RULE-BASED IN-NETWORK PROCESSING FOR EVENT-DRIVEN APPLICATIONS IN WIRELESS SENSOR NETWORKS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

ÖZGÜR ŞANLI

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY
IN
COMPUTER ENGINEERING

JUNE 2011

Approval of the thesis:

RULE-BASED IN-NETWORK PROCESSING FOR EVENT-DRIVEN APPLICATIONS IN WIRELESS SENSOR NETWORKS

submitted by **ÖZGÜR ŞANLI** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen _____
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Adnan Yazıcı _____
Head of Department, **Computer Engineering**

Prof. Dr. Adnan Yazıcı _____
Supervisor, **Computer Engineering Dept., METU**

Assoc. Prof. Dr. İbrahim Körpeoğlu _____
Co-supervisor, **Computer Engineering Dept., Bilkent University**

Examining Committee Members:

Prof. Dr. Özgür Ulusoy
Computer Engineering Dept., Bilkent University _____

Prof. Dr. Adnan Yazıcı
Computer Engineering Dept., METU _____

Prof. Dr. Müslim Bozyiğit
Computer Engineering Dept., METU _____

Assoc. Prof. Dr. Ahmet Coşar
Computer Engineering Dept., METU _____

Assist. Prof. Dr. Sinan Kalkan
Computer Engineering Dept., METU _____

Date: _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: ÖZGÜR ŞANLI

Signature :

ABSTRACT

RULE-BASED IN-NETWORK PROCESSING FOR EVENT-DRIVEN APPLICATIONS IN WIRELESS SENSOR NETWORKS

Şanlı, Özgür

Ph.D., Department of Computer Engineering

Supervisor : Prof. Dr. Adnan Yazıcı

Co-Supervisor : Assoc. Prof. Dr. İbrahim Körpeoğlu

June 2011, 130 pages

Wireless sensor networks are application-specific networks that necessitate the development of specific network and information processing architectures that can meet the requirements of the applications involved. The most important challenge related to wireless sensor networks is the limited energy and computational resources of the battery powered sensor nodes. Although the central processing of information produces the most accurate results, it is not an energy-efficient method because it requires a continuous flow of raw sensor readings over the network. As communication operations are the most expensive in terms of energy usage, the distributed processing of information is indispensable for viable deployments of applications in wireless sensor networks. This method not only helps in reducing the total amount of packets transmitted and the total energy consumed by sensor nodes, but also produces scalable and fault-tolerant networks. Another important challenge associated with wireless sensor networks is that the possibility of sensory data being imperfect and imprecise is high. The requirement of precision necessitates employing expensive mechanisms such as redundancy or use of sophisticated equipments. Therefore, approximate computing may need to be used instead of precise computing to conserve energy. This thesis presents two schemes that distribute information processing for event-driven reactive applications, which are interested

in higher-level information not in the raw sensory data of individual nodes, to appropriate nodes in sensor networks. Furthermore, based on these schemes, a fuzzy rule-based system is proposed that handles imprecision, inherently present in sensory data.

Keywords: distributed information processing, complex event detection, reactive rules, fuzzy rule-based systems, event-driven applications, wireless sensor networks

ÖZ

OLAY GÜDÜMLÜ UYGULAMALAR İÇİN KABLOSUZ DUYARGA AĞLARINDA KURAL TABANLI AĞ İÇİ VERİ İŞLEME

Şanlı, Özgür

Doktora, Bilgisayar Mühendisliği Bölümü

Tez Yöneticisi : Prof. Dr. Adnan Yazıcı

Ortak Tez Yöneticisi : Doç. Dr. İbrahim Körpeoğlu

Haziran 2011, 130 sayfa

Kablosuz duyarga ağları, uygulamaların gereksinimlerine göre geliştirilmiş ağ ve bilgi işleme mimarilerine ihtiyaç duyan, uygulama bağımlı ağlardır. Kablosuz duyarga ağlarıyla ilgili olarak karşılaşılan en önemli sorun, pillerle çalışan duyarga cihazlarının sahip olduğu kısıtlı enerji ve bilgi işleme kaynaklarıdır. Her ne kadar verileri merkezi bir noktada işlemek en sağlıklı sonuçları üretmeyi sağlayacak olsa da, bu yöntemle ham duyarga verilerinin ağ üzerinde sürekli transfer edilmesi gerektiğinden, enerjinin verimli bir şekilde kullanıldığı bir çözüm değildir. Enerji kullanımı bakımından en pahalı işlemler ağ üzerinden veri transferi işlemleri olduğundan, gerçek yaşamda uygulanabilecek kablosuz duyarga ağı uygulamalarının geliştirilebilmesi için verilerin dağıtık bir yapıda işlenmesi bir zorunluluktur. Bu, ağda transfer edilen toplam paket sayısının ve duyarga cihazları tarafından harcanan toplam enerjinin azaltılması noktasında fayda sağlamakla kalmayıp, daha ölçeklenebilir ve hata toleransı daha fazla olan ağ mimarilerinin oluşturulması noktasında da yardımcı olacaktır. Kablosuz duyarga ağları ile ilgili bir diğer önemli sorun, duyarga verilerinin kusurlu olması ve kesin olmaması ihtimalinin yüksek olmasıdır. Kesinlik, yedeklilik ya da sofistike cihazların kullanımı gibi pahalı yöntemleri gerekli kılar. Bu nedenle, enerjinin korunması için kesin hesaplama yöntemlerinin yerine yaklaşık hesaplama yöntemlerinin kullanılması gerekebilir. Bu tezde, ham haldeki du-

yarga verilerinden çok daha üst seviye bilgiyle ilgilenen olay tabanlı reaktif uygulamalar için bilgi işlemeyi, ağdaki uygun duyarga cihazlarına dağıtan iki yöntem sunulmaktadır. Duyarga verilerinin kaçınılmaz olarak sahip olduğu kesin olmama durumunu ele almak amacıyla da, bu yöntemleri baz alan bulanık kural tabanlı bir sistem önerilmektedir.

Anahtar Kelimeler: dağıtık bilgi işleme, karmaşık olay tespiti, reaktif kurallar, bulanık kural tabanlı sistemler, olay tabanlı uygulamalar, kablosuz duyarga ağları

I dedicate this thesis to my family.

ACKNOWLEDGMENTS

I had the opportunity to work with Prof. Dr. Adnan Yazıcı and Assoc. Prof. Dr. İbrahim Körpeoğlu whom I would like to express my sincere gratitude. I would not be able to finish this thesis without their encouragement, advice and guidance.

I would also like to thank my thesis jury members Prof. Dr. Özgür Ulusoy, Prof. Dr. Müslim Bozyiğit, Assoc. Prof. Dr. Ahmet Coşar and Asst. Prof. Dr. Sinan Kalkan for their valuable comments and guidance.

I would like to thank my friend Metin Koç for his support that helped me not to lose my motivation, which could easily be the case in such a long journey. Our discussions not only about the academia and research but also about life in general have been the most enjoyable memories of the last few years.

Last but not the least, I would like to thank my family for their patience and belief in me. Their continuous love and support make everything in my life possible.

TABLE OF CONTENTS

ABSTRACT iv

ÖZ vi

DEDICATON viii

ACKNOWLEDGMENTS ix

TABLE OF CONTENTS x

LIST OF TABLES xiii

LIST OF FIGURES xiv

CHAPTERS

1 INTRODUCTION 1

1.1 Motivation 3

1.2 Contributions 5

1.3 Organization of the Thesis 6

2 BACKGROUND INFORMATION AND RELATED WORK 8

2.1 Wireless Sensor Networks 8

2.1.1 Sensor Nodes 10

2.1.2 WSN Applications 11

2.2 Event Processing 13

2.2.1 Events 14

2.2.2 Temporal Aspects of Events 15

2.2.2.1 Temporal Relations 16

2.3 Reactive Rules 17

2.4 Related Work 18

3 RULE DECOMPOSITION FOR DISTRIBUTED RULE PROCESSING . . . 22

3.1 Event-Condition-Action Rules 22
3.1.1 Events 24
3.1.1.1 Event Composition Operators 24
3.1.1.2 Event Detection and Composition 26
3.1.2 Conditions 29
3.1.3 Actions 29
3.2 Node Classification and Roles 30
3.3 Proposed Rule Decomposition Algorithm 33
3.3.1 Terms, Definitions and Notations 33
3.3.2 Decomposition Algorithm: RBDA 35
3.4 Equivalence of Initial Rule and Decomposed Sub-rules 40
3.5 Upper Bound on the Number of Sub-rules 44
3.6 Rule Processing 45
4 RULE DECOMPOSITION: REVISITED 49
4.1 Variables 49
4.2 Decomposition Using Rules with Variables: RBDA-V 51
4.3 Equivalence of Initial Rule and Decomposed Sub-rules 56
4.4 Upper Bound on the Number of Rules 60
4.5 Rule Processing 60
5 IMPRECISION IN WSN 62
5.1 Dealing with Imprecision 62
5.2 Fuzzy Logic 64
5.2.1 Fuzzy Sets and Membership Functions 65
5.2.2 Fuzzy Operators 67
5.2.2.1 Conjunction Operator 67
5.2.2.2 Disjunction Operator 70
5.3 Fuzzy Rules and Fuzzy Inferencing 71
5.3.1 Fuzzification 73
5.3.2 Defuzzification 76
5.4 Fuzzy Composition 77

5.4.1 Fuzzy Composition Graph 78
5.5 Rule Reduction 83
5.5.1 Distributed Fuzzy Composition 92
5.5.2 Fuzzy Composition of Distributed Rules with Variables . . 96
6 APPLICATION SCENARIO 104
6.1 Properties and Requirements 104
6.2 System Architecture 105
6.3 Healthcare Monitoring Rules 106
7 PERFORMANCE EVALUATION 110
7.1 Performance of the Algorithms 110
7.2 Simulations for In-Network Processing 113
7.2.1 Energy Model 115
7.2.2 Network Setup 116
7.2.3 Application Setup 117
7.2.4 Simulation Results 118
7.2.5 Healthcare Monitoring 119
8 CONCLUSIONS AND FUTURE WORK 122
REFERENCES 124
CURRICULUM VITAE 129

LIST OF TABLES

TABLES

Table 2.1 Sensor Node Specifications 12

Table 2.2 Sampling rates of different phenomena 12

Table 2.3 Information Processing Architectures 13

Table 5.1 The truth table of the binary conjunction 68

Table 5.2 The truth table of the binary disjunction 70

Table 5.3 Possible input combinations for 3 input parameters that each can be a member of one of 4 fuzzy sets 84

Table 7.1 Space needed by rule-bases 113

Table 7.2 Power required for sensor node's operations 116

Table 7.3 Number of packet transmissions for healthcare monitoring 121

Table 7.4 Number of hops required for a decision 121

LIST OF FIGURES

FIGURES

Figure 2.1 Sensor nodes (a) Mica-2 mote (b) BTnode (c) Spec-2 mote (d) WeC mote (e) PicoBeacon mote (f) TelosB mote 9

Figure 2.2 Components of a sensor node 10

Figure 3.1 Rule-based information processing in a node 27

Figure 3.2 Event Vector 28

Figure 3.3 Rule-based information processing in wireless sensor networks 30

Figure 3.4 Hierarchy of roles 32

Figure 3.5 Directed graph showing possible execution paths 33

Figure 5.1 An example for the characteristic function of a crisp set 66

Figure 5.2 An example for the membership function of a fuzzy set 66

Figure 5.3 Conjunction, disjunction and complement operations in bivalent logic . . . 67

Figure 5.4 Conjunction, disjunction and complement operations in multi-valued logic 68

Figure 5.5 Triangular membership function 74

Figure 5.6 Trapezoidal membership function 74

Figure 5.7 Gaussian membership function 75

Figure 5.8 Weighted directed acyclic graphs for fuzzy composition 79

Figure 5.9 Possible edge-vertex-edge triplets for a FCG-SR 81

Figure 5.10 Output Unification 85

Figure 5.11 Fuzzy composition graphs for three rules 85

Figure 5.12 Fuzzy composition graph for a combined rule 88

Figure 5.13 Fuzzy composition for a combined rule 90

Figure 5.13 Fuzzy composition for a combined rule (cont.) 91
Figure 5.14 Fuzzy composition graphs for sub-rules (2-layer decomposition) 93
Figure 5.15 Fuzzy Composition Graph for sub-rules (2-layer decomposition with variables) 95
Figure 5.16 Possible edge-vertex-edge triplets for a FCG-CR with variables 96
Figure 5.17 Fuzzy composition for a sub-rule in layer 1 101
Figure 5.18 Fuzzy composition for a sub-rule in layer 2 102
Figure 5.18 Fuzzy composition for a sub-rule in layer 2 103

Figure 6.1 Healthcare monitoring network 105

Figure 7.1 Number of initial rules vs. number of sub-rules generated 111
Figure 7.2 Percentage of processable inputs vs. number of sub-rules generated 111
Figure 7.3 Number of different layers vs. number of sub-rules generated 112
Figure 7.4 Ratio of reduction in the number of rules - Layer 1 114
Figure 7.5 Ratio of reduction in the number of rules - Layer 2 114
Figure 7.6 Ratio of reduction in the number of bytes 115
Figure 7.7 Total number of packets transmitted by sensor nodes 118
Figure 7.8 Average energy consumption by sensor nodes 119

CHAPTER 1

INTRODUCTION

The digital revolution that we have witnessed for the last few decades has completely changed how we live. In health, education, economy, entertainment, security and many other fields we become totally dependent on the digital world. It is hard to imagine a world without digital equipments and digital data. Whats more, as the advancements in technology and science continue, we demand more from our electronic devices: sense and react to physical phenomena that happen around them, just like humans and every other living being in the world do.

Sensors are the instruments that help the electronic devices in perceiving what is going on around their environment. They act as a bridge between the analog physical world and the digital world. Although the usage of sensors in industry is not a new phenomenon, not until now have they been used extensively. Today they are so popular that most of the electronic machinery contain one or more sensors installed. Examples include vehicles, computers, smart phones, home appliances, just to name a few. It would be fair to say that commercial success of some of today's smart phones lies behind the fact that the sensors inside those phones enable more attractive applications to be developed.

Besides the personal use of the sensors, the advancements in the short-range wireless communication technology and the ability to produce low-power, small sensing units in an affordable price open up a new world of possibilities for applications that require interaction with the real world. Application scenarios that might utilize sensors include military applications (battlefield surveillance, chemical/biological/nuclear attack detection, monitoring enemy soldiers), health applications (early diagnosis of diseases, patient monitoring), environmental applications (forest fire detection, flood or tornado detection, weather predictions, tracking extinct

species), home applications (building smart home environment via the sensors built into the home appliances), traffic applications (controlling or monitoring traffic, human-less or auto-controlled vehicles) and so on [2, 64].

For the last decade, researchers from all around the world have placed great attention to wireless sensor networks (WSN) which are networks of small devices that have sensing, processing and communication capabilities. These devices are called sensor nodes and might be used to sense various physical phenomena like temperature, pressure, humidity, proximity, speed, acceleration, radiation, light, sound, and so forth.

In a typical sensor network, sensor nodes sense a stimuli and send their findings to a special node called sink. The sink is the ultimate computational unit in the network and it connects the sensor network to the outside world. One of the most important characteristics of the sensor network is that sensor nodes have scarce energy and computational resources that makes the task of dealing with sensor networks challenging. Although, the sink is generally considered to have plenty of resources, the overall operational lifetime of the network depends on the longevity of the sensor nodes.

An important property of WSNs that requires attention is the existence of unreliable and imprecise sensory data. Deployment of sensor nodes in the environments that are open to the interference of any physical event or object, the unreliable cheap sensor hardware, and unreliable communication are the major causes of this unreliability. In order to preserve the decision reliability even in the presence of unreliable data, inexact computation techniques need to be employed in WSNs.

There are a few more properties of WSNs that differentiate them from ordinary networks. One of them is that sensor networks are data-centric networks. The nature of WSN data plays an important role in the design of collection, computation and transportation algorithms. Another important property is that they are application specific networks deployed generally for a particular purpose. As a result of this, WSNs face an important challenge: different kinds of applications have different sets of requirements, which makes it difficult to design generalized algorithms that can be used in all application scenarios. For example, some applications are deployed to gain insights on an unknown physical phenomenon. Such research oriented applications require all sensor readings to be recorded for offline analysis. On the other hand, expectation from a forest fire detection or health-care monitoring application is the notifica-

tion of emergency cases. As these examples show, appropriate algorithms for an application should be developed with its own specific requirements.

Sensor network applications can be roughly classified into the following four categories according to what initiates processing and communication operations:

- Demand-driven applications are the type of applications that the flow of data in the network begins as a result of a request from an external entity demanding that data. A query describing the request is constructed and disseminated into the network, and in response, sensor nodes collect and send the necessary information back to the demandant.
- Event-driven applications are the type of applications that the flow of data in the network begins after the occurrence of an event that has importance for the application. There is no need for continuous flow of sensor readings or periodic query requests.
- Some applications might require a mixture of event-driven and demand driven approaches. In this type of applications, after the detection and notification of a stimuli by sensor nodes, further requests are directed towards them for more information.
- A final type of applications requires the periodic and the continuous flow of sensory data in the network. All data periodically sampled by sensor nodes are processed and/or sent to the sink.

A remarkable majority of the sensor network applications are monitoring and detection applications. These applications are suitable to employ event-driven paradigm. In order to take advantage of the event-driven approach, suitable mechanisms need to be established that can produce high level information from raw data, decide if the information is of interest to the application and propagate the relevant information to appropriate nodes in the sensor network.

1.1 Motivation

Sensor nodes basically perform three operations: sensing, processing and communication. The energy required to accomplish these tasks is supplied by a battery, which is generally not replaceable. Thus, energy efficiency is of utmost importance for WSNs. The energy consumption of sensing and processing operations are far less compared to the energy consumption of

transmission and reception of data. As described in [49], transmitting 1 Kb of data at a distance of 100 meters costs 3 joules in a noise-free environment. Nevertheless, the same amount of energy is consumed by a general purpose processor with 100 MIPS/W capability for the execution of 3 million instructions. This suggests that reducing the volume of network traffic could help in conserving the scarce energy resources.

Sensor networks should employ distributed algorithms for both information processing and communication. Employing a central approach requires the data, which needs to be processed by WSN applications, to be transformed into some central location, and the results of the processing to be transferred back to the appropriate nodes in the network. Central approach has the advantage that a global view of the network is present while processing data. But there are several drawbacks which are unacceptable for a WSN:

- *Increased delay*: Propagation of the sensory data to the sink and the results of the computation back to the sensor or actor nodes introduces some delay which might be unacceptable for some type of applications like large-scale sensor network applications with real-time response requirements. Being able to respond to some of the events as a result of the processing carried out inside the network is a desirable property.
- *Increased network traffic*: Transmitting every bit of data in the network whether it is relevant to the application objectives or not increases the amount of network traffic.
- *Reduced sensor and network lifetime*: Inefficient consumption of the energy resources results in the reduced sensor node lifetime which in turn leads to reduced sensor network lifetime.
- *Non-scalability*: Adding more sensor nodes generally creates a negative influence on the WSN as the total number of packets transmitted in the network and passing through the nodes close to the sink will increase: causing more energy consumption and decreasing the sensor network lifetime.
- *Non-fault-tolerance*: In a centralized approach, the sink takes all the responsibility for information processing. As a result, the sink node becomes a single point of failure for the WSN.

In-network processing is a distributed information processing technique that is used to reduce the network traffic in sensor networks. If in-network processing is used, all or part of the data

is processed by the nodes inside the sensor network rather than processing all data at a central node. As a result, refined higher level information is transported inside the network instead of transporting raw sensory data.

When in-network processing is employed in a WSN, sensor readings can be processed near to the sensed area and the redundant or unnecessary data can be filtered out. In this way, the amount of traffic transported in the network decreases and this improves the energy utilization, prolongs the lifetime of both individual sensor nodes and the entire sensor network. Furthermore, processing data inside the sensor network results in a more fault-tolerant and scalable solution.

Employing rules instead of embedding the logic into application code has some advantages [68]. First, rules can be stored outside in a rule-base which improves the modularity, maintainability and extensibility of applications. Second, rules have a high-level declarative syntax, so they can easily be analyzed and optimized. Finally, rules provide a generic mechanism to express the reactive behavior contrary to the application code that is typically specialized to a particular type of reactive scenario.

1.2 Contributions

Information processing in WSNs is a challenging task due to their unique characteristics. In this thesis we study the subject of distributed processing of imprecise data in WSNs so to achieve the goal of energy efficient information processing. As a result of this research efforts, we can summarize the following as our main contributions:

- We present a mechanism that classifies the nodes of a hierarchical WSN. This mechanism is based on what sensor nodes can process and what they cannot. As a result of the classification, each sensor node has an associated role assigned to itself.
- Rule-based inferencing is used for the fusion of data in WSNs. While dealing with rules, the antecedents of them need to be described in a general and abstract form. So, we present a generalized form of the antecedents of the rules.
- We introduce a decomposition algorithm that optimally distributes rule processing into several nodes in a WSN.

- As the proposed decomposition algorithm has an exponential space complexity, we introduce a new way of representing rules and a new decomposition algorithm which works with this new rules.

- Fuzzy logic is considered to be a suitable method to deal with imprecision in WSNs, but fuzzy rule-based systems have the deficiency of large numbers of rules. In order to get rid of this problem, a scheme that reduces the number of rules in a fuzzy rule-base is proposed.

