DeJong, G. (1985). Learning schemata for natural language processing. *Urbana*, *51*, 61801.

Mueller, E. T. (2014). *Commonsense reasoning: an event calculus based approach.* Morgan Kaufmann.

Norvig, P. (1983). Frame activated inferences in a story understanding program. In *Ijcai* (pp. 624–626).

Orsi, G., & Tanca, L. (2011). Context modelling and context-aware querying. In *Datalog reloaded* (pp. 225–244). Springer.

Ostroff, J. S. (1989). *Temporal logic for real-time systems* (Vol. 40). Research Studies Press Advanced Software Development Series.

Perttunen, M., Riekki, J., & Lassila, O. (2009). Context representation and reasoning in pervasive computing: a review. *International Journal of Multimedia and Ubiquitous*

Engineering, *4*(4).

Reddy, M., & Gupta, A. (1995). Context interchange: a lattice based approach. *Knowledge-Based Systems*, *8*(1), 5–13.

Riguzzi, F. (2013). Mcintyre: A monte carlo system for probabilistic logic programming. *Fundamenta Informaticae*, *124*(4), 521–541.

R.Karni, & A.Gal-Tzur. (1990). Paradigms for knowledge-based systems in industrial engineering. *Journal of Artificial Intelligence in Engineering*, *5*(3), 126-141.

Sagonas, K., Swift, T., & Warren, D. (1999, 02). Xsb as an efficient deductive database engine. *ACM SIGMOD Record*, *23*. doi: 10.1145/191843.191927

Sarmah, A. K., Hazarika, S. M., & Sinha, S. K. (2015). Formal concept analysis: current trends and directions. *Artificial Intelligence Review*, *44*(1), 47–86.

Sato, M., Sakurai, T., & Kameyama, Y. (2001). A simply typed context calculus with first-class environments. In *Proceedings of flops'01: the 5th international symposium on functional and logic programming* (pp. 359–374).

S.C.Feng, W.Z.Bernstein, T.Hedberg, J., , & Feeney, A. (2017). Towards knowledge management for smart manufacturing. *ASME Journal of Computingand Information Science in Engineering*, *17*(3), 1-40.

Schilit, B., Adams, N., & Want, R. (1994). Context-aware computing applications. In *Mobile computing systems and applications, 1994. wmcsa 1994. first workshop on* (pp. 85–90).

Seligman, J., & Moss, L. S. (1997). Situation theory. *Handbook of logic and language*, 239–309.

Serafini, L., & Bouquet, P. (2004). Comparing formal theories of context in ai. *Artificial intelligence*, *155*(1), 41–67.

Shanahan, M. (1999a). The event calculus explained. In M. J. Wooldridge & M. Veloso (Eds.), *Artificial intelligence today: Recent trends and developments* (pp. 409–430). Berlin, Heidelberg: Springer Berlin Heidelberg. Retrieved from `https://doi.org/10.1007/3-540-48317-9_17` doi: 10.1007/3-540-48317-9_17

Shanahan, M. (1999b). The event calculus explained. In *Artificial intelligence today* (pp. 409–430). Springer.

Shehzad, A., Ngo, H. Q., Pham, K. A., & Lee, S. (2004). Formal modeling in context aware systems. In *Proceedings of the first international workshop on modeling and retrieval of context.*

Shoham, Y. (1991). Varieties of context. *Artificial intelligence and mathematical theory of computation: papers in honor of John McCarthy*, 393–408.

Sordo, M., Tokachichu, P., Vitale, C. J., Maviglia, S. M., & Rocha, R. A. (2017). Modeling contextual knowledge for clinical decision support. In *Amia annual symposium proceedings (published 2018)* (p. 1617-1624).

Strang, T., & Linnhoff-Popien, C. (2004). A context modeling survey. In *Workshop proceedings.*

Strang, T., Linnhoff-Popien, C., & Frank, K. (2003). Applications of a context ontology language. *Proceedings of SoftCOM 2003*, 14–18.

Tekle, K. T., & Liu, Y. A. (2010). Precise complexity analysis for efficient datalog queries. In *Proceedings of the 12th international acm sigplan symposium on principles and practice of declarative programming* (pp. 35–44).

Tekle, K. T., & Liu, Y. A. (2011). More efficient datalog queries: subsumptive tabling beats magic sets. In *Proceedings of the 2011 acm sigmod international conference on management of data* (pp. 661–672).

Ullman, J. D. (1989). Bottom-up beats top-down for datalog. In *Proceedings of the eighth acm sigact-sigmod-sigart symposium on principles of database systems* (pp. 140–149).

Ullman, J. D. (1990). *Principles of database and knowledge-base systems.* USA: W. H. Freeman & Co.

Umbrico, A., Cesta, A., Cortellessa, G., & Orlandini, A. (2020). A holistic approach to behavior adaptation for socially assistive robots. *International Journal of Social Robotics*, 1–21.

Van Emden, M. H., & Kowalski, R. A. (1976, oct). The semantics of predicate logic as a programming language. *J. ACM*, *23*(4), 733–742. Retrieved from `https://doi.org/10.1145/321978.321991` doi: 10.1145/321978.321991

Wan, K. (2006). *Lucx: Lucid enriched with context* (Unpublished doctoral dissertation). Concordia University.

Wan, K., Alagar, V., & Pacquet, J. (2005). An architecture for developing context-aware systems. In *Proceedings of 2nd international workshop on modeling and retrieval of* *context (marc2005)* (p. 48-62). LNCS, Springer-Verlag, Vol. 3946.

Wan, K., Alagar, V., & Paquet, J. (2005). A context theory for intensional programming. In *Workshop on context representation and reasoning (crr05), paris, france.(july 2005).*

Wang, X. H., Zhang, D. Q., Gu, T., & Pung, H. K. (2004a). Ontology based context modeling and reasoning using owl. In *Pervasive computing and communications workshops, 2004. proceedings of the second ieee annual conference on* (pp. 18–22).

Wang, X. H., Zhang, D. Q., Gu, T., & Pung, H. K. (2004b). Ontology based context modeling and reasoning using owl. In *Pervasive computing and communications workshops, 2004. proceedings of the second ieee annual conference on* (pp. 18–22).

Weyhrauch, R. (1980). Prolegomena to a theory of mechanized formal reasoning. *Journal of Artificial Intelligence*, *13*(1), 133-176.

Winograd, T. (2001). Architecture for context. *Human-Computer Interaction*, *16*, 401-419.