- For the composition of fuzzy membership values, a specific type of edge-labeled weighted directed acyclic graph is defined. Also, a fuzzy composition algorithm utilizing this new type of graph is presented that can be used by fuzzy inference engines.

- An application scenario that might benefit from the proposed approaches is described.

1.3 Organization of the Thesis

The organization of the thesis is as follows: In the second chapter, first we give introductory information about WSNs, in-network processing in WSNs, event processing and the rule-based event processing. In the second part we talk about other work that is related to our study.

In the third chapter, we first discuss about rules, give the necessary definitions that will be used in the rest of the thesis. Then we describe a mechanism that shows how to engineer roles in a sensor network. Based upon these, we describe how the rules that are intended to be run on a central processing unit can be distributed into the sensor network. Finally, we provide the proof that the distributed sub-rules achieve the same functionality as the non-decomposed original rule and present the complexity analysis of the algorithm.

In the fourth chapter, we elaborate on the shortcomings of the algorithm introduced in the third chapter. Then we show a new way of representing rules. Based on this new representation, we present a new algorithm that removes the deficiencies of the initial algorithm. Similar to the third chapter, functional equivalence of sub-rules and non-decomposed original rule is proved, and the complexity analysis of the new algorithm is given.

In the fifth chapter, we discuss the imprecision inherently present in WSNs and describe a

fuzzy rule-based system for dealing with it. Also, to eliminate the drawbacks of the traditional fuzzy rule-based systems regarding large numbers of rules, we propose a scheme that reduces the number of rules in a fuzzy rule-based system, and based on it, we present a fuzzy composition mechanism that can be used by fuzzy rule engines.

In the sixth chapter, we discuss the possible application scenarios that might benefit from our approach. We select the health-care monitoring application as the sample application scenario. We give the details of the architecture of such an application, and provide a set of health-care monitoring rules.

In the seventh chapter, we present the results of experiments that are conducted to examine the performance of the algorithms. We also present the results of the simulations that show the benefits of employing an in-network processing strategy.

Finally in the eighth chapter, we summarize our study and discuss about the possible future directions of it that will improve the presented approaches.

CHAPTER 2

BACKGROUND INFORMATION AND RELATED WORK

2.1 Wireless Sensor Networks

Due to the advancements in wireless communication technologies, micro-electro-mechanical systems (MEMS), and low-cost and miniaturized integrated circuitry fabrication, it is possible to develop small devices that combine the sensing, processing and communication operations. Figure 2.1 shows the examples of such tiny devices.

A network of small sensing devices using wireless communication technology is called a WSN. There are several properties of WSNs that differentiate them from ordinary networks that we are accustomed to [51]. The following are the most important ones:

- *Deployment.* Wireless sensor networks are deployed in physically harsh environments not in air-conditioned, protected system rooms.
- *Data-centricity.* The most important thing in the network is the sensed data. However, which node senses or which node routes that data is not important for most of the applications. One crucial requirement is to transmit that data to the sink.
- *Resource constraints.* There is a tradeoff between being small and having plenty of resources. As the sensor nodes need to be small, their processing, communication and power resources are scarce.
- *Energy efficiency and lifetime.* One of the most important challenge associated with wireless sensor networks is the efficient utilization of the energy resources. While the improvements in the electronic and mechanical components continue, the improvements in the battery is limited.

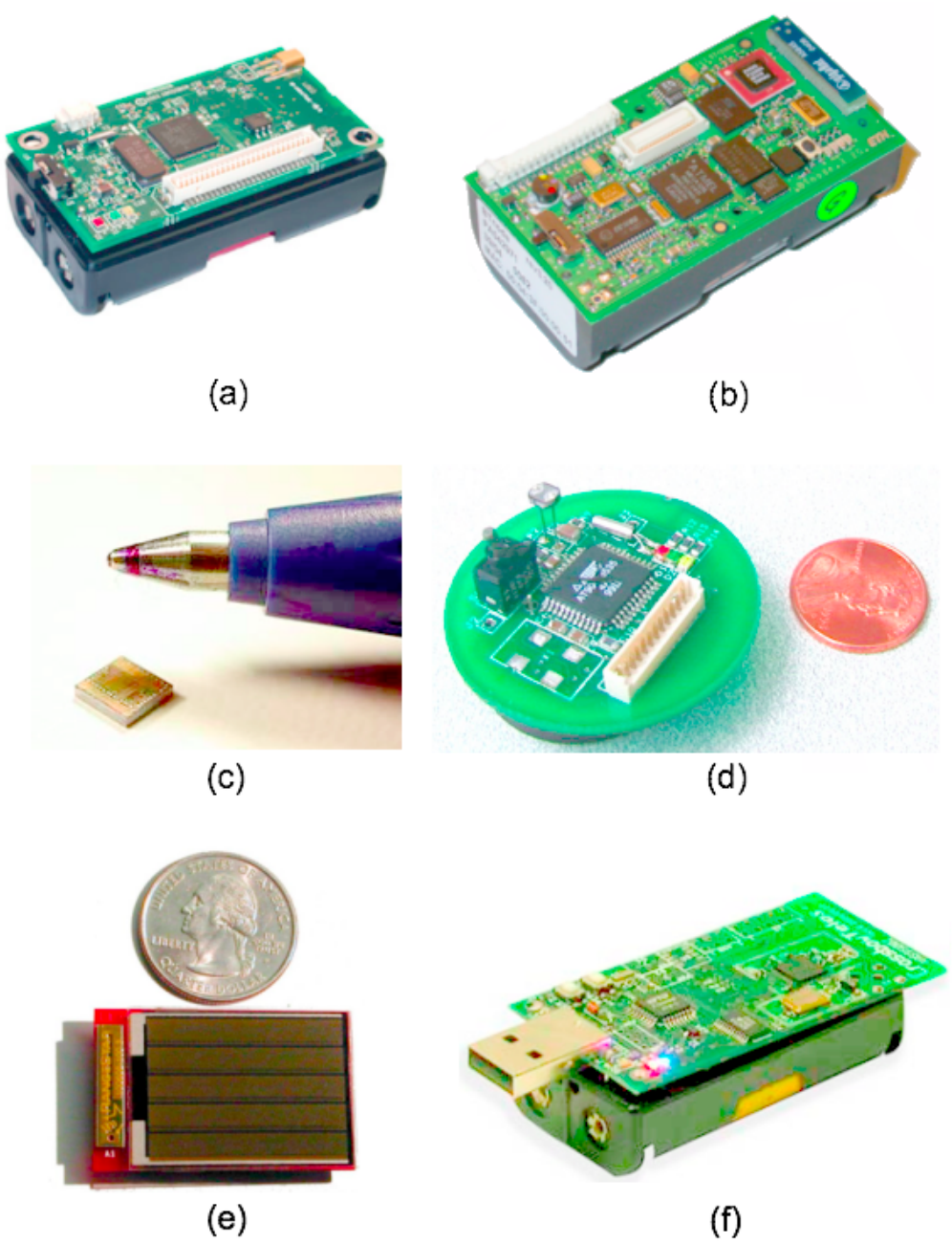


Figure 2.1: Sensor nodes (a) Mica-2 mote (b) BTnode (c) Spec-2 mote (d) WeC mote (e) PicoBeacon mote (f) TelosB mote

- *Distributed operation.* Centralized operations are simply too costly to be employed in wireless sensor networks. Lightweight distributed algorithms are needed for processing and networking operations.

- *Autonomous/unattended operation.* Once deployed, sensor nodes need to operate autonomously. They remain operational until their batteries die out. After that, they would

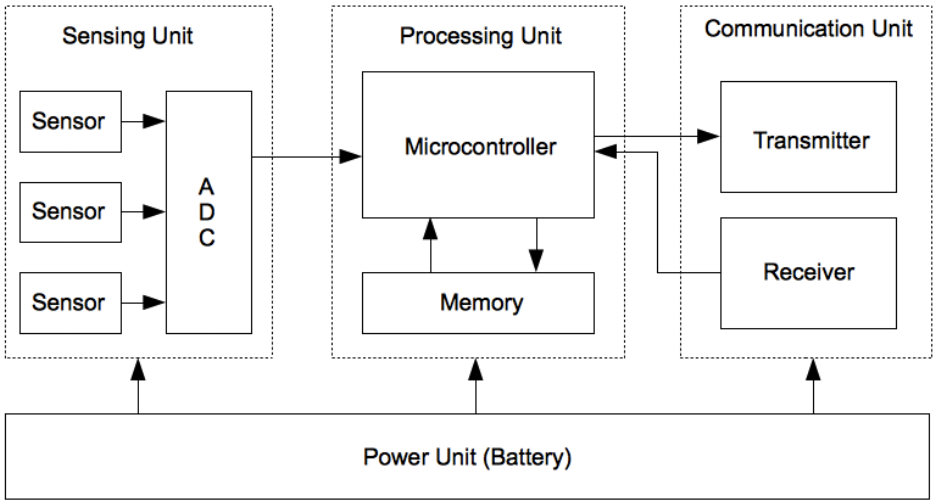


Figure 2.2: Components of a sensor node

be simply forgotten. They cannot be used for another purpose. While they are operational, there is no intervention in any way by humans that change some property of them. The sensor network need to self-organize itself starting right after the deployment and adapt to the changes like the broken nodes, obstacles that prevent communication, etc.

- *Non-reliability.* In addition to the possibility of deploying sensor nodes in physically harsh environments, cheap hardware manufacturing process, and use of wireless networking open up a plethora of possible failures associated with sensor nodes and sensor networks. Failures lead to non-reliability.

2.1.1 Sensor Nodes

Starting from the late 90s, several sensor node platforms, also called mote, have emerged up until today. Mica and its successors Mica-2 and Mica-Z [40], Telos [40], Intel mote [43], BTnode [11], Pluto and its successor Shimmer mote [55] are a few examples of such nodes.

A sensor node is the smallest element of the wireless sensor networks. Figure 2.2 shows typical components of a sensor node:

1. *Sensing Unit:* Sensors generate analog electrical signals representing the physical phenomenon. Then, these signals are converted to digital signals by an analog-to-digital-converter (ADC). Sensing hardware generally consume small amounts of power. However, what is sensed has a direct influence on the requirements for computational and power resources. The requirement that how often the sensed phenomenon needs to be sampled varies. Table 2.2 shows the sampling rates of a representative set of physical phenomena. As the requirement for the sampling frequency increases, there is a need for better computational resources. More computational resources means more energy consumption.

2. *Processing Unit:* Generally consists of a micro-controller and additional memory. It is responsible for manipulating data. Micro-controllers that can be given as examples include Texas Instrument's MSP430, Intel's 8051, Atmel's ATmega103/128, and ARM's ARM7TDMI.

3. *Communication Unit:* Responsible for enabling the communication with other entities in the sensor network. Due to the flexibility that is provided by wireless technologies, RF communication is generally used. Initial sensor nodes used IEEE 802.11 WiFi standards for wireless communication. However, soon it turned out to be too costly for such small devices. Then new set of standards has emerged. Among the most popular ones are 802.15.4 [28] and ZigBee [66]. Low range and low power RF transmitting and receiving hardware include Chipcon's CC1000 [13], CC1100, CC2420 [14], Nordic's nRF2401 [45], RFM's TR1000 [50], and Infineon's TDA5250 [29].

4. *Power Unit:* Responsible for supplying the necessary power for the operation of the other components. Generally an irreplaceable battery is used for this purpose. Compared to other components of a sensor node, the battery is the least improved component. There is more space for technological improvements in the field of science of battery.

2.1.2 WSN Applications

There are so many application scenarios that we can think of. The following is a brief list of them:

Table 2.1: Sensor Node Specifications [60]

	Mica2Dot	*MicaZ*	*MSB430*	*Sun Spot*
Microcontroller	ATmega128L	ATmega128L	TI MSP430F1612	ARM7
Architecture	8-bit	8-bit	16-bit	32-bit
SRAM	4 KB	4 KB	5 KB	256 KB
Flash	128 KB	128 KB	55 KB	2 MB
Radio	CC1000	MPR2400	CC1020	CC2420
RF Band	315-916 MHz	2,4 GHz	402-915 MHz	2,4 GHz

Table 2.2: Sampling rates of different phenomena [27]

Phenomena	**Sample Rate (Hz)**
Very low frequency	
Atmospheric temperature	0.017 - 1
Barometric pressure	0.017 - 1
Low frequency	
Heart rate	0.8 - 3.2
Volcanic infrasound	20 - 80
Natural seismic vibration	0.2 - 100
Mid frequency (100 Hz - 1000 Hz)	
Earthquake vibrations	100 - 160
ECG (heart electrical activity	100 - 250
High frequency (> 1 KHz)	
Breathing sounds	100 - 5 K
Industrial vibrations	40 K
Audio (human hearing range)	15 K - 44 K
Audio (muzzle shock-wave)	1 M
Video (digital television)	10 M

Table 2.3: Comparison of centralized and distributed information processing architectures for WSNs

	Centralized	**Distributed**
Accuracy	global, more accurate results	less precise, localized results
Fault Tolerance	single-point of failure	highly fault-tolerant
Scalability	non-scalable	highly scalable
Manageability	easy to manage	difficult to construct and manage
Synchronization and Coordination	required for communication operations	required both for communication and data processing
Redundancy	redundant data and communication	possibility to eliminate redundant data and communication
Energy Efficiency	less efficient	depends on the implemented protocols/algorithms
Traffic Volume	high	low - medium
WSN Lifetime	short	depends on the implemented protocols/algorithms
Processing	Complex operations	Simple operations
Delay	long	short
Resources	Plentiful	Limited

- *Tracking Applications.* Examples include enemy tracking, animal tracking, vehicle tracking, and human tracking.

- *Monitoring Applications.* Examples include habitat monitoring, patient monitoring, environmental monitoring, structural health monitoring, inventory monitoring, machine monitoring, volcano monitoring, and coal mine monitoring.

- *Detection Applications.* Examples include forest fire detection, flooding detection, early earthquake detection, and other surveillance applications.

- *Smart Surrounding Applications.* Examples include smart home, and smart building.

2.2 Event Processing

Event processing is the process of detecting, consuming and reacting to events. Managing events are the main concern in many disciplines. For example, Security Event Management products in IT security correlate events coming from different sources in order to detect if there is anything happening abnormal [4, 52]. In wireless sensor networks, almost all track-

ing and monitoring applications depend on the occurrence and processing of events and the reactions taken.

2.2.1 Events

The term event is described in dictionary as ”something that takes place which is significant, interesting or unusual”. Landing of an airplane, an earthquake or a volcanic eruption, entering or leaving a building, pressing a key in keyboard, hardware or software interrupts in a computation system, completion of a transaction in a database system etc. can be given as examples to events from different domains.

In the context of wireless sensor networks we define an event as any change in the state of the data being monitored where the change has significance for the application. Arrival of a network packet or a value exceeding some threshold might be typical examples of the sensor network events.

Events can be categorized as primitive and composite events:

- **Primitive Event:** A primitive event, also called simple or raw event, is the single and atomic occurrence of an event.
- **Composite Event:** A composite event, also called complex event, is an abstract event which is derived as a result of the composition of primitive or other complex events using operators of an event algebra. Events that constitute the composite event are called component events....

Most of the time, a primitive event on its own is not enough for drawing conclusions or taking actions. In order to capture a more real and complicated application scenarios, complex events need to be used. For logical composition of the events, several event algebras have been proposed: SNOOP [67], SAMOS [22], NAOS [15], EPL [41], CHIMERA [39], ODE [24], and so on. Typical event algebra operators that have been used by them are:

- **Conjunction -** $(a \wedge b)$: Occurrence of both a and b.
- **Disjunction -** $(a \vee b)$: Occurrence of either a or b.

- **Negation -** $(!a)$: Non-occurrence of a.
- **Sequence -** $(a; b)$: Occurrence of a before b.
- **Iteration -** (a^*) : Multiple occurrences of a.

2.2.2 Temporal Aspects of Events

Temporal knowledge about events is an indispensable element of event processing. A complex event pattern might require the fulfillment of special temporal constraints between the component events. For instance, the complex event "a followed by b" require not only the occurrence of events a and b, but also the satisfaction of the temporal condition that a happens before b.

In order to be able to evaluate temporal relations between events, the following time-related information need to be used:

- **Event Detection Time:** It is the time instant in which the event is detected. This is trivial for a simple event as there is only one time instant in which the event is detected. Nevertheless, detection time of a composite event is the detection time of the terminating event that is the lastly detected component event.
- **Event Duration:** Events can be categorized as instantaneous events and durative events. An instantaneous event is the one happens in an instant and vanishes immediately after occurrence. It is not possible to detect the same event after that instant. On the other hand, a durative event lasts for a specific amount of time in which the conditions that describe the event hold true. A gun shooting can be given as an example to an instantaneous event, whereas the event that human body temperature being above 39° Celcius is a durative event. Generally the conditions that lead to the detection of events does not disappear immediately, therefore durative events are more common compared to instantaneous ones. The time interval for which the state that create the event remain unchanged is called the event duration.
- **Event Validity Interval:** An event, whether it be an instantaneous or durative event, can be consumed by the event processing system only within a specific time interval. It cannot be used before or after that time interval. The interval in which the event

can be used by the event processing system is called the event validity interval. Note that although an instantaneous event does occur in an instant, that event can still be composed by other events for deriving composite events at a later time.

There are two basic models for event timestamps used in event processing:

- **Point-based timestamps:** A point-based timestamp refers to a single value in time domain; i.e., it represents a time instant. In event processing, it has fairly limited usage such as in the systems that only deal with instantaneous primitive events. However, almost none of the complex events and durative events could be assigned a point-based timestamp. As shown in [], use of point-based timestamps even for instantaneous events might yield some incorrect semantic evaluations.
- **Interval-based timestamps:** An interval-based timestamp is described by a starting and an ending time, and written as *(starting_time, ending_time)*. If there is a need to refer to an instant, an interval-based timestamp can be used to describe it, which is the case that starting and ending times are the same. Interval-based timestamps are more suitable for complex event processing.

2.2.2.1 Temporal Relations

Allen's Interval Algebra enables to reason using intervals. Although 13 relations has been given in [3], as 6 relations are simply the inverse of their counterparts, we can talk about 7 basic interval relations. Let's assume that events a and b have the timestamps $(a.t_s, a.t_e)$ and $(b.t_s, b.t_e)$. Then the formal definition of the temporal relations are as follows:

- **BEFORE:** If a happens before b, then the ending time of event a needs to be smaller than the starting time of event b: $a.t_e < b.t_s$
- **MEETS:** If a meets b, then the ending time of a is equal to the ending time of event b: $a.t_e = b.t_s$
- **OVERLAPS:** If a overlaps b, then the starting time of event a is smaller than the starting time of event b and the ending time of a is smaller than the ending time of b: $(a.t_s < b.t_s)$ & $(a.t_e < b.t_e)$

- **STARTS:** If a starts b, then the starting time of both a and b need to be the same: $a.t_s = b.t_s$

- **FINISHES:** If a finishes b, then the ending time of both a and b need to be the same: $a.t_e = b.t_e$

- **DURING:** If a happens during b, then the starting time of b need to be smaller than the starting time of a and the ending time of a need to be smaller than that of b: $(a.t_s > b.t_s)$ & $(a.t_e < b.t_e)$

- **EQUAL TO:** If a is equal to b then both the starting time and the ending time of a and b need to be the same: $(a.t_s = b.t_s)$ & $(a.t_e = b.t_e)$

2.3 Reactive Rules

It is possible to express the events of interest and the reactive behavior inside the application or outside in a rule-base. In the first approach, software developer writes down all the event handling operations inside the application. He use interrupts that alert the occurrence of some event and event handlers that implement the reactive behavior. In the second approach, the complex events which represent the conditions that need to be met, and the actions are placed outside the application in a rule-base [60, 7].

The rule-based approaches for reactive event processing is categorized as follows [46]:

- Production Rules: They are in the form of *"WHEN <conditions> DO <actions>"*. A change in the state captured by the condition results in the firing of the rule. The results of the actions may further change the state that needs to be captured by other rules. A forward chaining mechanism is used as the rule processing semantic in order to realize such a chain of state changes . Production rules have been used in expert systems since 1980 and there are several successful commercial applications of them.

- *ECA Rules:* First explored by database community in the field of active databases. They are in the form of *"ON <event> IF <conditions> THEN <actions>"*. The occurrence of the event trigers the rule. Next, conditions are evaluated. If they satisfy, then the related actions are taken.

- *Rule-Based Event Processing Languages:* A combination of Complex Event Processing (CEP) for real-time event detection and reactive rules for declarative representation and appropriate reactions.

2.4 Related Work

One of the important topic in sensor networks is the efficient query processing. There are several approaches in handling queries in sensor networks. One of the approaches is that the query is handled by a central node. The sensor nodes send their data into the central node for storage. The central node based on this data answers the queries. Another approach is the COUGAR approach [63]. In this approach, a query optimizer is located at the gateway node to generate distributed query plans after receiving queries from the outside. The query plan is generated according to catalog information (Catalog keeps the meta-data such as remaining usable battery of sensor nodes, sensor positions, node density of a specific area, system workload, etc. that are used to create better query plans) and query specification. The query plan specifies the data flow between sensor nodes and the exact computation plan at each sensor node. When such a plan is generated, it is disseminated to all relevant sensor nodes. Data records propagates to gateway node while in-network processing happens on-the-fly.

Yet another approach is the one that is employed by the ACQUIRE [53]: Acquire is a data-centric querying mechanism that address complex, one-shot queries for replicated data. Query is considered to be an active entity and propagated through the network until it is fully resolved. While the query propagates in the network, each active node, node that currently handles the active query, use the information from all other nodes within d-hop range to resolve the query. If the information kept in the cache is not up-to-date, an update process is started. The active node sends a request to be replied by all nodes that are at maximum d hops away. If the query cannot be resolved fully at this step, a next node is selected and the query is forwarded to that node. The node selection process might be totally random or may be based on much more intelligent decisions (e.g., the node that can resolve maximum). At each step, active query gets smaller and smaller until the last piece of original query is finally resolved at which point the query result is sent back to the query issuer.

A formal model is generally used for the analysis and the verification of the modeled sys-

tem. A survey in formal approaches to model and analyze distributed and concurrent systems shows that most of the popular approaches are based on the theoretical models such as finite state machines, petri nets, timed automata, and process algebra [5]. Actually this is natural, because such formal techniques can be analyzed and verified mathematically.

The study on composite event detection was first conducted in active database field. In [23], a mechanism suitable to model the semantics of composite events and the implementation of the event detector in active databases is described. Composite events are described in an event specification language based on petri nets. Input places in the petri net are used to model component events and output places are used to model composite events. Also some auxiliary places are used to model the abstract events such as timeouts. Colored petri nets are used so that the event parameters such as event type, time, location can be carried throughout the system. Every time the occurrence of an event is signaled, the input place of the petri net which models the appropriate event pattern is marked with a token. Then playing the token game, new markings are obtained. If a token is placed in an end place which has no transitions the composite event is said to be detected.

In [35], an active database model based on fuzzy ECA rules with a fuzzy petri net representation to process the information at a central location is proposed. They argue that processing information at a central location helps to have a global view and, and as a result, yields much more sound conclusions. So all the sensor readings are processed at that central location. The fuzzy petri net is constructed using the fuzzy ECA rules of the application domain. When sensor readings or queries arrive at the inference engine, fuzzy inference engine makes deductions using the rules and the data stored in fuzzy active database.

Cougar [63], TinyDB [37] and Directed Diffusion [30] are the popular data processing schemes that employ in-network processing in sensor networks. Cougar and TinyDB views the sensor network as a large distributed database and instead of collecting data at the sink, queries are distributed into the network. Query processing systems are demand-driven in nature, so they are not suitable for event-driven applications. In Directed Diffusion, interests, which are attribute-value pairs, are placed in the network by the sink and the nodes send their data to it if the interest is satisfied. Data is aggregated along the way back to the sink. Similar to previous approaches, it is demand-driven and not suitable for applications that require continuous monitoring. Furthermore, attribute-value pairs are not always expressive enough to describe

what is interesting. Therefore, unnecessary packets might still be transported.

In [48], a composite event processing framework that works on top of a range of publish / subscribe systems is proposed. Distributed composite event detectors are installed at various locations in the sensor network according to the requirements of the application such as latency, reliability or bandwidth usage. Although this work is similar to our study in the way that composite event expressions are decomposed into sub-expressions as we do in rule-base decomposition process, their proposition is not specifically about the wireless sensor networks and therefore, they do not take the constraints present in the wireless sensor networks into account. Furthermore they do not discuss about how the decomposition is done. On the other hand, we give an algorithmic way of decomposition that suits to the requirements of wireless sensor networks. Additionally, we consider a hierarchical network architecture, and we describe precisely how we can classify the nodes based on what they can process, and therefore where the information processing engines can be placed.

In [31], complex events with temporal and spatial constraints and correlations are detected by employing a hierarchical approach. An event description language based on petri nets, named SNEDL, is used to specify the events. Sensor readings and messages are modeled using tokens. Places represent the states in which the objects can be, arcs indicate the communication path and transitions represent a sensor nodeÕs processing of the sensor readings or messages. Events are classified as mote-level, group-level, and base-level events. At mote-level, individual readings from sensor nodes are processed and the outputs of this processing are fed into the inputs of the next level. Similarly, the result of group-level processing is used by the base-level. It is the base-level where the final decision about a composite event that the application is interested in is made. Although a hierarchy of event detectors are described in this paper, it does not discuss about how and based on what criterion these event detectors are created. On the other hand, we show how we classify different hierarchical levels from information processing perspective.

In [47], a rule-based distributed fuzzy logic reasoning engine is described. The reasoning engine employs simple if-then rules for the decision process and it uses fuzzy logic to fuse the individual sensor readings and the neighbor observations to get more accurate results. Although if-then rules are used for information processing, the main motivation of the study in [47] is to improve reliability of the decisions and it mentions only about processing done

at a single node. However, our method is about distributed processing in the entire wireless sensor network, and in our approach sensor nodes cooperate for the purpose of reducing network traffic and energy consumption.

In [1], a proactive and distributed mechanism is proposed to detect the sets of interrelated events, also called contexts. Event notifications are delivered to special nodes which are connected through an overlay network. These nodes make partial context decisions and forward their decision to the next node in the overlay. However, the approach adopted in [1] is to express the logical relations as disjunctions of conjunctions of premises whereas our approach is to express them as conjunctions of disjunctions of premises. Furthermore, in [1] nodes need to keep the address of the sensor nodes that are responsible for processing the next input element. As we use a hierarchical sensor network architecture, sensor nodes in our approach only need to know how they can reach the nearest node in the next hierarchical level. This simplifies the routing strategy.

In [32], a fuzzy rule-based system is demonstrated for event detection in WSNs. They show that using fuzzy logic improves decision accuracy. To tackle with the problem of the exponentially growing size of fuzzy rule-bases, they combine rules with the same consequent, similar to what we do in our rule reduction scheme. But contrary to our scheme, they generate new predicates that express the same events as the non-combined rules would catch. But our approach does not require changing the definitions of events.

CHAPTER 3

RULE DECOMPOSITION FOR DISTRIBUTED RULE PROCESSING

There are obvious benefits of centralized data-processing. For example a global view of data may bring more accurate results and only the processing resources of a central node need to be considered. Nevertheless, as we have discussed before, this paradigm is generally not suitable for WSNs and therefore data need to be processed in a distributed manner.

This chapter contains our first effort to distribute information processing into WSN. In the following section, we first discuss about rules that we use as the tool expressing application logic. Then, we present a mechanism that shows how we determine the roles in a sensor network setup. After that, we introduce our rule decomposition algorithm. The input to the algorithm is a rule from the central rule-base and the output is a set of sub-rules for different roles that in effect do the same thing as the original rule while it is possible to process them in a distributed manner. Thereafter, we give the proof that shows the functional equivalence of a central rule and its decomposed sub-rules, and then talk about the complexity analysis of the decomposition algorithm. Finally, we present how rule engines that are deployed in the sensor nodes work.

3.1 Event-Condition-Action Rules

In our approach, application logic is expressed by a set of ECA rules that represent a set of statements. The execution of these statements depends on the occurrence of specific event patterns and the satisfaction of the constraints on the events that constitute those event patterns. The statements express the actions to be taken and the conclusions to be drawn.

Employing ECA rules instead of embedding the logic into application code has several advantages [68]:

i. Rules can be stored outside an application in a rule-base, which improves the modularity, maintainability and extensibility of the application.

ii. Rules have a high-level declarative syntax, so they can easily be analyzed and optimized.

iii. Rules provide a generic mechanism to express reactive behavior, contrary to the application code, which is typically specialized to a particular type of reactive scenario.

In a procedural paradigm, how an application reacts to events should be coded by the application developers. However, by employing a declarative, rule-based paradigm, domain experts may directly determine the type of reaction. Therefore, application developers only need to develop a general purpose rule engine that processes rules. Such separation of duties is favorable, because in this way each person does what he is most proficient at.

ECA rules are in the form of:

```
IF              <Event Pattern>
PROVIDED THAT   <Set of constraints>
THEN            <Set of Actions>
```

The first part of a rule contains the event pattern that needs to be detected in order for that rule to fire. However, the detection of the event pattern does not guarantee the triggering of the action statements in the third part of the rule. The context in which the event pattern has been detected is of profound importance for intelligently determining the implications of the detected events. The conditions and constraints used for evaluating the context are listed in the second part of the rule.

ECA rules have been mostly used in active database field so far. In these systems, the event that triggers a rule is a single event like insertion of an entry into a table, completion of a transaction or similar database operations. What is placed in the condition part is for checking whether a specific situation has occurred as a result of the detected event. For example, after an insertion operation, the situation that the number of entries in a table exceeds some specific

value might be the condition for an ECA rule. Upon detection of such a condition, the rule may trigger the actions, e.g., purging some of the entries.

However, ECA rules in our study has slightly different usage compared to ECA rules in active databases. The main difference is in how the first and second parts of rules are used. First of all, a complex event pattern fires a rule. All the logical composition is done in the first part. The second section of the rule lists the conditions and constraints of the component events that describe the context.

3.1.1 Events

The IF-part, also called the antecedent, of a rule contains one or more events, i.e., a set of events. Without detecting the event pattern in the antecedent, it is not possible to execute statements in the action section, also called the consequent. We define an event as any change in the state of the data being monitored where the change has significance for the application. The arrival of a network packet or a value exceeding some threshold might be typical examples of WSN events.

A single event on its own is not generally enough for drawing conclusions or taking actions. In order to capture a real and complicated application scenario, we need to use more complex event patterns, formed by the logical composition of simple events using the operators of an event algebra.

3.1.1.1 Event Composition Operators

There are four core composition operators used in event algebras that have been introduced in the literature so far [18]:

- **Conjunction.** Conjunction of two events a and b, $(a \cdot b)$, specifies that both the event a and the event b need to occur and at the time of composition they must be valid, i.e., their time window should not have been expired. Apart from the constraint about the validity interval, there are no other temporal constraints.
- **Sequence.** Sequence of two events a and b, $(a\,;\,b)$, specifies that both of the events a and b need to occur. However, contrary to conjunction operator, the order of events is

important and the event a has to occur before the event b. Both events need to be valid during composition.

- **Disjunction.** Disjunction of two events a and b, $(a \mid b)$, specifies that either the event a or the event b should occur. Only temporal constraint is that the selected event that makes the evaluation true need to be valid, i.e., the time of composition lies within its validity interval.

- **Negation.** Negation of an event a, $(!a)$, specifies that the event a has not been occurred for a specific amount of time before the composition takes place.

In addition to these core operators, there are other operators that have been proposed like concurrency operator which requires two events to occur in parallel or iteration/counting operator which specifies a specific number of occurrences of some event, and so on. Most of these extra operators are the results of the effort that tries to combine the temporal and logical relationships of the events. The research that has been conducted in the event processing have placed great attention to temporal aspects of the processing. However, event processing in WSNs brings another important aspect into consideration: spatial constraints.

Although spatial information or source of an event might have little or no usage in a centralized active database system, it has an important part of the event processing in WSNs. Similar to the temporal aspects, spatial information are used to determine the context in which the event pattern has been detected. Based on these facts, we argue that separating the context evaluation from the logical composition of the events is a better approach. Logical event composition operators should be free of the context information which should be evaluated somewhere else. Otherwise, we have to redefine the old operators and/or add new operators into the event algebra for covering different circumstances.

Let's consider the following case: Assume that a sensor network is spatially divided into several regions. Our constraint is that two events need to occur but they have to be produced in adjacent regions. If we merge the semantics of spatial constraints with logical operators, then we have to define a new operator *adjacent* that detects the occurrences of these events and verifies that they are in the adjacent regions.

In addition to the cases that temporal and spatial constraints are considered individually, in some circumstances, the temporal and spatial constraints need to be considered together. For

example, *"the event A in region R1 need to occur before the event B in region R2"*. It is obvious that we shall not consider the semantics of logical composition, and temporal and spatial constraints in a single event operator, otherwise we have to deal with lots of event composition operators (a single logical operator would result in as many operators as the product of the number of temporal relations and spatial relations in the worst case).

In our study we take only three operators as the logical operators: *conjunction, disjunction* and *negation*. We do not use sequence operator, which is listed as a core operator above, as a logical composition operator, since sequence can be defined as a conjunction with a before relation in between the component events.

3.1.1.2 Event Detection and Composition

We use the term event for both the event definition and the event instance. We may draw an analogy between classes and objects in object-oriented paradigm with event definitions and event instances in event-driven systems. Event definitions in a rule are represented by predicates.

Definition 3.1.1 *A **predicate** is a binary function that takes variable number of arguments and returns a value of true or false.*

A true evaluation of a predicate means that the event has occurred; i.e., the event instance has been captured.

In order to evaluate a predicate, the inputs that are assigned to the parameters of the predicate need to be present where the processing takes place; e.g., the predicate $Less\_Than(t, -5)$ returns true if the input that t stands for is available and it is less than -5.

Some of the predicates might be evaluated by using a single operation, like comparing a parameter with a constant. On the other hand, some of them may require more than a single operation in their evaluation. For example, a predicate that checks whether the last five readings of a sensory data is above some threshold value cannot be evaluated by a single operation. In the presence of such a predicate, evaluating it over and over again for every rule that use it is a waste of energy and processing resources. If the state does not alter, that is the inputs remain unchanged, it would be wise to evaluate predicates only once and use the

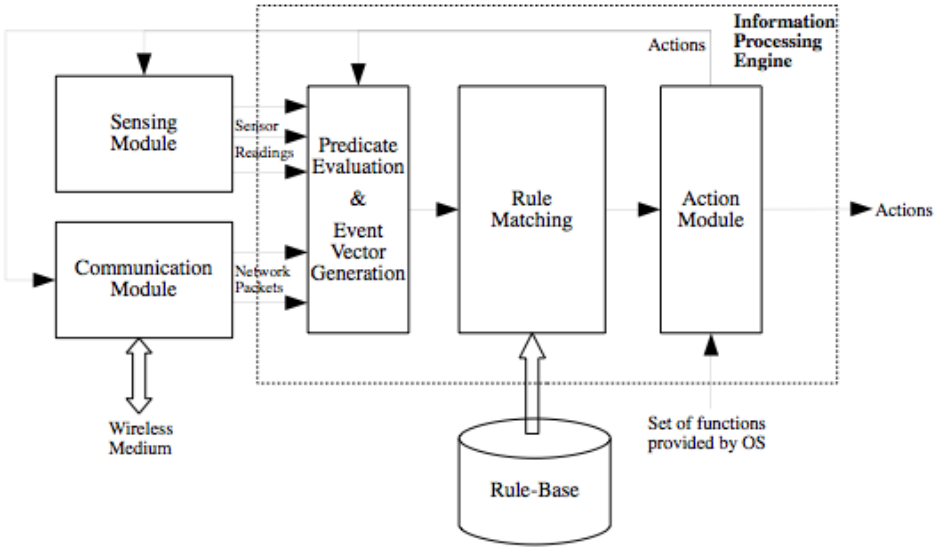


Figure 3.1: Rule-based information processing in a node

outcomes of the evaluations while processing rules. In a sensor network, environment is generally sensed periodically which means that the sensor readings will remain unchanged until the next sampling interval begins.

One of the important goals of information processing in wireless sensor networks is to devise lightweight methods for the processing of data. In order to achieve an efficient evaluation of the predicates we separate the predicate evaluation and the rule processing components in the event processing system. The predicate evaluation component is responsible for determining the events that can be used in the event composition for the detection of the event patterns that fire the rules. Rule processing component, also called the rule engine, is responsible for detecting the complex event patterns, and upon detection, determining whether the available context allows triggering of the actions. Figure 3.1 shows typical components of an event processing system and their interaction with each other in a sensor node.

For any node in the sensor network, there is a limited set of events which are used to describe the event pattern that forms the antecedent of the rules employed by the node being in concern. If each of these events is assigned a specific ID, then it is possible to use these IDs in the definition of the rules instead of the actual predicates.

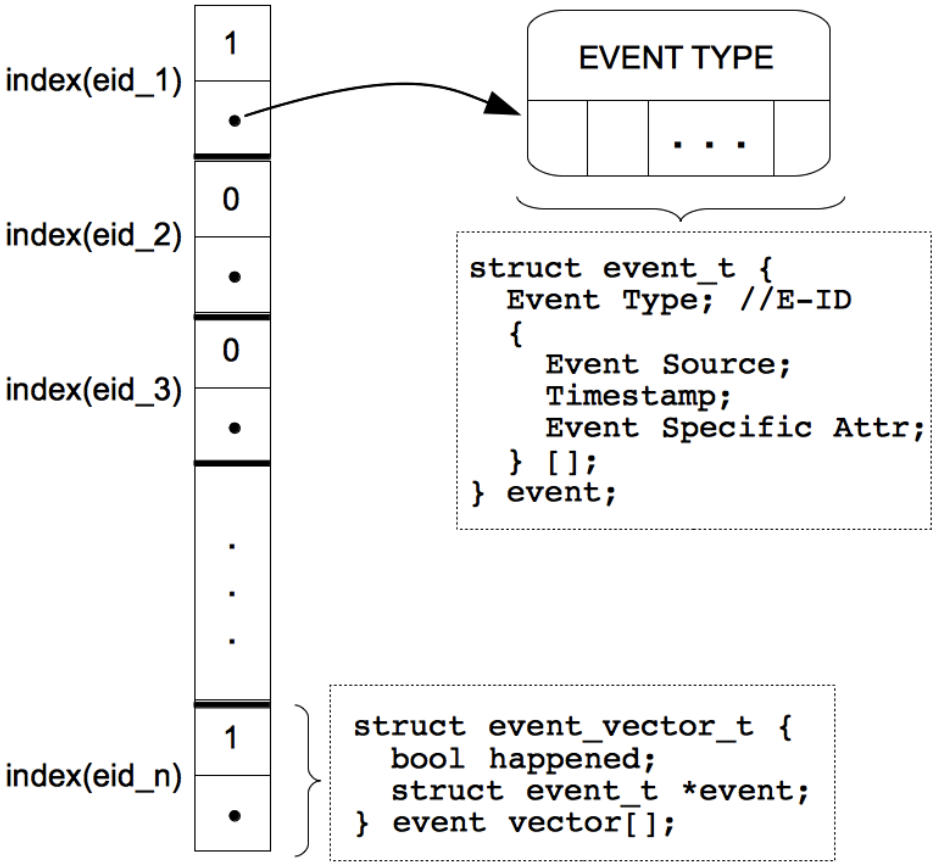


Figure 3.2: Event Vector

Definition 3.1.2 *An **event-ID** is a unique identifier that is assigned to an event type.*

Using event-IDs instead of the actual predicates are for the purpose of eliminating redundant predicate evaluations and speeding up the rule matching process. Once predicates are evaluated and events are determined, this knowledge is used in the rest of the execution of a rule engine. The processing carried out in the predicate evaluation module is made available to the rule engine in the form of an event vector.

Definition 3.1.3 *An **event vector** is a vector that lists all possible events that can be consumed by a rule engine, together with the information whether they have been occurred or not. Each entry of the vector takes a value of either '1', meaning the event has occurred, or '0', meaning the event has not been occurred.*

There is a one-to-one correspondence between event-IDs and indexes of the vector. When an event is detected, the field pointed by that event's ID in the event vector is set to 1. A value of 0 means that the event has not been occurred and 1 means the event has just happened. By looking at the event vector at a particular time, it is possible to determine which rules could fire at that instant.

Assignment of the event-IDs is done after rule-base decomposition. An event-ID has a local scope. In other words, re-use of the event-IDs is possible and an event-ID may point to two different events in two different types of nodes.

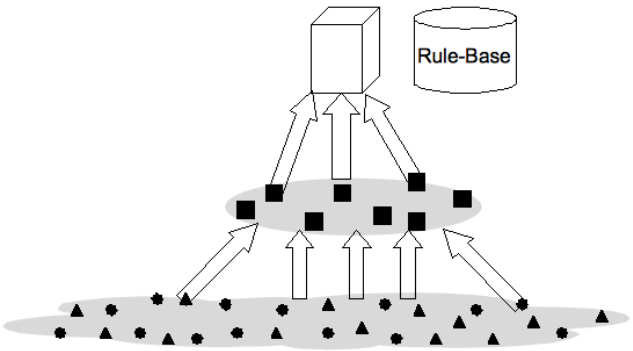
3.1.2 Conditions

An event pattern which describes the logical composition of the events cannot be solely used for triggering reactions. Events might have different meanings in different contexts. For example, "high pulse rate" has different thresholds for adults and children, or if an adult is doing exercise, the event "high pulse rate" does not signify an abnormal condition. Another example might be that we may not correlate a visual stimuli to an audial stimuli if the former happens after the latter. For example we cannot hear the sound of an explosion before seeing the explosion itself.

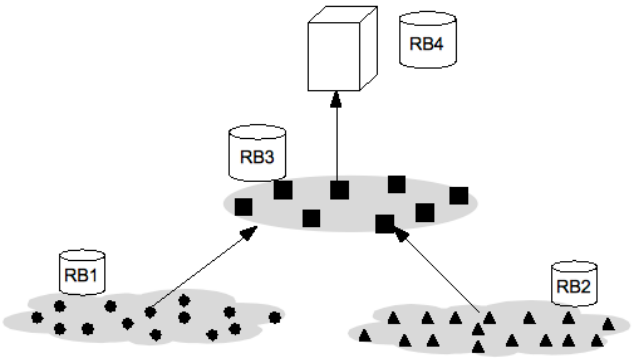
In our approach, context information is placed in the second section of a rule. This section consists of a set of constraints and conditions defined as relations between the component events placed in the antecedent of the rule, the IF-part. In order for the context evaluation not to fail, all constraints of and relations between the component events, which are used for the composition of the complex event pattern at run-time, need to satisfy.

3.1.3 Actions

The final part of a rule resembles the interrupt handlers implemented in operating systems. Interrupts raised due to an hardware or software exception cause the operating system to execute the statements in the corresponding interrupt handlers. While a rule is processed, after the detection of the complex event pattern and the satisfaction of all of the constraints and conditions, the action part of the rule is triggered, and the list of statements placed in this part is executed.



(a) Centralized



(b) Distributed

Figure 3.3: Rule-based information processing in wireless sensor networks

Action part of the rule contains a subset of the functions provided by the node that the rule is running on. For instance, an action for a typical sensor node might be sending a notification message over the network, or changing a parameter of the node itself like sampling rate or sleeping duration, or storing a value in the persistent memory.

3.2 Node Classification and Roles

The sensor nodes in a WSN may be classified into various types due to heterogeneity in the hardware or the logical roles that the sensor nodes possess [8, 19, 20]. We can think of several logical roles which is related to the network and the information processing architecture employed. If we consider the network architecture, a node may take the role of source which initiates the network traffic, a relay which simply forwards other nodes' packets, a sink which is the ultimate destination, and so on and so forth. In terms of information processing, a node might be assigned a data disseminating source, an aggregator, a data fusion node, or similar roles.

Due to the data-centric nature of sensor networks, networking and data management aspects cannot be thought of separately. An information processing role shall be assigned to a node that has another role related with networking. For example, for a cluster based sensor network, it is natural to assign a cluster-head the role of both the relay and the aggregator.

In a sensor network, physical capabilities and responsibilities of nodes might differ. Two nodes may sense different stimuli or one of the nodes may contain more powerful processor and enhanced battery resource compared to the other nodes. For example, for an early forest fire detection application, one type of the sensor nodes might be sensing temperature, humidity, atmospheric pressure and similar weather related stimuli while another type of sensor nodes may be using optical sensors to detect flame or smoke.

Hardware heterogeneity explicitly indicates that nodes with differing hardware cannot assume the same roles. However, even in homogeneous sensor networks in which all sensor nodes are of the same type in terms of hardware, different roles shall be used. Obviously, roles in such networks are assigned to the sensor nodes according to what logical responsibilities they assume during the operation of the network. This type of networks generally require the role assignment to be dynamic. In other words, the role of a node should be able to change over the course of operation of the network, because some of the roles may put too much burden on the sensor nodes' computational and energy resources [10, 16, 34, 61]. Using adaptive strategies that help in sharing out the price of such roles among all of the nodes of a WSN is a common technique used to prolong the WSN lifetime.

In the context of our study, a role emphasizes what a node can process. There might be various data types involved in data processing. For example, sensory data such as temperature, pressure and acceleration or data that is generated as a result of aggregation or fusion of other data, are data types that might be present in a WSN. Let T be the set of all different data types appearing in a WSN and $P(T)$ denote the power set of T. Then the set of roles can be defined as $R = \{r \mid \exists r \in P(T)\}$ with the cardinality m such that:

1. $T = \bigcup_{i \to 1}^{m} (r_i)$
2. Given $r_i \in R$, $r_j \in R$, $1 \leq i \leq m$, $1 \leq j \leq m$, and $i \neq j$, one of the following needs to be satisfied:
   (a) r_i and r_j are mutually disjoint,

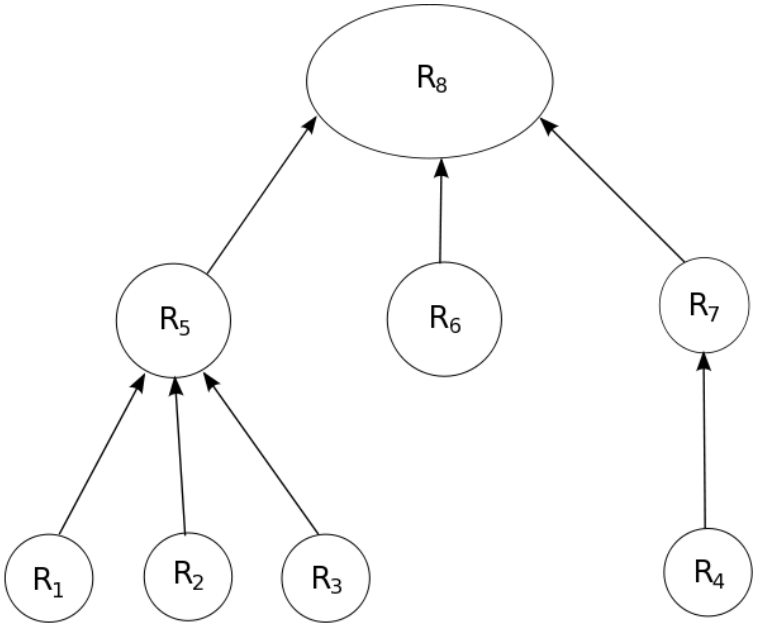


Figure 3.4: Hierarchy of roles

(b) $r_i \subset r_j$ or

(c) $r_i \supset r_j$

The first condition denotes that every data type is assigned to a role such that it will be processed by one type of node in the network. The second condition states that two roles do not intersect unless one of them is the proper subset of the other. 2.a ensures that a physical phenomenon is sensed by only one type of node. This condition makes sure that our assumption that "there is no redundancy in sensor hardware" holds true. In other words, a physical phenomenon is sensed by only one type of node. We think that this is a fair assumption since the presence of redundant sensing hardware complicates the task of designing appropriate algorithms for WSN applications. If some data, say temperature, is sensed by more than one type of node in a WSN application, then the flow of the process of the application might change by which node the temperature is sensed. It will be easier to deal with a WSN having non-redundant sensing hardware.

In a hierarchical sensor network, we assume that nodes that are in the higher levels have the ability to see the data of the lower levels, because sensor nodes send their data to the nodes on the upper levels. Conditions 2.b and 2.c represent this assumption. If one of the roles is a proper subset of the other, this implies that the node employing the subset role is in a lower

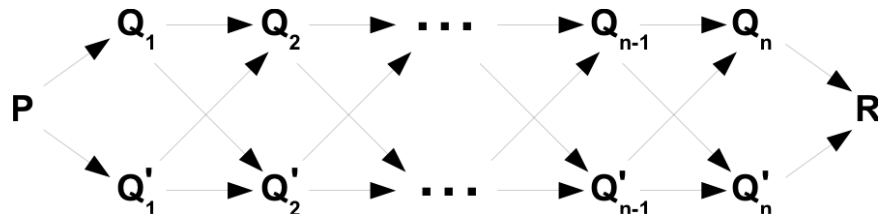


Figure 3.5: Directed graph showing possible execution paths

level.

Classification of the nodes is the process of determining every possible r_i that encloses the set of data types and pertains to the above constraints and conditions. Roles are determined in a bottom-up fashion: starting from the available data types, each data type is assigned to one of the appropriate roles.

In figure 3.4, a hierarchy of roles is shown. From that figure we can conclude the following:

- R_1, R_2, R_3 and R_4 are disjoint.
- R_5, R_6 and R_7 are disjoint.
- R_1, R_2 and R_3 are the proper subsets of R_5
- R_4 is a proper subset of R_7.
- R_5, R_6 and R_7 are the proper subsets of R_8

Roles are precomputed, and assignment of them to actual nodes might be static or dynamic. However, role assignment is not the subject of this study. We use roles as inputs in our rule decomposition algorithm, which is described in the following section.

3.3 Proposed Rule Decomposition Algorithm

3.3.1 Terms, Definitions and Notations

The antecedent of a rule is in the form of conjunctions and/or disjunctions of the predicates, which are atomic boolean expressions. However, our algorithm cannot decompose a rule with

an arbitrary composition of the predicates in its antecedent. It requires the antecedent to be in a special form: conjunctive normal form.

Definition 3.3.1 *A **disjunctive clause** is the disjunction of predicates:*

$$(P_1 \mid P_2 \mid \cdots \mid P_i)$$

Definition 3.3.2 *Antecedent of a rule is said to be in **conjunctive normal form** if it contains only conjunctions of disjunctive clauses:*

$$(P_{11} \mid P_{12} \mid \cdots \mid P_{1k}) \;\&\; \cdots \;\&\; (P_{i1} \mid P_{i2} \mid \cdots \mid P_{in})$$

Definition 3.3.3 *A disjunctive clause is said to be satisfied if at least one of the predicates, which form the clause, evaluates to true.*

Definition 3.3.4 *A rule is said to be fireable if all of the conjuncts, disjunctive clauses, in its antecedent satisfy.*

For any node in a sensor network, the predicates of a rule, which is supposed to be executed in that node, can be classified into two: the predicates that might be processed by the node, and the predicates that might not. For example, let's consider a temperature measuring sensor node. A predicate that uses the temperature information can be evaluated by this node, while a predicate whose evaluation depends on pressure information obviously cannot.

Definition 3.3.5 *The input set of a node is the set of predicates that the node can evaluate and determine their truth values.*

Using the knowledge of what a node can evaluate, a disjunctive clause can be categorized into one of the following:

i. **Fully processable disjunctive clause**: All predicates can be evaluated by the node.

ii. **Non-processable disjunctive clause**: None of the predicates can be evaluated by the node.

iii. **Semi-processable disjunctive clause**: There are predicates both that can be evaluated and that cannot be evaluated by the node.

Example. If a node's input set is $\{P_1,\ P_2,\ P_3\}$, i.e., it can evaluate predicates P_1, P_2 and P_3, $(P_1 \mid P_2)$ is a fully processable disjunctive clause, $(P_4 \mid P_5 \mid P_6)$ is a non-processable disjunctive clause, and $(P_1 \mid P_3 \mid P_4)$ is a semi-processable disjunctive clause for that node.

It is possible to put any expression that is not in conjunctive normal form into an equivalent expression in conjunctive normal form. Therefore, considering a node and its input set, the antecedent of a rule can be represented in the following generalized format:

$$P \ \& \ (Q_1 \mid Q_1') \ \& \ (Q_2 \mid Q_2') \wedge \cdots \ \& \ (Q_n \mid Q_n') \ \& \ R$$

where P is the conjunction of fully processable disjunctive clauses, R is the conjunction of non-processable disjunctive clauses, and each $(Q_i \mid Q_i')$, $1 \leq i \leq n$, is a semi-processable disjunctive clause. Note that the expression Q_i' is not the negation or inverse of Q_i. Each Q_i is a disjunction of the predicates that can be evaluated by the node, whereas, each Q_i' is a disjunction of the predicates that cannot be evaluated.

3.3.2 Decomposition Algorithm: RBDA

The rule-base decomposition algorithm that we propose, RBDA, first creates the sub-rule-base for the type of nodes that are classified at lowest hierarchy. Unlike the other types of nodes, such nodes' input set does not have any subset which is the input set of other types of nodes. If such a subset existed, then the these nodes would not be on the lowest level; higher layers have more inputs, which include inputs from lower layers.

Next, the rule-base will be created for the nodes with the property such that subsets of those nodes' input set can only be the input set of a node from a lower level, not from the higher levels. This iterative process is repeated until a sub-rule-base is created for each different type of node. The data is processed in the lowest possible level and it is not relayed into the upper layers. This bottom-up approach in creating the sub-rule-bases supports the idea that "data should be processed as close to its source as possible".

The rule-base decomposition algorithm takes the central rule-base and the input set of the type of node that we are intending to create a sub-rule-base for as the inputs. Let q be the set

Algorithm 1 Rule-Base Decomposition Algorithm 1: RBDA

INPUT: IS =input set, RB=original rule-base

OUTPUT: RB=original rule-base (modified), NRB=new sub-rule-base

for all *Rule* in RB **do**
  Set P, R, q, q'
  $\mathbb{P} \leftarrow powerset(q)$
  for all M in $\mathbb{P}$ **do**
    if M equals to q **then**
      if (R is NULL & (P is not NULL | q is not empty)) **then**
        Add $[(P \ \& \ conjunction\_of\_elements\_of\_M) \rightarrow O]$ into SRB
      else if q is empty **then**
        if (P is NULL) **then**
          Add $[R \rightarrow O]$ into RB
        else
          Add $[P \rightarrow O_k]$ into NRB
          Add $[(O_k \ \& \ R) \rightarrow O]$ into RB
        end if
      else
        Add $[(P \ \& \ conjunction\_of\_elements\_of\_M) \rightarrow O_k]$ into SRB
        Add $[(O_k \ \& \ R) \rightarrow O]$ into RB
      end if
    else
      Add $[(P \ \& \ conjunction\_of\_elements\_of\_M) \rightarrow O_k]$ into SRB
      $M' \leftarrow \{Q'_i \mid Q_i \notin M\}$
      Add $[(O_k \ \& \ conjunction\_of\_elements\_of\_M' \ \& \ R) \rightarrow O]$ into RB
    end if
  end for
  Remove *Rule* from RB
end for

$\{Q_1\ ,\ Q_2\ ,\ \cdots\ ,\ Q_n\}$ and q' be the set $\{Q'_1\ ,\ Q'_2\ ,\ \cdots\ ,\ Q'_n\}$, which are constructed using the input set of the node. Furthermore, let $\mathbb{P}$ be the powerset of q. For a member M of the powerset $\mathbb{P}$, if M and q are not identical, then a rule having the following antecedent will be added into the sub-rule-base:

$$P\ \&\ \prod_{i=1}^{m} M(i),$$

where m is the cardinality of M and $M(i)$ is the i^{th} element of M. The output part of the rule will be an auxiliary output O_k representing the current matching conditions of the original rule. If the Q part of a disjunctive clause evaluates to false, it is still possible that overall disjunctive clause holds true as the Q' part of the clause might result in a true evaluation. For this reason, when an auxiliary output is generated, i.e., it is not possible to decide whether the conditions for the original rule hold true or not, this auxiliary output should be forwarded to the upper layers in the hierarchy so that the inputs available there can be used to further validate the conditions.

If we define the set M' as $M' = \{Q'_i \mid Q_i\ \notin\ M\}$ with the cardinality $(n - m)$, then a new rule with the original output O and the following antecedent is added into the central rule-base:

$$O_k\ \&\ \left(\prod_{i=1}^{n-m} M'(i)\right)\ \&\ R,$$

where $M'(i)$ is the i^{th} element of M'.

If M and q are identical, the original rule is removed from the central rule-base. If R is missing in addition to that condition, the formula for adding a new rule into the newly generated rule-base does not differ from the previous case except when the rule will have the original output O rather than an auxiliary output as its output part.

The above process should be repeated for each member of the powerset $\mathbb{P}$ so that all possible combinations of condition matchings are enumerated. As a result of this process, a central rule is decomposed into multiple sub-rules and placed in two rule-bases: the newly created sub-rule-base and the modified central rule-base.

The above steps are for the decomposition of a single rule. In order to generate the complete sub-rule-base, the operations taken for just one rule should be repeated for every rule in the original rule-base. Furthermore, the described process only creates a sub-rule-base for one type of node and the complete distribution of information processing into the sensor network requires the creation of sub-rule-bases for every type of node residing in different hierarchies.

The following examples are given to illustrate the RBDA. For the sake of simplicity, the condition part of the rules are omitted.

Example 1. Let's consider the following rule:

$$(a_1 \ \& \ (b_1 \mid c_1) \ \& \ (d_1 \mid e_2 \mid f_2) \ \& \ (g_1 \mid h_2) \ \& \ i_2) \rightarrow O$$

where subscripts represent the hierarchical level where the related input can be processed. This rule is going to be distributed into 2 different rule-bases; i.e., only one iteration of the algorithm is enough to derive the necessary rules. The input set for the first hierarchical level is $IS = \{a_1, b_1, c_1, d_1, g_1\}$. Using this, we come up with:

- $P = (a_1 \ \& \ (b_1 \mid c_1))$
- $q = \{d_1, g_1\}$
- $q' = \{(e_2 \mid f_2), h_2\}$
- $R = i_2$

The powerset $\mathbb{P}$ is equal to $\{\{\}, \{d_1\}, \{g_1\}, \{d_1, g_1\}\}$. If we follow the steps described in the algorithm, the following rules would be added into the new sub-rule-base:

$$(a_1 \ \& \ (b_1 \mid c_1)) \rightarrow X_1$$
$$(a_1 \ \& \ (b_1 \mid c_1) \ \& \ d_1) \rightarrow X_2$$
$$(a_1 \ \& \ (b_1 \mid c_1) \ \& \ g_1) \rightarrow X_3$$
$$(a_1 \ \& \ (b_1 \mid c_1) \ \& \ d_1 \ \& \ g_1) \rightarrow X_4$$

If none, one or both of d_1 and g_1 are detected, this information is sent to the upper hierarchical layer for further processing provided that $(a_1 \ \& \ (b_1 \mid c_1))$ is detected. The original rule being decomposed is removed from the original rule-base and the following rules are added:

$$(X_1 \ \& \ (e_2 \mid f_2) \ \& \ h_2 \ \& \ i_2) \rightarrow O$$
$$(X_2 \ \& \ h_2 \ \& \ i_2) \rightarrow O$$
$$(X_3 \ \& \ (e_2 \mid f_2) \ \& \ i_2) \rightarrow O$$
$$(X_4 \ \& \ i_2) \rightarrow O$$

Example 2. Let's now consider another rule for a three level hierarchy:

$$(a_1 \ \& \ (b_1 \mid c_2 \mid d_3) \ \& \ (e_1 \mid f_3)) \rightarrow O$$

where subscripts represent the hierarchical level where the related input can be processed. In this example we are going to distribute this rule into 3 different rule-bases. In the first iteration of the algorithm, $P = a_1$, $q = \{b_1, e_1\}$, $q' = \{(c_2 \mid d_3), f_3\}$ and R is null. The powerset $\mathbb{P}$ is equal to $\{\{\}, \{b_1\}, \{e_1\}, \{b_1, e_1\}\}$. If we follow the steps described in the algorithm, the following rules would be added into the new sub-rule-base:

$$\begin{aligned}(a_1) &\rightarrow X_1\\ (a_1 \ \& \ b_1) &\rightarrow X_2\\ (a_1 \ \& \ e_1) &\rightarrow X_3\\ (a_1 \ \& \ b_1 \ \& \ e_1) &\rightarrow O\end{aligned}$$

If all of a_1, b_1 and e_1 evaluate to true, then the rule engine reaches a conclusion and the associated actions are taken. On the other hand, if at least one of b_1 or e_1 yields a false value or they cannot be evaluated because of the absence of the input, then the available information is fused and the result is sent to the node in the next hierarchical level. Furthermore, auxiliary outputs are added as inputs into the input set of the next hierarchical level. After the first iteration, original rule-base is going to have the following rules:

$$\begin{aligned}(X_1 \ \& \ (c_2 \mid d_3) \ \& \ f_3) &\rightarrow O\\ (X_2 \ \& \ f_3) &\rightarrow O\\ (X_3 \ \& \ (c_2 \mid d_3)) &\rightarrow O\end{aligned}$$

In the second iteration of the algorithm, sub-rule-bases for the second and third hierarchical levels are generated by decomposing those 3 rules above. For the first rule, $P = X_1$, $q = \{c_2\}$, $q' = \{d_3\}$ and $R = f_3$. For the second rule, $P = X_2$, $R = f_3$, q and q' are empty. Finally for the third rule, $P = X_3$, $q = \{c_2\}$, $q' = \{d_3\}$ and R is null. Sub-rules

that are generated for the second hierarchical level are as follows:

$$(X_1) \rightarrow Y_1$$

$$(X_1 \ \& \ c_2) \rightarrow Y_2$$

$$(X_2) \rightarrow Y_3$$

$$(X_3) \rightarrow Y_4$$

$$(X_3 \ \& \ c_2) \rightarrow O$$

After the second iteration, the original rule-base which is the rule-base of the third hierarchical level is going to have the following rules:

$$(Y_1 \ \& \ d_3 \ \& \ f_3) \rightarrow O$$

$$(Y_2 \ \& \ f_3) \rightarrow O$$

$$(Y_3 \ \& \ f_3) \rightarrow O$$

$$(Y_4 \ \& \ d_3) \rightarrow O$$

3.4 Equivalence of Initial Rule and Decomposed Sub-rules

Before we proceed with the proof of the RBDA, let's give the common facts, terms and notations used in this section.

By a single operation of the algorithm, a rule is decomposed into sub-rules, which are then to be processed by two different types of nodes. The node whose input set is used to determine P, Q, Q' and R is referred to as the first node, and the other as the second node. The antecedent of the original rule might contain P, $(Q_i \mid Q'_i)$, R or any combination of them. There are three cases that this could possibly fit:

- There is no $(Q \mid Q')$ in the rule.
  - **Case 1.** There is either P or R. If there is P, $r : P$ is placed in the first node's rule-base, and if R, $r : R$ is placed the second node's rule-base.
  - **Case 2.** There are both P and R. A single pair of rules is generated: $r_1 : P$, which is placed in the first node's rule-base, and $r_2 : r_1 \ \& \ R$, which is placed in the second node's rule-base.

- **Case 3.** There are $(Q \mid Q')$ and possibly P and/or R. If there are n conjuncts in the form of $(Q \mid Q')$, there can be $n + 1$ cases that we need to consider: 0 to n Qs being detected by the first node. For each of n+1 cases, there are $C(n, i)$ different combinations of Qs where $0 \leq i \leq n$. Therefore, there can be a total of $\sum_{i=0}^{n} C(n, i) = 2^n$ different cases that can be encountered during run-time processing. The RBDA exhaustively generates a distinct pair of rules for every possible case. Let q be the set $\{Q_1, Q_2, \cdots, Q_n\}$, q' be the set $\{Q'_1, Q'_2, \cdots, Q'_n\}$, s be any subset of q and s' be the subset of q' such that $(Q'_i \in s') \Leftrightarrow (Q_i \notin s)$. The antecedents of a pair of rules r_1 and r_2 generated by the RBDA are as follows:

  - r_1 : $(P \ \& \ <conjunction\ of\ elements\ of\ s>)$, which is placed in the first node's rule-base and
  - r_2 : $(<output\ of\ r_1> \ \& \ <conjunction\ of\ elements\ of\ s'> \ \& \ R)$, which is placed in the second node's rule-base.

  If, however, there is no R and the case considered is for n Qs, only a single rule is added into the first node's rule-base: r : $(P \ \& \ <conjunction\ of\ elements\ of\ q>)$. For any pair of r_1 and r_2, the union of the conjuncts of r_1 and r_2 covers all the conjuncts of the original rule, since s together with s' cover all conjuncts that are in the form $(Q \mid Q')$ of the original rule.

Lemma 1: An event pattern captured by a rule is also captured by the sub-rules, which are decomposed from that rule.

Proof: We need to prove that if original rule satisfies, r or r_2 also satisfies. Consider the three cases one by one:

1. *P or R*: As the original rule and the rule placed in the sub-rule-base are the same, the same events are detected.
2. *P and R*: Satisfaction of the original rule means that P and R evaluate to true. If P satisfies, r_1 also satisfies. Similarly, the satisfaction of r_1 and R results in the satisfaction of r_2. Therefore, if the original rule satisfies, sub-rules r_1 and r_2 also satisfy.
3. *Existence of* $(Q \mid Q')$: Satisfaction of the original rule implies that P, R and each $(Q \mid Q')$ evaluate to true. Since an exhaustive list of possible cases enumerated by

sub-rules, at least one rule pair out of 2^n pairs will catch the event pattern detected by the original rule.

Lemma 2: An event pattern that cannot be captured by a rule cannot be captured by the sub-rules generated as a result of the decomposition of that rule.

Proof: We need to prove that if the original rule does not satisfy, neither does r or r_2. Consider the three cases one by one:

1. *P or R*: As the original rule and the rule placed in the sub-rule-base are the same, the same event pattern should yield the same result.
2. *P and R*: If the original rule does not satisfy, either P or R should yield a false evaluation. If P were false, r_1 and, as a result, r_2 would not satisfy. If R were false, r_2 would not satisfy. Therefore, if the original rule does not satisfy, neither does sub-rule r_2.
3. *Existence of* $(Q \mid Q')$: If the original rule does not satisfy, there must be at least one conjunct that evaluates to false. Since the conjuncts of the original rule are covered by all pairs of sub-rules, each pair of sub-rules should have at least one conjunct that does not satisfy, which means that if the original rule does not satisfy, sub-rules do not satisfy either. The same reasoning applies to the case in which there is only r, which covers all conjuncts of the original rule.

Theorem 1: For a two-layer hierarchy, sub-rules that are decomposed from a rule capture exactly the same events as the original rule.

Proof: In order for this to be true, the following conditions should hold:

- an event pattern should be captured by sub-rules if and only if it could be captured by the original rule,
- an event pattern should not be captured by sub-rules if and only if it could not be captured by the original rule.

Let $A(e)$ be a proposition expressing an event pattern e detected by the original rule, $B(e)$ be a proposition expressing e detected by the sub-rules of that rule, $A'(e)$ be the proposition expressing e not detected by the original rule, and $B'(e)$ be the proposition expressing e not detected by the sub-rules. We need to prove that $(A(e) \Leftrightarrow B(e))$ and

$(A'(e) \Leftrightarrow B'(e))$. In order to prove $(A(e) \Leftrightarrow B(e))$, we need to show $(A(e) \Rightarrow B(e))$ and $(B(e) \Rightarrow A(e))$. Similarly, in order to prove $(A'(e) \Leftrightarrow B'(e))$, we need to show $(A'(e) \Rightarrow B'(e))$ and $(B'(e) \Rightarrow A'(e))$. However, we know from propositional logic that $(A(e) \Rightarrow B(e)) \vdash (B'(e) \Rightarrow A'(e))$ and $(A'(e) \Rightarrow B'(e)) \vdash (B(e) \Rightarrow A(e))$. Therefore, it is enough to show that $(A(e) \Rightarrow B(e))$ and $(A'(e) \Rightarrow B'(e))$. The proofs for these are given in lemmas 1 and 2.

Theorem 2: For a k-layer hierarchy, sub-rules that are decomposed from a rule capture exactly the same events as the original rule.

Proof: (Proof by Induction) A single operation of the decomposition algorithm generates two sub-rule-bases: a new rule-base for the node whose input set is used by the algorithm, and a modified original rule-base (MORB).

- *Initial Step:* For two-layer hierarchy, a single running of the algorithm generates two sub-rule-bases. The sub-rules of a decomposed rule capture exactly the same events as the original rule, as proved by Theorem 1.

- *Inductive Step:* For a k-layer hierarchy, the decomposition algorithm should be run $(k-1)$ times. We have to prove that for an arbitrary $k \geq 3$, if the theorem holds for $(k-1)$ sub-rule-bases, it also holds for k sub-rule-bases. Let's assume that the algorithm is run $(k-2)$ times, which generates $(k-1)$ rule-bases, and the events detected by the original rule are exactly the same as the events detected by sub-rules distributed into $(k-1)$ rule-bases. In order to generate the final rule-bases for $(k-1)^{th}$ and k^{th} layers, the decomposition algorithm should be run once more, taking the following inputs: the input set of the $(k-1)^{th}$ layer and the MORB, generated from the $(k-2)^{th}$ iteration of the algorithm. Theorem 1 proves that sub-rules that are decomposed from a rule in MORB and placed in the newly generated rule-bases capture exactly the same event patterns as the event patterns detected by that decomposed rule. Based on this, we can conclude that sub-rules of an initial rule placed in k rule-bases capture exactly the same events as that rule does, since the event patterns captured by the rules of first $(k-2)$ together with the patterns captured by MORB constitute the event pattern of the initial rule.

3.5 Upper Bound on the Number of Sub-rules

The main factors affecting the running time of the decomposition algorithm and the number of sub-rules generated are the number of different node classifications, the number of rules in the original rule-base and the cardinality of the set q of each rule.

Let k be the number of different classifications, r be the number of rules in the original rule-base, n_i be the total number of sub-rules that are generated from the i^{th} rule, n_i^k be the number of sub-rules for the k^{th} classification that are generated from the i^{th} rule and q_i^k be the cardinality of the set q of the i^{th} rule for the k^{th} classification.

For a typical sensor network, we expect the number of different classifications at different hierarchies to have a small value. For example, in a cluster-based homogeneous sensor network this number will most probably be 3. The cardinality of the set q changes with each rule and each classification. Similar to the number of classifications, we anticipate a small value for it on the average case.

The worst case in decomposing rules for a classification k occurs when there are P and R parts in each sub-rule generated for classification $k+1$. In such a case, the number of rules that are generated from one rule is as follows:

$$\begin{aligned} n_i^1 &= 2^{q_i^1} \\ n_i^2 &= n_i^1 * 2^{q_i^2} \\ &= 2^{q_i^1} * 2^{q_i^2} \\ &\vdots \\ n_i^k &= n_i^k * 2^{q_i^k} \\ &= 2^{q_i^1} * 2^{q_i^2} * \cdots * 2^{q_i^k} \end{aligned}$$

$$n_i^k = \prod_{j=1}^{k} 2^{q_i^j} \tag{3.1}$$

Using these, we can calculate the upper bound on the total number of sub-rules generated for the i^{th} rule as follows:

$$n_i \leq \sum_{t=1}^{k} \left(\prod_{j=1}^{t} 2^{q_i^j} \right) = \sum_{t=1}^{k} 2^{\left(\sum_{j=1}^{t} q_i^j\right)} \tag{3.2}$$

Therefore, the upper bound on total number of sub-rules are:

$$n = \sum_{i=1}^{r} n_i \tag{3.3}$$

$$n \leq \sum_{i=1}^{r} \left(\sum_{t=1}^{k} 2^{\left(\sum_{j=1}^{t} q_i^j\right)} \right) \tag{3.4}$$

Total number of sub-rules generated has exponential space complexity. Since rule-base decomposition is statically done outside of the sensor network, algorithm running time has little importance in terms of energy efficiency of the sensor network. However, number of rules influence both space and rule-processing time of sensor nodes. More rules being placed in a rule-base means that more time and computation required for processing them.

3.6 Rule Processing

Given a set of realized events, rule engine is responsible for finding the appropriate rules that match the event patterns in their antecedents and satisfy the constraints in their condition parts. There might be multiple rules that could fire with the set of available events. In such a case, the selection of the rules that should fire depends on the requirements of applications. The requirements may differ from an application to another application. Therefore, instead of handling the rule selection problem in a generic way, it is much more appropriate to give the rule-base developers the ability to control which rules could fire during run-time. However, the necessary tools need to be provided to them. For example, an action that blocks processing the rest of the rules in a rule-base, or the ability to block the execution of a specific action statement, should be provided in the platforms that perform rule processing.

There are several ways that a rule engine could accomplish the task of processing rules, and choosing the appropriate mechanisms depend on the concerned application. The following are the possible strategies that may be used:

- The rule engine may only fire a single rule. The problem is that which rule should be selected. The rule engine might employ the following strategies:
  - It may select the first rule that could fire.
  - It may select a rule randomly.

Algorithm 2 A Single Iteration of Processing by Rule Engine

```
1:  RB = Rule-Base
2:  minPriority = The smallest priority that of a rule which could fire
3:  numRules = Number of rules that can fire
4:  stopProcessing ← 0
5:  n ← 0,  g ← 0
6:  for all Rule in RB do
7:      if stopProcessing then
8:          break;
9:      end if
10:     if (g == group-id(Rule)) then
11:         continue;
12:     end if
13:     p ← priority(Rule);
14:     if (p < minPriority) then
15:         continue;
16:     end if
17:     if ((numRules != 0) & (n > numRules)) then
18:         break;
19:     end if
20:     if Rule matches event pattern then
21:         if Rule satisfy constraints then
22:             Trigger Actions
23:             n++
24:             g ← group-id(Rule)
25:         end if
26:     end if
27: end for
```

  - If rules are assigned priorities, it may select the highest priority rule that could fire.

- The rule engine may fire multiple rules and all of the associated actions are performed. In such a case, starting from the top of the rule-base and searching for the rules that could fire, the rule engine has the problem of when to stop processing.

  - All rules that could fire may be processed. In such a case, the rule engine need to go on with the processing until it reaches the end of the rule-base.
  - Rule processing may be stopped by an explicit statement in the action part of a fired rule which bans the processing of the rest of the rules.
  - Only a specific number of rules may fire. For example, an application might require at most five rules to be fired in a specific time interval.
  - If assigned priorities, rules with a priority that is above some value may fire. If there are rules that could fire but they have priorities below the threshold value, different strategies may be employed. For example, a strategy may be to never execute them, and another one may be to fire them if there are not any rules fired before.

- The rule engine may fire multiple rules, but some of the actions may need to be performed only once. For example, in a healthcare monitoring application, if multiple rules can fire, and those rules contain ”Call 911” action, running this action more than once is not desirable, while other actions may be performed more than once.

There are some rules that should never fire together. All the rules of a rule-base that are decomposed from the same rule form a group. During the execution, only one of them should be able to fire. In order to assure that, each rule in a rule-base should be assigned a rule group-id, and only one of them should be executed at a particular moment. The sub-rules that are decomposed from a rule all have the same rule group-id as the rule that is decomposed.

If multiple rules could fire in a rule group, there is only one way to select a rule: In a group of rules, the most specific rule that satisfy the conditions should fire. A rule is more specific if the number of elements of the set q being a part of the antecedent is more compared to another rule. For example, $(P \ \& \ Q_1 \ \& \ Q_2)$ is more specific than P, or $(P \ \& \ Q_1)$, or $(P \ \& \ Q_2)$.

Algorithm 2 provides a mechanism that can be employed by rule engines. This algorithm assumes rules have been assigned priorities.

CHAPTER 4

RULE DECOMPOSITION: REVISITED

The main drawback of the algorithm given in the previous chapter, RBDA, is that all possible n-ary combinations of the elements of the set q need to be enumerated. In such a case, the number of sub-rules generated grows exponentially depending on the number of elements in q. This situation results in two major problems. First, more rules means that more space is required for storage, yet storage is a luxury for a typical sensor node. Second, a larger rule-base leads to increased processing overhead as the information-processing engine needs to go over much more rules for the event detection and correlation operations.

In order to reduce the number of sub-rules generated, we extend our algorithm with a new one that employs rules with variables, helping us to get rid of the need for statically listing all possible input combinations of a rule.

4.1 Variables

We defined the term predicate as any binary function that takes a variable number of arguments and returns a value of true or false. Furthermore, given a node's input set, we presented a generalized format of the antecedent of the rules:

$$P \;\&\; (Q_1 \mid Q'_1) \;\&\; (Q_2 \mid Q'_2) \;\&\; \cdots \;\&\; (Q_n \mid Q'_n) \;\&\; R,$$

where P, R, Q_i and Q'_i have the same meanings as described in the previous section.

Definition 4.1.1 *Let q be the set $\{Q_i \mid 1 \leq i \leq n\}$ and q' be the set $\{Q'_i \mid 1 \leq i \leq n\}$. A variable is a symbol that represents some m-ary combination of the elements of the sets q or q', where $0 \leq m \leq n$.*

In the previous algorithm, for decomposing a rule which contains elements of q in it, the cases that embody the presence of from none to n Q's need to be enumerated. However, use of the variables eliminates the requirement for this enumeration.

Let's consider, for example, the following antecedents: P, $(P\ \&\ Q_1)$, $(P\ \&\ Q_2)$, $(P\ \&\ Q_1\ \&\ Q_2)$. We can represent these in a single expression with the help of the variable: $(P\ \&\ \vartheta)$, where ϑ stands for either ϕ, Q_1, Q_2 or $(Q_1\ \&\ Q_2)$. By the help of variables, there is no need to exhaustively list all possible input combinations.

Rules used in the new algorithm differ from the conventional rules in that additional array structures are associated with the variables. Array entries correspond to the elements of the sets q or q', or to the results of their evaluations. However, note that a variable may only be used for either the elements of set q or q', but not both. In other words, all entries of an array need to be related to the elements of the same set.

There are two roles that a variable can assume. The first role is to statically store the information about the disjunctive clauses to be matched, which are the elements of the sets q or q'. The second role is to keep and transfer dynamic knowledge about the results of the evaluation of the disjunctive clauses during rule processing. For this purpose, we define two types of variables:

1. *Predicate Matching Variable (PMV):* This is responsible for statically storing all possible disjunctive clauses that could be matched at run-time. The rule engine uses PMV to determine what to search for in the available event stream at run-time.

2. *Knowledge Transfer Variable (KTV):* This holds the results of the processing carried out by a node, and is used by the nodes at the upper layers to determine what should be matched. As KTV stores the results of evaluations, its contents are meaningful only at run-time. The antecedent of a rule cannot contain a KTV alone; it needs to be accompanied by a PMV. Actually, there is one-to-one relationship between the entries of the PMV and the KTV. The KTV contains the results of the evaluations of the elements of the set q from previous layers and the PMV stores the elements of the set q'. Variables in the consequent can only be KTV.

Sub-rules that are produced as a result of the decomposition algorithm might contain zero, one or two variables in their antecedents.

- If there are no variables in a rule, the rule that is decomposed to produce that rule has an empty set q.

- If there is a single variable in a rule, it stores all elements of the set q. The variable is a PMV and this rule cannot be further decomposed. In the presence of such a rule, there must be an accompanying rule employed in the next (upper) layer. Rules generated for the nodes of the lowest layer cannot have more than one variable.

- If there are two variables in a rule, the first variable, which is a KTV, holds the knowledge about the processing carried out in the lower layer. The second variable is a PMV and it stores the elements of the set q' that is constructed using the input set of the lower layer. Besides keeping the relationship with the lower layer, the second variable also stores the elements of the set q, which is constructed for the current layer, such that processing in the current layer can be used to shape the processing carried out in the upper layers.

4.2 Decomposition Using Rules with Variables: RBDA-V

Similar to our previous algorithm, given an input set for a node, the new algorithm, RBDA-V, decomposes a rule-base into two rule-bases: a new rule-base for the node, *NRB*, and the modified original rule-base, *ORB*.

Algorithm 3 gives a pseudocode implementation of the new rule-base decomposition process. It begins by putting the antecedent of the original rule into the conjunctive normal form. Each conjunct is either a single predicate or a disjunction of predicates. Conjuncts form P if all predicates can be evaluated by the node and conjuncts that cannot be evaluated form the R part of the antecedent. If the conjunct is in disjunctive normal form and only some part of it can be evaluated by the node, then the part that can be processed constitutes Q and the remaining part constitutes Q'. The antecedent of a rule can be written as P & $(Q_1 \mid Q'_1)$ & $\cdots$ & $(Q_n \mid Q'_n)$ & R.

Let q be the set $\{Q_1,\ Q_2,\ \cdots,\ Q_n\}$ and q' be the set $\{Q'_1,\ Q'_2,\ \cdots,\ Q'_n\}$. If the rule to be decomposed only contains P, it is added into the *NRB*, and if it only contains R, added into the *ORB* unaltered. If the rule contains both P and R but not any $(Q \mid Q')$, a sub-rule with P as

Algorithm 3 Decomposition Algorithm Using Rules with Variables: RBDA-V - Part 1

INPUT: IS =input set, RB=original rule-base

OUTPUT: RB=original rule-base (modified), NRB=new sub-rule-base

1: **for all** *Rule* in RB **do**
2: Set P, R, q, q' ;
3: $n \leftarrow$ cardinality of set q
4: **if** *Rule* does not contain variables **then**
5: **if** ((R is NULL) & (P is not NULL | $n \geq 1$)) **then**
6: Add $[(P \ \& \ conjunction\_of\_all\_Q_i) \rightarrow O]$ into NRB
7: **else if** q is empty **then**
8: **if** (P is NULL) **then**
9: Add $[R \rightarrow O]$ into RB
10: **else**
11: Add $[P \rightarrow O_p]$ into NRB
12: Add $[(O_p \ \& \ R) \rightarrow O]$ into RB
13: **end if**
14: **else**
15: **for** $i = 1$ to n **do**
16: $\vartheta[i] = Q_i$
17: $\omega[i] = Q'_i$
18: **end for**
19: Add $[(P \ \& \ \vartheta) \rightarrow \mu]$ into NRB
20: Add $[(\mu \ \& \ \omega \ \& \ R) \rightarrow O]$ into RB
21: **end if**
22: **else** ▷ // $\Big(P \ \& \ \mu \ \& \ \omega \ \& \ (Q_1|Q'_1) \ \& \ \cdots \ \& \ (Q_n|Q'_n) \ \& \ R\Big)$
23: $\vartheta \leftarrow remove\_non\_processable\_disjuncts(\omega)$
24: $\omega \leftarrow remove\_processable\_disjuncts(\omega)$
25: **if** (R is NULL) **then**
26: Add $[(P \ \& \ \mu \ \& \ \vartheta \ \& \ conjunction\_of\_all\_Q_i) \rightarrow O]$ into NRB
27: **end if**

Algorithm 4 Decomposition Algorithm Using Rules with Variables: RBDA-V - Part 2

28: $k \leftarrow$ max index of ϑ
29: **for** $i = 1$ to n **do**
30: $\vartheta[k + i] = Q_i$
31: $\omega[k + i] = Q'_i$
32: **end for**
33: Add $[(P \ \& \ \mu \ \& \ \vartheta) \rightarrow \tau]$ into NRB
34: Add $[(\tau \ \& \ \omega \ \& \ R) \rightarrow O]$ into RB
35: **end if**
36: Remove *Rule* from RB
37: **end for**

its antecedent and an auxiliary output as its consequent is added into the NRB. The associated rule to be placed in the RB has the conjunction of auxiliary output and R as its antecedent, together with the original rule's consequent as its consequent.

Provided that the set q is not empty, let n be the number of elements of the set q. With our new algorithm, instead of generating 2^n pairs of rules, a rule can be decomposed into at most three sub-rules regardless of the value of n: one or two rules for the NRB and one rule for the ORB. Two sub-rules, for the NRB, are generated when there is no R in the original rule and it is possible to reach a state where the original rule's actions can be taken. One of the rules contains the original rule's consequent, which covers the case that all elements of the set q are captured. And the other rule, which has an auxiliary output, is used to cover the cases in which not all the elements of the set q are captured.

Let ϑ represent the variable used in the sub-rule to be placed in the NRB and $\vartheta[\,]$ represent the array associated with it. If the set q is non-empty and the rule does not contain any variables, then the elements of q are assigned to the elements of the array associated with the variable:

$$\vartheta[i] = Q_i, \quad 1 \leq i \leq n$$

If P is absent, only ϑ, otherwise P and ϑ constitute the antecedent of the sub-rule to be placed in the NRB. The rule has has an auxiliary output, μ. It is a KTV and used as the first variable in the accompanying rule to be placed in the ORB. This rule has a second variable, ω, which is a PMV and holds the elements of the set q':

$$\omega[i] = Q'_i, \quad 1 \leq i \leq n$$

μ, ω, and, if it exists, R comprise the antecedent of the rule to be placed in the *ORB*. Its consequent is equal to the original rule's consequent. In addition to this pair of rules, if there is no R, then an additional rule, having the original rule's output as its consequent and the conjunction of the elements of the set q and, if it exists, P as its antecedent is added into the *NRB*.

If the rule to be decomposed contains variables, the generalized format for the antecedent must look like the following:

$$\left(P \,\&\, \mu \,\&\, \omega \,\&\, (Q_1|Q_1') \,\&\, \cdots \,\&\, (Q_n|Q_n') \,\&\, R\right),$$

where μ, a KTV, corresponds to the entries of the set q, and ω, a PMV, holds the entries of the set q'. Note, however, that these sets are not the ones used for the current decomposition; they are created and used in the previous run of the RBDA-V. Let k be the size of the arrays associated with μ and ω. Let ϑ be assigned to ω with the non-processable predicates removed. Furthermore, let the processable predicates of ω be removed, leaving ω only with non-processable inputs. The array for ϑ will hold not only the elements coming from ω, but also the elements of the set q, which has the cardinality n:

$$\vartheta[k+i] = Q_i, \quad 1 \leq i \leq n$$

μ, ϑ and, if it exists, P constitute the antecedent of the sub-rule to be placed in the *NRB*. The rule has an auxiliary output, τ. It is a KTV and used as the first variable in the accompanying rule that is placed in the *ORB*. This rule uses ω as the second variable:

$$\omega[k+i] = Q_i', \quad 1 \leq i \leq n$$

τ, ω, and, if it exists, R comprise the antecedent of that rule. Its consequent is equal to the original rule's consequent. In addition to this pair of rules, if there is no R, then a rule with the original rule's output as its consequent and the conjunction of the elements of the set q, μ, ϑ and, if it exists, P as its antecedent is added into the *NRB*.

The last step is to remove the original rule from the *ORB*. Nevertheless, the steps described above decomposes only a single rule into sub-rules. These operations should be repeated for every rule in the *ORB* in order to generate the complete sub-rule-base for the type of node it is. The input set of this node is used in the RBDA-V. Furthermore, the described process only creates a sub-rule-base for one type of node; the complete distribution of information

processing into the WSN requires the creation of sub-rule-bases for each type of node residing at different layers.

Example 1. Let's consider the following rule:

$$(a_1 \ \& \ (b_1 \mid c_1) \ \& \ (d_1 \mid e_2 \mid f_2) \ \& \ (g_1 \mid h_2) \ \& \ i_2) \rightarrow O$$

where subscripts represent the hierarchical level where the related input can be processed. This rule is going to be distributed into 2 different rule-bases; i.e., only one iteration of the RBDA-V is enough to derive the necessary rules. The input set for the first hierarchical level is $IS = \{a_1, b_1, c_1, d_1, g_1\}$. Using this, we come up with:

- $P = (a_1 \ \& \ (b_1 \mid c_1))$
- $q = \{d_1, g_1\}$
- $q' = \{(e_2 \mid f_2), h_2\}$
- $R = i_2$

If we follow the steps described in the RBDA-V, the following rule would be added into the new sub-rule-base:

$$(a_1 \ \& \ (b_1 \mid c_1) \ \& \ \vartheta[d_1, g_1]) \rightarrow \mu[d_1, g_1]$$

The original rule being decomposed is removed from the original rule-base and instead the following rule is added:

$$(\mu[d_1, g_1] \ \& \ \omega[(e_2 \mid f_2), h_2] \ \& \ i_2) \rightarrow O$$

Example 2. Let's consider the following rule for a three-level hierarchy:

$$a_1 \ \& \ (b_1 \mid c_2 \mid d_3) \ \& \ (e_1 \mid f_3)) \rightarrow O,$$

where subscripts represent the level where the related input can be processed. In this example we are going to distribute this rule into three different rule-bases. In the first iteration of the RBDA-V, $P = a_1$, $q = \{b_1, e_1\}$, $q' = \{(c_2 \mid d_3), f_3\}$ and R is null. If we follow the steps described in the RBDA-V, the following rules would be added into the new sub-rule-base:

$$a_1 \ \& \ b_1 \ \& \ \ e_1 \rightarrow O$$

$$a_1 \ \& \ \vartheta[b_1, e_1] \rightarrow \mu[eval(b_1), eval(e_1)]$$

$eval(p)$ is the result of the evaluation of p. Elements of the KTVs are given for the sake of understandability. If all of a_1, b_1 and e_1 evaluate to true, then the rule engine reaches a conclusion and associated actions are taken. On the other hand, if at least one of b_1 or e_1 yields a false value or they cannot be evaluated because of the absence of input, then the available information is fused and the result is sent to the node in the next layer. After the first iteration, the original rule is removed from the *ORB* and the following rule is added into it:

$$\mu[eval(b_1), eval(e_1)] \ \& \ \omega[(c_2 \mid d_3), f_3] \rightarrow O$$

In the second iteration of the RBDA-V, sub-rule-bases for the second and third levels are generated by decomposing the above rule. The non-processable predicates in ω are d_3 and f_3. Based on this information, the sub-rule that is generated for the second level is as follows:

$$\mu[eval(b_1), eval(e_1)] \ \& \ \vartheta[c_2, -] \rightarrow \tau[eval(b_1) \mid eval(c_2), eval(e_1)]$$

After the second iteration, the original rule-base, which is the rule-base of the third level, have the following rule:

$$\tau[eval(b_1) \mid eval(c_2), eval(e_1)] \ \& \ \omega[d_3, f_3] \rightarrow O$$

4.3 Equivalence of Initial Rule and Decomposed Sub-rules

By a single operation of the RBDA-V, a rule is decomposed into sub-rules to be processed by two different types of nodes. Let's refer to the rule-base of the node where the input set is used to determine P, Q, Q' and R as the first rule-base, and the other one as the second rule-base.

The antecedent of the original rule might contain P, a KTV/PMV pair, $(Q_i \mid Q'_i)$, R or any combination of these. It might contain no or two variables in its antecedent: a KTV (corresponding to the elements of q) and a PMV (corresponding to the elements of q'). They are related such that if the consequent of a rule does not contain a variable (i.e., it is a decision rule) and the non-variable conjuncts have been satisfied, the following condition must hold in order for the rule to fire: $\forall i, (KTV[i] \mid PMV[i]) \rightarrow true$.

There are four cases that the original rule could possibly be in:

- There are neither variables nor $(Q \mid Q')$ in the rule.
  - **Case 1.** There is either P or R. If there is P, $r: P$ is placed in the first rule-base and if R, $r: R$ is placed the second rule-base.
  - **Case 2.** There are both P and R. A single pair of rules is generated: $r_1 : P$, which is placed in the first rule-base, and $r_2 : r_1 \ \& \ R$, which is placed in the second rule-base.
- **Case 3.** There are $(Q \mid Q')$ and possibly P and/or R in the rule, but no variables. Let ϑ and μ correspond to the elements of q, and ω corresponds to the elements of q'. A pair of rules, r_1, for the first rule-base, and r_2, for the second rule-base, are as follows: $r_1 : ([P \ \&] \ \vartheta) \rightarrow \mu$, $r_2 : (\mu \ \& \ \omega \ [\& \ R]) \rightarrow O$. If there is no R, an additional rule is added into the first rule-base: $r : ([P \ \&] < conj\_of\_elements\_of\_q >) \rightarrow O$. Actually, r_1 and r_2 could detect the same event patterns as r, but its sole purpose is to enable the first node to react to events.
- **Case 4.** There is a KTV/PMV pair and possibly $(Q \mid Q')$ and/or P and/or R in the rule. Let KTV be μ and PMV be θ for the original rule, ϑ and τ correspond to the elements of q and processable entries of θ, and ω correspond to the elements of q' and the non-processable elements of θ. The antecedents of a pair of rules r_1 and r_2 generated by the RBDA-V are as follows: $r_1 : ([P \ \&] \ \mu \ \& \ \vartheta) \rightarrow \tau$, $r_2 : (\tau \ \& \ \omega \ [\& \ R]) \rightarrow O$. If there is no R, an additional rule is added into the first rule-base:
  $r : ([P \ \&] \ \mu \ \& \ \vartheta \ [\& < conj\_of\_elements\_of\_q >]) \rightarrow O$. It is used for enabling the first node to react to events.

An event pattern detected by the original rule is also detected by sub-rules if r_2 or r fires. Because an event pattern that is detected by r is also detected by r_1, we will omit discussing it in the rest of the proof. Furthermore, as the proofs for cases 1 and 2 are similar to the ones given in Section 3, here we discuss only cases 3 and 4.

Lemma 3: An event pattern captured by the original rule is also captured by sub-rules, which are decomposed from that rule.

Proof: Let $E(Pr)$ represent the result of the evaluation of Pr. The firing of the original rule means that all conjuncts satisfy; in other words, $E(P)$, $E(R)$, $\forall i \ E(Q_i \mid Q'_i)$ and $\forall i \ E(\mu[i] \mid \theta[i])$ (only for case 4) are true.

Case 3. As $E(P)$ is true, r_1 fires with $\mu[i]$ set to true if $E(Q_i)$ is true, or else it is set to false. Since all $(Q_i \mid Q_i')$ evaluate to true, if $\mu[i]$ $(= E(Q_i))$ is not true, $\omega[i]$ $(= E(Q_i'))$ needs to satisfy. As a result, the condition that "$\forall i,\ (KTV[i] \mid PMV[i]) \rightarrow true$" holds and because $E(R)$ is true, r_2 satisfies.

Case 4. Let there be x conjuncts in the form of $(Q \mid Q')$ and the number of entries of μ be y. The first y elements of ϑ contain the disjunctions of the processable predicates of θ and the remaining x elements correspond to Q_i. Similarly, the first y elements of ω contain the disjunctions of the non-processable predicates of θ and the remaining x elements correspond to Q_i'. τ contain the results of the evaluations of the elements of ϑ. As $E(P)$ is true, r_1 fires, with $\tau[i]$ is set to true if $\mu[i]$ is true or $\vartheta[i]$ satisfies, or else it is set to false. As all $(Q_i \mid Q_i')$ evaluate to true, the disjunction of $\tau[i]$ and $\omega[i]$, where $(y+1) \leq i \leq (y+x)$, needs to produce true as well. Since $\theta[i] = \vartheta[i] \mid \omega[i]$ for $1 \leq i \leq x$, what is captured by θ in the original rule will also be captured by the τ and ω variable pair. As a result, the condition that "$\forall i,\ (KTV[i] \mid PMV[i]) \rightarrow true$" holds and because $E(R)$ is true, r_2 satisfies.

Lemma 4: An event pattern not captured by the original rule is not captured by the sub-rules, which are decomposed from that rule, either.

Proof: If the original rule does not satisfy, at least one of P, R, $(Q_i \mid Q_i')$ or $\theta[i]$ (only for case 4) should fail to satisfy. If $E(P)$ is false, r_1 and in turn, r_2 do not fire for cases 3 and 4. Similarly, if $E(R)$ is false, r_2 fails to satisfy. If $E(P)$ and $E(R)$ are true:

- *Case 3.* As the original rule fails to fire, the following condition should hold: $\exists i,\ E(Q_i \mid Q_i') \rightarrow false$. In such a case, although r_1 fires, r_2 fails to fire since the condition that "$\forall i$ $(KTV[i] \mid PMV[i])$ is true" does not satisfy.
- *Case 4.* If there exists at least one i for which $E(Q_i \mid Q_i')$ is false, r_2 does not fire for the reason explained for case 3. Otherwise, the following condition must be true: $\exists i, (\mu[i] \mid \theta[i]) \rightarrow false$. Since $\theta[i] = \vartheta[i] \mid \omega[i]$, what is captured by θ in the original rule will also be captured by the τ and ω variable pair in r_2. As a result, the condition that "$\forall i,\ (KTV[i] \mid PMV[i]) \rightarrow true$" would fail and therefore r_2 fails to satisfy.

As we can see, it is not possible to fire r_2 if the original rule fails to fire.

Theorem 3: For a two-layer hierarchy, sub-rules that are decomposed from a rule capture exactly the same events as the original rule.

Proof: In order for this to be true, the following conditions should hold:

- an event pattern should be captured by sub-rules if and only if it could be captured by the original rule,
- an event pattern should not be captured by sub-rules if and only if it could not be captured by the original rule.

The proofs given for lemmas 3 and 4 demonstrate that these conditions hold true.

Theorem 4: For a k-layer hierarchy, sub-rules that are decomposed from a rule capture exactly the same events as the original rule.

Proof: (Proof by Induction) A single operation of the RBDA-V generates two sub-rule-bases: new rule-base for the node whose input set is used by the algorithm, and the modified original rule-base.

- *Initial Step:* For 2-layer hierarchy, single running of the RBDA-V generates two sub-rule-bases. The sub-rules of a decomposed rule capture exactly the same events as the original rule, as it is proved by Theorem 1.
- *Inductive Step:* For a k-layer hierarchy, the decomposition algorithm should be run $(k-1)$ times. We have to prove that for an arbitrary $k \geq 3$, if the theorem holds for $(k-1)$ sub-rule-bases, it also holds for k sub-rule-bases. Let's assume that the RBDA-V is run $(k-2)$ times, which generates $(k-1)$ rule-bases, and the events detected by the original rule are exactly the same as the events detected by sub-rules distributed into $(k-1)$ rule-bases. In order to generate the final rule-bases for $(k-1)^{th}$ and k^{th} layers, the decomposition algorithm should be run once more taking the inputs: the input set of the $(k-1)^{th}$ layer and the modified original rule-base, MORB, generated from the $(k-2)^{th}$ iteration of the RBDA-V. Theorem 1 proves that sub-rules that are decomposed from a rule in MORB and placed in the newly generated rule-bases capture exactly the same event patterns as the event patterns detected by that decomposed rule. Based on this, we can conclude that sub-rules of an initial rule placed in k rule-bases capture exactly the same events as that rule does, since the event patterns captured by the rules of first

$(k-2)$ together with the patterns captured by MORB constitute the event pattern of the initial rule.

4.4 Upper Bound on the Number of Rules

The use of variables in rules helps us to transfer the burden of space into computation. The main factor that affect the number of sub-rules generated by the RBDA-V is the number of hierarchical layers. In any layer except the highest layer, there can be at most two sub-rules generated from one rule. In the highest hierarchical layer, there can be at most one rule.

Let k be the number of different hierarchical layers, and r be the number of rules in the original rule-base. In the worst case, the number of sub-rules generated from a single rule is going to be less than or equal to $(2k-1)$. Therefore the upper bound on the total number of rules that can be used by the application is $n(2k-1)$.

4.5 Rule Processing

The topics of the selection of rules that could fire, and when to stop the rule processing are discussed in Chapter 3. Conceptually, there is not any difference in the handling of these issues. On the other hand, in our new approach, rule engine has to perform an additional task: matching variables with the elements of the sets q and q', which are the sets used during the decomposition of the rule that this rule is decomposed from.

The minimum requirements for a rule to be eligible for firing is that non-variable elements in the antecedent need to be detected. If the consequent is an auxiliary output, then not all of the elements in the array associated with the variable need to satisfy. This auxiliary output is used as a variable in the rules of the next layer.

In a rule which contains two variables, the first variable stores information about events that have been detected by the lower layers thus far. The second variable matches the remaining events using the information in the first variable. The second variable may also be used to pass the result of the partial processing, which has been performed up until the current layer, onto higher layers.

Let's consider the disjunctive clause $(Q \mid Q')$. Q is stored in the first variable and Q' is stored in the second variable. The index of the array entry that holds Q is equal to index of the entry that holds Q'. If the i^{th} entry of the first variable evaluates to true, then the rule engine does not need to check the satisfaction of the i^{th} entry of the second variable. Furthermore, if the second variable is used to pass the information onto higher layers, the i^{th} entry is treated as if it has been satisfied. In other words, the i^{th} entry of the second variable of the rule in the next layer does not need to be checked.

Let's consider the rule from Example 2:

$$(a_1 \ \& \ (b_1 \mid c_2 \mid d_3) \ \& \ (e_1 \mid f_3))) \rightarrow O$$

Sub-rules generated are:

$$\begin{aligned}
Layer1: &\quad (a_1 \ \& \ b_1 \ \& \ \ e_1) \rightarrow O \\
Layer1: &\quad (a_1 \ \& \ \vartheta[b_1, e_1]) \rightarrow \mu[b_1, e_1] \\
Layer2: &\quad (\mu[b_1, e_1] \ \& \ \vartheta[c_2, *]) \rightarrow \tau[c_2, *] \\
Layer3: &\quad (\tau[c_2, *] \ \& \ \omega[d_3, f_3]) \rightarrow O
\end{aligned}$$

If a_1, b_1 and e_1 occur, then the rule in layer one fires and action O is taken. Now let's assume that events a_1, c_2 and f_3 have occurred. In the first layer, the second rule will fire even though ϑ does not match b_1 and e_1. Therefore, $\mu[false, \ false]$ is sent to the second layer. In the second layer, ϑ matches c_2 and therefore $\tau[true, \ false]$ is sent to the third layer. In the third layer, ω matches f_3 and the result of the evaluation becomes $\omega[true, \ true]$. As a result, the rule fires and action O is taken.

CHAPTER 5

IMPRECISION IN WSN

An indispensable requirement for data processing in WSNs is to deal with unreliable and imprecise sensory data. There are various reasons behind this important challenge. First, sensor nodes are deployed in environments that are open to the interference of some physical events or objects, not deployed in special protected areas. Second, as huge numbers of nodes are generally required for a typical WSN application, cheap fabrication techniques need to be used to produce affordable sensor nodes. Although this results in inexpensive nodes, this also leads to less reliable sensor nodes. Another problem is the unreliable wireless communication, which is the de facto communication mechanism in WSNs. In order to preserve the decision reliability even in the presence of unreliable data, inexact computation techniques need to be employed in WSNs.

In this chapter, after giving some preliminary information, we describe a fuzzy rule-based system for WSNs. Moreover, as the most important deficiency of fuzzy rule-based systems is the large numbers of rules, we propose a new method that reduces the number of rules in a fuzzy rule-base, and based on this, we propose an algorithm to be used by fuzzy inference engines for fuzzy compositions.

5.1 Dealing with Imprecision

There are several strategies that can be used to deal with imprecision in information processing [44]. Bayesian inference is an information fusion technique where probability theory is used to combine the information in order to find out whether a hypothesis is true or not. The uncertainty is represented by conditional probabilities which can be a value between zero and

one. A value of zero represents that a hypothesis is absolutely false and one represents that it is absolutely true. Bayesian inference uses Bayes' rule:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

where the a posteriori probability $P(A|B)$ represents the belief of hypothesis A given the information B. This probability is obtained by multiplying $P(A)$, the a priori probability of hypothesis A, by $P(B|A)$, the probability of receiving B given A is true. The probability $P(B)$ is considered as a normalizing constant.

The Dempster-Shafer inference is based on evidential reasoning [17, 54], which is a generalization of the bayesian theory. Instead of dealing with probabilities used in bayesian inference, it combines evidence from different sources to come up with a belief. This mechanism is suitable especially for multi-sensor data fusion [44]. The beauty of this mechanism is that the information to be fused might be in different granularities. Each sensor can provide information in different levels of details.

Another method for handling imprecision is fuzzy logic. It deals with uncertainty by mapping crisp values into linguistic variables, which broadly describe the value. The main advantage of fuzzy inference is that it transforms a complex problem into a more simpler problem which can be dealt with more easily.

Bayesian inference requires prior knowledge about the probabilities, $P(B|A)$ and $P(B)$, which is not practically possible to be obtained. Large amounts of data need to be collected, and statistical analysis techniques need to be used to generate those prior probabilities, costing time and expense. While Dempster-Shafer inference does not assign a priori probabilities, it requires complex operations to be performed which is beyond the computational abilities of a typical sensor node.

For the above reasons, we choose to use fuzzy logic in handling imprecision and uncertainty in WSNs. It is simple to be implemented on sensor nodes with low resources. The problems can be described much more easily because it is much more closer to the human way of thinking. People use fuzzy if-then rules all the time. For example, "If I go to bed before 11p.m., I can get up earlier" is a typical fuzzy rule.

5.2 Fuzzy Logic

In traditional two-valued logic, also called bivalent logic, a statement is either true or false, zero or one, right or wrong, in or out, valid or not valid. However, we are not living in a digital world and two values are not enough to model the the analog world that we are living in. The seminal paper of L. Zadeh in 1965 [65], which introduced the concept of fuzzy sets, underlies fuzzy logic.

There are generally imprecisions in inputs and/or outputs of people's decisions. For example, the statement "*the height of John is greater than 170cm*" is either true or false in traditional logic. However, if you ask a person that whether the height is greater than 170cm or not, he will probably give inexact answers, such as *maybe*, *probably* or *not sure*. Besides the inexactness of the resulting decision, the input used for the decision may not be precise. For example, even if a person does not know the precise value of John's height, he can still give the answer *YES* for the question "Is John's height greater than 170cm?". While giving their answers, people most probably tend to rely on their opinions, instincts, unreliable observations and other subjective judgements rather than measuring John's height to give a definite answer. Reasoning with imprecision is quite an effective strategy because relying on precise evaluations of precise inputs may not be possible all the time. Besides that, precision generally brings too much cost and complexity for the decision process, which makes precise computing infeasible for some applications.

Humans' way of reasoning is quite a successful mechanism as they are considered to be the most intelligent living being in the world. In the light of evolutionary theory, a thing has to be successful in order to live up. One of the most important aspects of this reasoning mechanism is that complexity is reduced by employing approximate reasoning. Of course there are cases in which precision is very important, but for most of the time approximate reasoning yields acceptable results.

Precision increases complexity, so reasoning mechanisms that use precise inputs and outputs require time and resource. Besides that, precision is not achievable every time. For example, it is inevitable that there will be noise in the analog signal captured by the sensor nodes that are deployed in a harsh physical environment. This noise will then cause deviations from the actual values in digitized sensor readings. Therefore, it is preferable to have a reasoning

mechanism that can tolerate this kind of inaccuracies.

Fuzzy logic helps to model the human way of reasoning. Literals, such as *tall*, *strong*, *big*, etc., are used to describe fuzzy sets.

5.2.1 Fuzzy Sets and Membership Functions

An ordinary set is based on the two-valued logic. An item is either a member of the set or not. Such a set A on a universe of discourse U can be represented by the following characteristic function:

$$\mu_A(x) = \begin{cases} 1 & x \in A, \\ 0 & x \notin A. \end{cases}$$

The characteristic function of the set A can also be represented as a functional mapping as follows:

$$\mu_A(x) : U \rightarrow \{0, 1\}$$

Although ordinary sets are simple and easy to operate on, they are not suitable to be used in most of the real world cases. In real life, i.e., the analog world that we are living in, it is hard to classify things as pure black or white. There are grays that represent partial truthfulness or falseness. Humans' interaction with their surroundings generally contains these types of partial information or conclusions. For example, let us consider the following statement: *John is tall*. Let us further assume that John's height is 182 cm. John is not very tall but he is slightly above the average height of the population, so he may be considered tall.

A fuzzy set is an extension of a crisp set that allows partial memberships [21]. The degree of membership of an element x of a fuzzy set A on a universe of discourse U is defined by the following membership function, which is an extension of the characteristic function for ordinary sets:

$$\mu_A(x) : U \rightarrow [0, 1],$$

where 0 means no membership and 1 means full membership. In contrast to the characteristic function of ordinary sets, $\mu_A(x)$ can take any real value in the range 0 to 1: $0 \leq \mu_A(x) \leq 1$. A degree of membership that is in between 0 and 1 models partial membership.

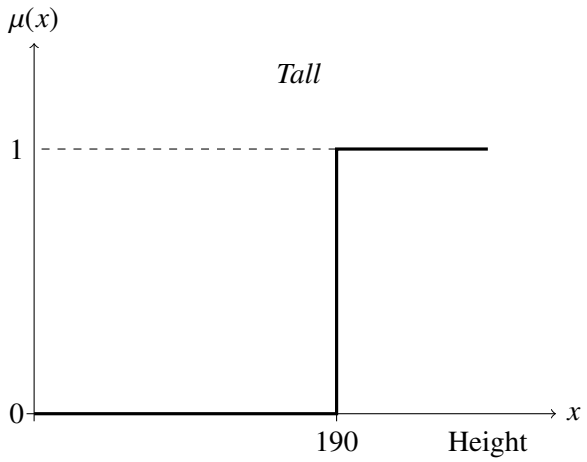


Figure 5.1: An example for the characteristic function of a crisp set

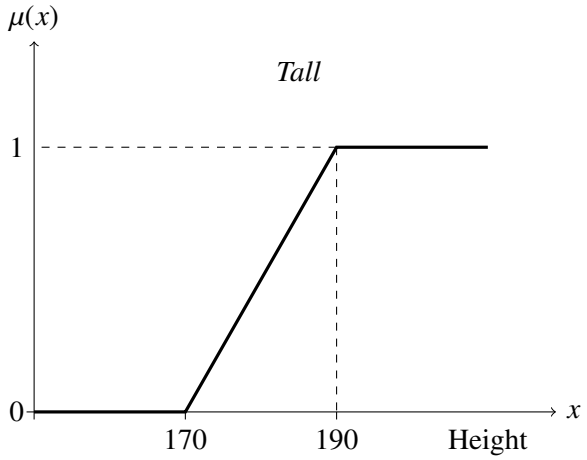


Figure 5.2: An example for the membership function of a fuzzy set

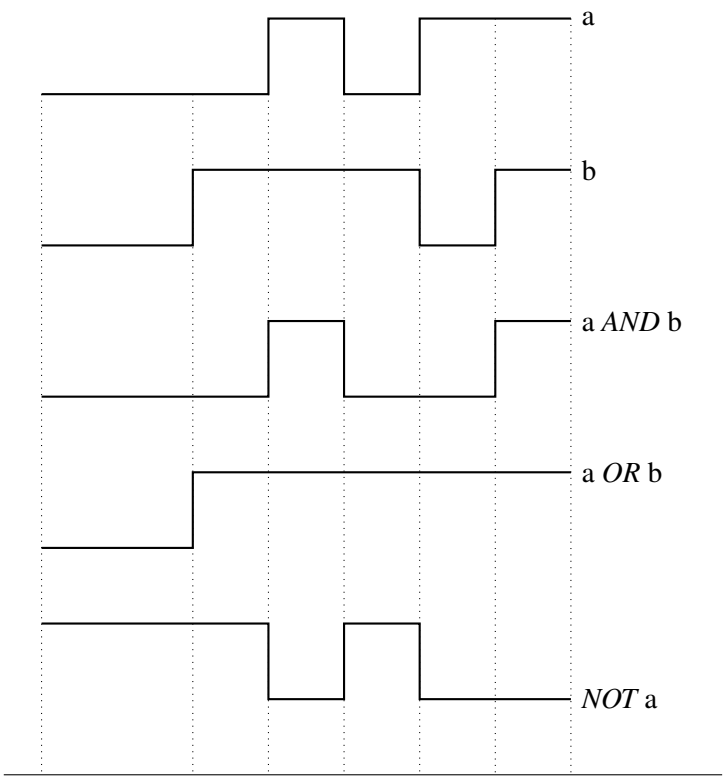


Figure 5.3: Conjunction, disjunction and complement operations in bivalent logic

5.2.2 Fuzzy Operators

Intersection and union are the basic operations of crisp sets that correspond to conjunction (AND) and disjunction (OR) operations in bivalent logic. These are also the basic operators in a fuzzy logic system. However, as expected, they have slightly different definitions from the ordinary ones [21].

5.2.2.1 Conjunction Operator

The truth table for the conjunction operator, also called AND operator, of the classical binary logic can be seen in Table 5.1. This operator corresponds to the intersection operation in the classical set theory. Instead of considering only zero and one, the binary conjunction can be extended on the unit interval [0, 1] as follows [56]:

Definition 5.2.1 *Binary conjunction on the unit interval is a function* $f : [0,1]\ x\ [0,1] \rightarrow [0,1]$ *which satisfies the following conditions* $\forall x, y, z \in [0,1]$*:*

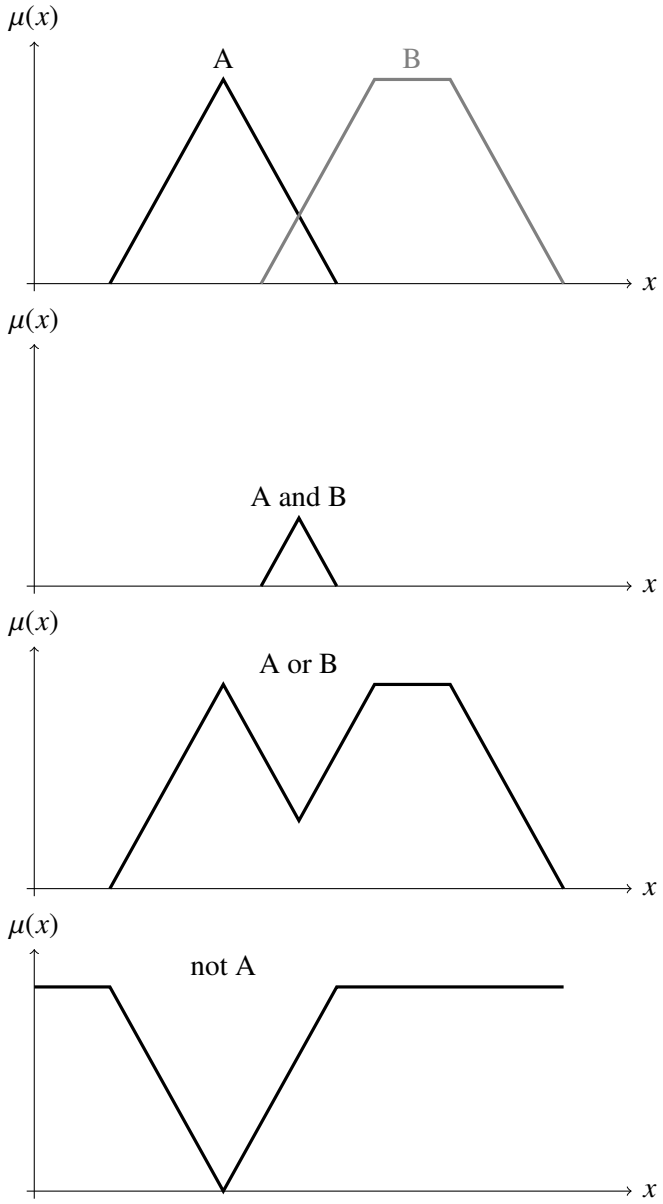


Figure 5.4: Conjunction, disjunction and complement operations in multi-valued logic

Table 5.1: The truth table of the binary conjunction

P_1	P_2	$P_1 \wedge P_2$
0	0	0
0	1	0
1	0	0
1	1	1

i. $f(x, 1) = x$, *and*

ii. Monotonicity: $(x \leq y) \Rightarrow (f(x, z) \leq f(y, z)\ \&\ f(z, x) \leq f(z, y))$.

The first condition makes sure that '1' is interpreted as an identity element. The monotonicity condition assures that an increase in the conjunct's value does not produce a decrease in the resulting truth value.

There is not a single implementation of the conjunction operator for fuzzy logic systems. In accordance with requirements, different correspondences between operands may be defined. The conjunction operation in fuzzy logic is typically modeled as a triangular norm, t-norm for short.

Definition 5.2.2 *A triangular norm is a function* $t : [0, 1]\ x\ [0, 1] \rightarrow [0, 1]$ *which satisfies the following conditions* $\forall x, y, z \in [0, 1]$*:*

i. Boundary: $t(x, 1) = x$ *and* $t(0, 0) = 0$,

ii. Monotonicity: $(x \leq y) \Rightarrow (t(x, z) \leq t(y, z))$,

iii. Commutativity: $t(x, y) = t(y, x)$,

iv. Associativity: $t(x, t(y, z)) = t(t(x, y), z)$.

The boundary condition ensures that the behavior of the conjunction operator for crisp sets is modeled correctly. The monotonicity condition states that an increase in x and y do not produce a decrease in the result. The commutativity condition guarantees that the order in which x and y are combined makes no difference. Finally, the associativity condition ensures that any number of sets can be combined in any order.

Every t-norm is a conjunction on the unit interval. Typical t-norms that are seen in the literature are as follows [25]:

- **Gödel/Minimum t-norm:** $T(x, y) = min(x, y)$.
- **Product t-norm:** $T(x, y) = xy$.
- **Łukasiewicz t-norm:** $T(x, y) = max(0, x + y - 1)$.

Table 5.2: The truth table of the binary disjunction

P_1	P_2	$P_1 \vee P_2$
0	0	0
0	1	1
1	0	1
1	1	1

- **Drastic product t-norm:** $T(x,y) = \begin{cases} min(x,y) & \text{if } x = 1 \text{ or } y = 1, \\ 0 & \text{otherwise.} \end{cases}$
- **Nilpotent minimum t-norm:** $T(x,y) = \begin{cases} 0 & \text{if } x + y \leq 1, \\ min(x,y) & \text{otherwise.} \end{cases}$

5.2.2.2 Disjunction Operator

The truth table for the disjunction operator, also called OR operator, of the classical binary logic can be seen in Table 5.2. This operator corresponds to the union operation in the classical set theory. Instead of considering only zero and one, the binary disjunction can be extended on the unit interval [0, 1] as follows [56]:

Definition 5.2.3 *Binary disjunction on the unit interval is a function* $f : [0,1]\ x\ [0,1] \rightarrow [0,1]$ *which satisfies the following conditions* $\forall x, y, z \in [0,1]$*:*

i. $f(x,0) = x$*, and*

ii. Monotonicity: $(x \leq y) \Rightarrow (f(x,z) \leq f(y,z)\ \&\ f(z,x) \leq f(z,y))$.

The first condition makes sure that '0' is interpreted as an identity element. The monotonicity condition assures that an increase in the disjunct's value does not produce a decrease in the resulting truth value.

Similar to the conjunction operator, there is not a single implementation of the disjunction operator for fuzzy logic systems. Different correspondences between operands may be defined according to needs. The disjunction operation in fuzzy logic is typically modeled as a triangular conorm, t-conorm or s-norm for short.

Definition 5.2.4 *A triangular conorm is a function* $s : [0,1]\ x\ [0,1] \rightarrow [0,1]$ *which satisfies the following conditions* $\forall x, y, z \in [0,1]$*:*

i. Boundary: $s(x,0) = x$ *and* $s(1,1) = 1$*,*

ii. Monotonicity: $(x \leq y) \Rightarrow (s(x,z) \leq s(y,z))$*,*

iii. Commutativity: $s(x,y) = s(y,x)$*,*

iv. Associativity: $s(x, s(y,z)) = s(s(x,y),z)$*.*

The above conditions for t-conorms are similar to the conditions for t-norms. The boundary condition ensures that the behavior of the disjunction operator for crisp sets is modeled correctly. The monotonicity condition states that an increase in x and y does not produce a decrease in the result. The commutativity condition guarantees that the order in which x and y are combined does not make any difference. Finally, the associativity condition ensures that any number of sets can be combined in any order.

Every t-conorm is a disjunction on the unit interval. Typical t-conorms that are seen in the literature are as follows [25]:

- **Gödel/Minimum t-conorm:** $S(x,y) = max(x,y)$.
- **Probabilistic sum t-conorm:** $S(x,y) = x + y - xy$.
- **Bounded sum/Łukasiewicz t-conorm:** $S(x,y) = max(1, x+y)$.
- **Drastic sum t-conorm:** $S(x,y) = \begin{cases} max(x,y) & \text{if } x = 0 \text{ or } y = 0, \\ 1 & \text{otherwise.} \end{cases}$
- **Nilpotent maximum t-conorm:** $S(x,y) = \begin{cases} 1 & \text{if } x + y \geq 1, \\ max(x,y) & \text{otherwise.} \end{cases}$

5.3 Fuzzy Rules and Fuzzy Inferencing

Fuzzy inferencing is the process that maps a set of inputs into a set of outputs using fuzzy logic. The relationship between inputs and outputs are formulated by means of if-then rules. The following is what happens in traditional rule-based inferencing:

Rule: IF x is A THEN y is B.

Evidence: x is A.

Conclusion: y is B.

This is a strict relation that the *consequent* can only be true when the *antecedent* is fully true. Fuzzy rules helps in describing non-strict relationships between input and output domains. The following is an example for fuzzy rule-based inferencing:

Rule: IF x is A THEN y is B.

Evidence: x matches A to a degree of μ.

Conclusion: y complies with B to a degree of μ.

There is not a single type of fuzzy rules, nor there is a single type of fuzzy inferencing systems. A type of fuzzy inferencing system generally requires a specific type of fuzzy rules. Different types of fuzzy rules emerge from where fuzzy sets are used. Fuzzy sets may be in a rule's antecedent, in its consequent, or in both. Although there are several fuzzy inferencing systems proposed in literature, two of them are widely accepted: Mamdani-type fuzzy inferencing system [38] and Sugeno-type fuzzy inferencing system [58, 59], also known as Takagi-Sugeno-Kang inferencing method. In Mamdani's method, fuzzy sets are used for both the antecedent and the consequent of a rule. A rule used in Mamdani-type inferencing has the form "IF x_1 is A_1 and x_2 is A_2 and $\cdots$ and x_n is A_n THEN y is B", where x_1 to x_n are the inputs, y is the output, A_1 to A_n and B are the fuzzy sets. However, in Sugeno-type inferencing only the antecedent contains fuzzy sets. The consequent is a crisp function of the inputs. A typical rule used in Sugeno-type inferencing has the form "IF x_1 is A_1 and x_2 is A_2 and $\cdots$ and x_n is A_n THEN $y = f(x_1, x_2, \cdots, x_n)$".

The interpretation of an if-then rule requires the evaluation of the antecedent, applying the result of the antecedent to the consequent, aggregating rule outputs and determining a final output [21]. The steps required for the evaluation of an antecedent are the same for both inferencing mechanisms. They are as follows:

i. **Fuzzification:** The membership value of each input is determined for the available fuzzy sets.

ii. **Fuzzy Composition:** Fuzzy predicates are combined by means of fuzzy composition operators and a single value that describe the firing strength of the rule, which is also called the degree of confidence of the rule, is calculated.

For Mamdani-type inferencing, the rest of the steps are as follows:

i. **Applying the result of the antecedent to the consequent:** If the rule's antecedent is true to some degree μ, then the consequent should also be true to the same degree μ. So, in this step, the rule's consequent is correlated with the degree of confidence of the rule. Clipping is the most common method used in this step which can be described as cutting the membership function of the output at the level of μ.

ii. **Aggregation:** Outputs of individual rules, which are fuzzy sets, are combined to generate a single fuzzy set.

iii. **Defuzzification:** A crisp output value is calculated from the combined fuzzy set.

In Sugeno-type inferencing, the output is calculated using the following formula:

$$x^* = \frac{\sum_{i=1}^{n} w_i f_i}{\sum_{i=1}^{n} w_i}$$

where w_i is the firing strength of the i^{th} rule, and f_i is the result of the output function for the i^{th} rule.

5.3.1 Fuzzification

Fuzzification is the process that maps a crisp value to fuzzy membership values. In other words, it is used to determine the degree of membership of a crisp value for each of the available fuzzy sets. Membership functions are used for this purpose. Selection of an appropriate membership function for an input depends on the characteristics of the input and the set of rules using it. Therefore, experts who have a good understanding of the rules and the input domain need to choose the appropriate membership functions. The following are the most commonly used membership functions.

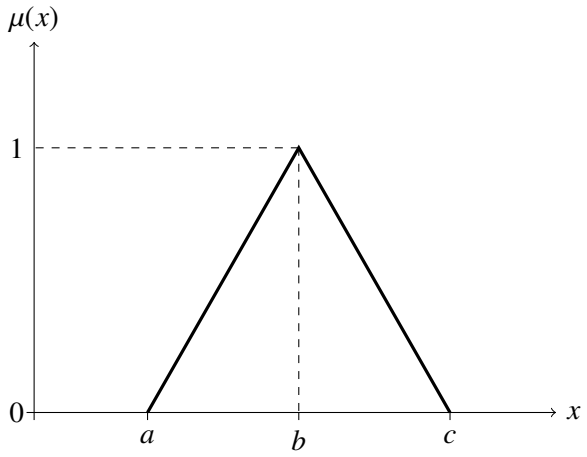


Figure 5.5: Triangular membership function

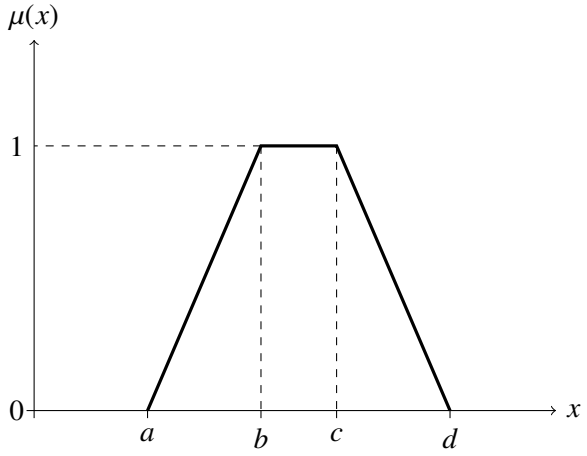


Figure 5.6: Trapezoidal membership function

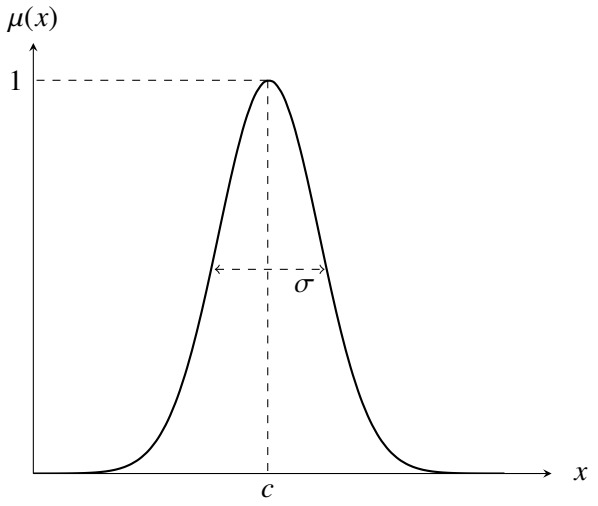


Figure 5.7: Gaussian membership function

i. A **triangular membership function** is characterized by 3 parameters: a, b and b. a and c are the left and right corners of the triangle on the $X-$axis. b is the value for which the membership grade is one.

$$\mu(x; l, c, r) = \begin{cases} 0 & \text{for } x \leq l \\ \dfrac{x-l}{c-l} & \text{for } l < x \leq c \\ \dfrac{r-x}{r-c} & \text{for } c < x \leq r \\ 0 & \text{for } x > r \end{cases}$$

ii. A **trapezoidal membership function** is characterized by 4 parameters: a, b, c and d. a and d are the left and right corners of the trapezoid on the $X-$axis. b and c are the projections of the left and right top corners of the trapezoid on the $X-$axis.

$$\mu(x; a, b, c, d) = \begin{cases} 0 & \text{for } x \leq a \\ \dfrac{x-a}{b-a} & \text{for } a < x \leq b \\ 1 & b < x \leq c \\ \dfrac{d-x}{d-c} & \text{for } c < x \leq d \\ 0 & \text{for } x > d \end{cases}$$

iii. A **gaussian membership function** is characterized by 2 values: σ and c. c is the center

and σ is the width of the membership function.

$$\mu(x; \sigma, c) = exp\left(\frac{-(x-c)^2}{2\sigma^2}\right)$$

5.3.2 Defuzzification

Defuzzification is the reverse of the fuzzification operation. If the output of a rule is characterized by one or more fuzzy sets, it may need to be mapped into a single scalar quantity. The process of mapping a fuzzy quantity into a single crisp value is called defuzzification. Before employing the defuzzification operation, output fuzzy sets are aggregated, by a union operator, into a single fuzzy set. Then, one of the several defuzzification methods, which have been proposed in the literature, is used to get a single scalar value. Similar to the fuzzification methods, selection of a defuzzification method depends on the requirements and experts who have a good understanding of the output domain need to select the appropriate method. The following are the most widely used defuzzification methods.

i. **The max-membership defuzzification** method is one of the most simple techniques used for defuzzification. It gives the crisp value with the highest degree of membership.

$$\mu(x^*) \geq \mu(x), \quad \forall x \in X$$

where x^* is the scalar output.

ii. **The mean of max defuzzification** method calculates the average of the abscissa of the points that has the highest membership value.

$$x^* = \frac{\sum_{i=1}^{n} x_i}{n}$$

where $\forall i, \forall x,\ \mu(x_i) \geq \mu(x)$.

iii. **The centroid defuzzification** method gives the $X-$coordinate of the point that is the centre of the area enclosed by the membership function that represents the aggregated fuzzy set and the $X-$axis. For a continuous membership function, it is calculated as follows:

$$x^* = \frac{\int_s x_i\, \mu(x)\, dx}{\int_s \mu(x)\, dx}$$

In the case of discrete values, it is calculated as follows:

$$x^* = \frac{\sum_{i=1}^{n} x_i \, \mu(x_i)}{\sum_{i=1}^{n} \mu(x_i)}$$

Although it is one of the most reliable defuzzification methods, it is computationally complex and costly.

iv. **The weighted average defuzzification** method calculates a weighted average of the outputs of rules.

$$x^* = \frac{\sum_{i=1}^{n} w_i \, \mu(x_i)}{\sum_{i=1}^{n} \mu(x_i)}$$

5.4 Fuzzy Composition

The first step in the processing of a fuzzy rule is to perform the fuzzification of inputs, and then to evaluate the antecedent to calculate the firing strength of it. For this purpose, a fuzzy rule engine needs to apply the necessary fuzzy composition operators, which is called fuzzy composition.

If a rule's antecedent is in a conjunctive normal form, the resulting value for the firing strength of the rule is calculated by combining the degree of membership value for each conjunct by means of fuzzy conjunction operation. But before this composition can take place, the degree of membership value for each conjunct needs to be computed. If a conjunct contains a single predicate, that value is equal to the membership value of the predicate. However, if the conjunct contains more than one predicate, which means that the conjunct is a disjunction of several predicates, the degree of grade for the conjunct is calculated by combining the degree of membership values for each predicate by means of fuzzy disjunction operation. For the rules that have antecedents in conjunctive normal form, fuzzy disjunction operation has precedence over fuzzy conjunction operation. Fuzzy inference engines need to apply disjunction operators before conjunction operators.

Example. Let us consider the following rule:

$$\text{IF } ((a \text{ is } A) \text{ and } ((b \text{ is } B) \text{ or } (c \text{ is } C)) \text{ and } ((d \text{ is } D) \text{ or } (e \text{ is } E))) \text{ THEN } o \text{ is } O \qquad (5.1)$$

This rule has three conjuncts: "a is A", "(b is B) or (c is C)", and "(d is D) or (e is E)". Let us use min and max as the fuzzy conjunction and disjunction operators respectively. As the first conjunct contains a single proposition, no fuzzy composition operation is required. However, for the second and third conjuncts, fuzzy disjunction operator needs to be used. The maximum degree of membership value of the propositions "b is B" and "c is C" determines the degree of grade for the second conjunct. Similarly, degree of grade for the third conjunct is the maximum of the degree of membership values for the propositions "d is D" and "e is E". Finally, the firing strength of the rule is equal to the minimum of the degrees of grades of these three conjuncts.

5.4.1 Fuzzy Composition Graph

Graphical modeling tools are used to represent the interactions in a structural system in a concise and compact manner. The well known adage "*a picture is worth a thousand words*" is the most important motivation for the use of graphical modeling tools. Visually representing components and their relations makes the analysis of the systems being modeled easier. Using a graphical tool to represent the compositional relationships between the predicates of a rule helps in visualizing what an inference engine does while processing the rule.

Petri nets are one of the popular graphical modeling tools that are used for modeling systems that are dynamic, distributed, concurrent, asynchronous and non-deterministic [42]. However, one of the major weaknesses of petri nets is the complexity problem; i.e., they become too large even for a modest-size system [42]. Moreover, compositional relationships of predicates are static and do not change over time. So, instead of using petri nets, a more simple graphical tool can serve the same purpose. To that end, a weighted directed acyclic graph (wDAG) can be used to represent the compositional pattern in the antecedent of a rule.

Definition 5.4.1 *A fuzzy composition graph (FCG) is a wDAG that is used to show the fuzzy logical relationships between the predicates in the antecedent of a rule. In a FCG, vertices model predicates, edges model fuzzy disjunction and conjunction operations, and weights associated with edges correspond to fuzzy membership values for the predicates.*

Figures 5.8(a)–(b) show two different wDAG representations of the composition of the elements in the antecedent of the above rule. For the sake of simplicity, the propositions are

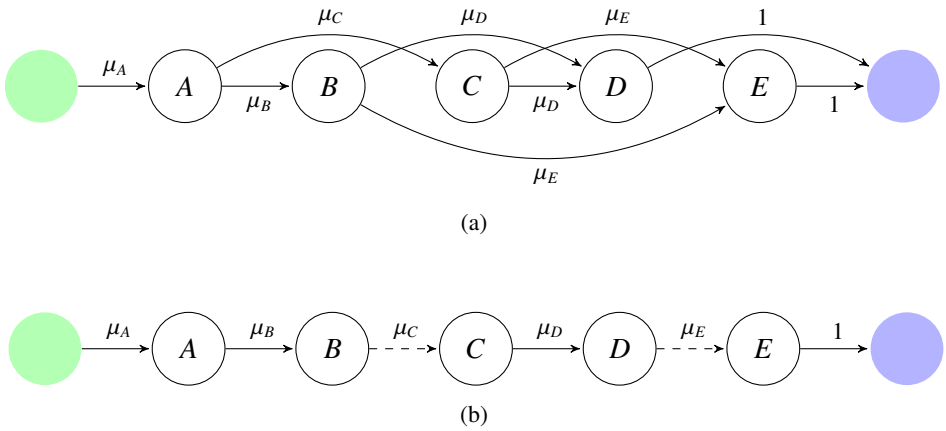


Figure 5.8: Weighted directed acyclic graphs for fuzzy composition: (a) Ordinary wDAG (b) A modified wDAG with different types of edges

labeled as single letters. In the first figure an ordinary wDAG is used to represent the relationships between the propositions. Starting with the source vertex, if the sink vertex can be reached, then this means that the antecedent is satisfied. The problem with the first graph is that if it is used for determining firing strength of the rule, selection of a path during inference is non-deterministic since it is possible for a vertex to have multiple outgoing edges. As a result, in order to come up with a correct decision, fuzzy inference engine needs to consider every possible path. After reaching the sink vertex or reaching a vertex that has an outgoing edge with a weight of zero, if there are still paths that have not been taken into account, the engine needs to backtrack to calculate the firing strengths by going over the remaining possible paths.

The second figure shows an alternative representation that we propose for the same fuzzy composition. It contains only a single path and the number of edges are constant, which is one less than the number of vertices.

Definition 5.4.2 *A fuzzy composition graph for a single rule (FCG-SR) is a wDAG that is characterized by a 7-tuple* $G = (V,\ E,\ s,\ f,\ T,\ F_w,\ F_t)$.

- V *is a set of vertices* $\{v_1,\ v_2,\ \cdots,\ v_n\}$,
- E *is a set of edges* $\{e_1,\ e_2,\ \cdots,\ e_m\}$,
- s *is a source vertex,* $s \in V$,

- *f is a sink vertex, $f \in V$,*
- *T is a set of edge types $\{t_1,\ t_2,\ \cdots,\ t_k\}$,*
- *F_w is a function $F_w : E \rightarrow [0, 1]$ that maps an edge in E to a real number between zero and one,*
- *F_t is a function $F_t : E \rightarrow T$ that maps an edge in E to a type,*

FCG-SR shown in 5.8(b) has two different types of edges: straightline and dashed. Edges with a straight line model fuzzy conjunction operation whereas edges with a dashed line model fuzzy disjunction operation. In both graphs, a weight associated with an edge correspond to the degree of membership value for the vertex that this edge is an incoming edge of. The weight for the incoming edges of the sink vertex should be one as these edges represent the fuzzy conjunction and the value calculated up until to the vertex one hop before the sink vertex is not altered as one is identity element for the conjunction operation. Vertices model fuzzy predicates or propositions, and during inference, they keep the values that are computed up until to them.

In a FCG-SR, there are three types of vertices:

- *Source vertex*. It only has outgoing edges, and it is the starting point for the processing carried out by the fuzzy inference engine that processes the associated rule. It does not correspond to any predicate in the antecedent of the rule. A FCG-SR needs to contain exactly one source vertex.
- *Sink vertex*. It only has incoming edges. It is the final node to be reached by a fuzzy inference engine, and it does not correspond to an actual predicate of a rule. Reaching this node implies that the rule being processed is satisfied. A FCG-SR needs to contain exactly one sink vertex.
- *Intermediary vertex*. It models a predicate or a proposition in the antecedent of a rule. Each intermediary vertex contain at least one incoming edge and one outgoing edge. In a FCG-SR, there must be at least one intermediary vertex.

The firing strength of a rule is computed by starting at the source vertex and going forward until the sink vertex is reached, and applying fuzzy composition operators along the path. By

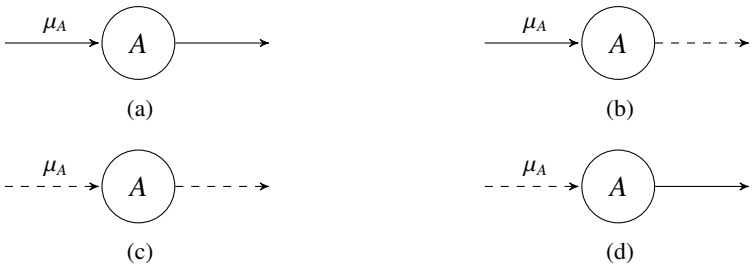


Figure 5.9: Possible edge-vertex-edge triplets for a FCG-SR

the time sink node is reached, the firing strength of the rule has been determined as well.

In the first graph in Figure 5.8, in order to go along a path, each edge on that path need to have a non-zero fuzzy membership value. If an inference engine encounters an edge with a value of zero, it needs to backtrack to explore other paths. For such a graph, a depth first search strategy can be used by a fuzzy inference engine. Along a path, the minimum weight value determines the resulting fuzzy composition value for that path.

In the second graph, even if an edge has a weight of zero, it may still be possible to carry on on that path. The fuzzy disjunction operation has precedence over the conjunction operation. In other words, disjunction operation needs to be applied before conjunction operation. So, when an inference engine reaches a vertex that has an outgoing edge with a dashed line, it needs to traverse subsequent edges with dashed lines before applying fuzzy conjunction operation. That is, if fuzzy inference engine encounters a vertex that has a straight-lined incoming edge and dashed-lined outgoing edge, before combining the fuzzy membership value calculated up until to that node with the weight associated with the lastly traversed edge, all the subsequent dashed edges need to be traversed to select the maximum weight value, which is then used in the fuzzy conjunction. During this calculation, if an edge with a weight value of zero is encountered, it is legal to continue traversing dashed lines as long as the final maximum value is not zero. As it can be seen, the edge-vertex-edge triplet changes how a fuzzy inference engine behaves. Figure 5.9 shows all possible edge-vertex-edge triplets for a FCG-SR.

- Figure 5.9(a) shows a vertex representing a predicate that is a conjunct on its own.
- Figure 5.9(b) shows a vertex representing a predicate that is the first element of a disjunctive clause.
- Figure 5.9(c) shows a vertex representing a predicate that is a component of a disjunc-

tive clause, and it is neither the first nor the last component of that disjunctive clause.

- Figure 5.9(d) shows a vertex representing a predicate that is the last element of a disjunctive clause.

Algorithm 5 shows how a fuzzy inference engine computes the firing strength of a rule represented by a FCG-SR.

Algorithm 5 Computing Firing Strength of a Single Rule

```
1:  INPUT: G = (V, E)
2:  OUTPUT: RS = firing strength of the rule
3:  RS = 1
4:  v = "source vertex"
5:  e_in ← NIL
6:  m = 1
7:  while v is not equal to "sink vertex" do
8:      e_out ← v.outgoing_edge
9:      if e_out is "dashed edge" then
10:         m ← e_in.weight
11:         while v = v.next do
12:             e_in ← e_out
13:             e_out ← v.outgoing_edge
14:             m ← max(e_in.weight, m)
15:             if e_out is "straight edge" then
16:                 break
17:             end if
18:         end while
19:     else if e_in is not NIL then
20:         m ← e_in.weight
21:     end if
22:     RS ← min(RS, m)
23:     e_in ← e_out
24:     v ← v.next
25: end while
```

When an ordinary wDAG is used by a fuzzy inference engine, the complexity for calculating the firing strength of a rule is $O(|V| + |E|)$ because whether a depth first or breadth first search algorithm is used, each vertex and edge is processed only once. On the other hand,

the complexity of Algorithm 5 is $O(2|V| - 1)$. Although each vertex and edge is processed only once similar to the previous case, the number of edges is equal to the number of vertices minus one. If the antecedent of a rule contains one or more disjunctive clauses, number of edges is always greater than or equal to number of vertices in an ordinary wDAG. Therefore, $O(2|V| - 1) \leq O(|V| + |E|)$. This makes us sure that FCG-SR is a better alternative for representing fuzzy compositions.

5.5 Rule Reduction

One of the problems of using fuzzy rule-based systems is that total number of rules may be large [6], and this is a major problem for wireless sensor networks that have sensor nodes with limited storage and processing capability. Compared to crisp rules, generally more fuzzy rules are required for describing an input output mapping while the latter one is the more intuitive way to express this relationship. For example, we can define very low, low, normal, high and very high as the fuzzy sets for the body temperature of a person. Compared to a crisp rule which contains a predicate that checks the temperature value being above 38 in its antecedent, two fuzzy rules may have to be used that only differ in the predicate that is related to body temperature: one of the rules is for a body temperature classified as high, and the other one is for very high.

Same input composition pattern may be used for several rules where the only difference is in fuzzy sets that input parameters are members of. To illustrate what is meant, let us consider the following rules:

Rule 1: IF x is X_1 and y is Y_1 THEN z is Z_1

Rule 2: IF x is X_2 and y is Y_1 THEN z is Z_1

Rule 3: IF x is X_1 and y is Y_2 THEN z is Z_1

Rule 4: IF x is X_2 and y is Y_2 THEN z is Z_2

where x and y are the input parameters, z is the output parameter, and X_i, Y_i and Z_i are the associated fuzzy sets. Although the event pattern for the first, second and third rules contains the composition of the same input parameters and producing the same output, they need to be

Table 5.3: Possible input combinations for 3 input parameters that each can be a member of one of 4 fuzzy sets

x	y	z
X_1	Y_1	Z_1
X_1	Y_1	Z_2
X_1	Y_1	Z_3
X_1	Y_1	Z_4
X_1	Y_2	Z_1
X_1	Y_2	Z_2
X_1	Y_2	Z_3
X_1	Y_2	Z_4
X_1	Y_3	Z_1
$\vdots$	$\vdots$	$\vdots$
X_4	Y_3	Z_4
X_4	Y_4	Z_1
X_4	Y_4	Z_2
X_4	Y_4	Z_3
X_4	Y_4	Z_4

written down separately in a typical fuzzy rule-based system. However, number of rules need to be reduced, as we have done in Chapter 4, in order to reduce the resource requirements for storage and processing.

In the worst case, the number of rules required for an event pattern is exponential depending on the number of input parameters and the number of fuzzy sets that input parameters may be a member of. Let us assume that an event pattern in the antecedent of a rule contains the conjunction of three inputs and each of them can be members of four fuzzy sets. Then, the number of rules for the worst case is $4 \cdot 4 \cdot 4 = 4^3$. Possible input combinations for such an event pattern can be seen in Table 5.3.

If an event pattern contains composition of n input parameters, then, in the worst case, the number of rules which use the same input parameters with the same event pattern can be calculated by the following formula:

$$\prod_{i=1}^{n} s_i \ ,$$

where s_i is the number of fuzzy sets that the i^{th} input can be a member of.

In this section we propose a mechanism that reduce total number of rules having a specific pattern in their antecedent to the number of output fuzzy sets that may be present in the

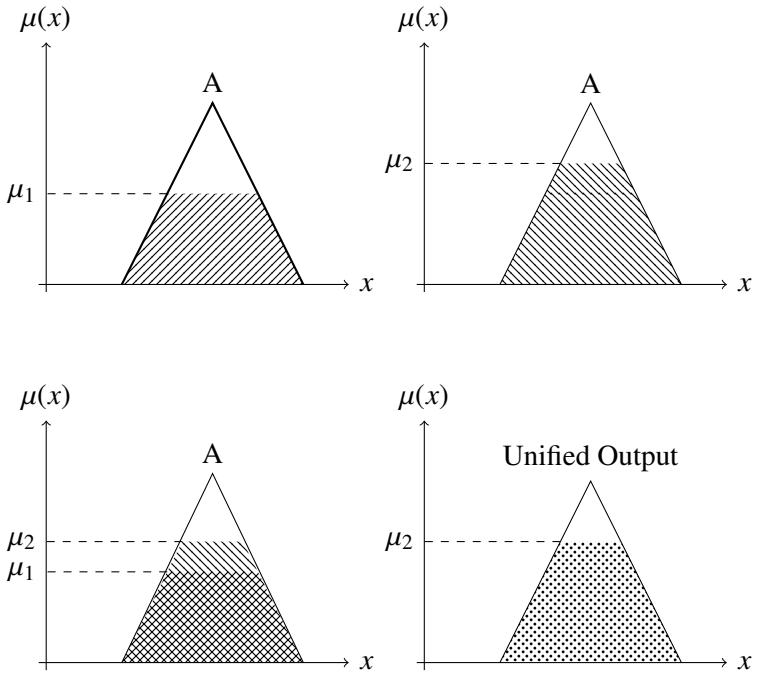


Figure 5.10: Output Unification

consequent of these rules. Our proposed mechanism combine all rules with the same event pattern and the same output into a single rule.

In a typical fuzzy inference system, fuzzy rules are evaluated independently and outputs of them are unified. However, if two rules produce the same output; i.e., the output parameter is assigned to the same output fuzzy set, the result of the unification only depends on the maximum of the firing strengths of these rules. Illustration of this can be seen in Figure 5.10.

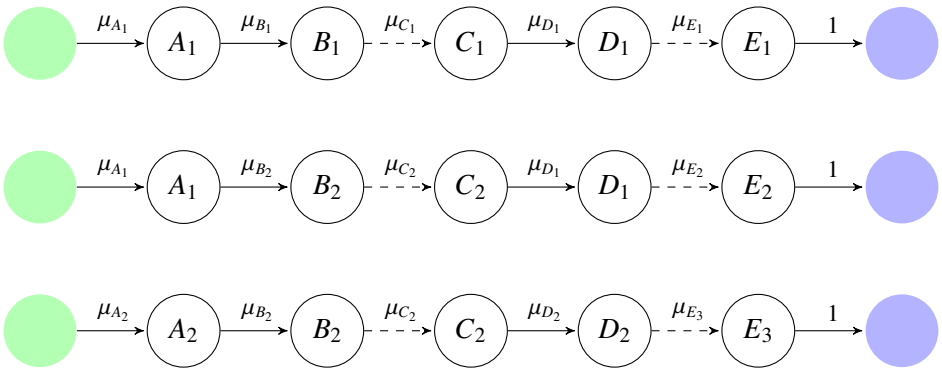


Figure 5.11: Fuzzy composition graphs for three rules

Example. Let us consider the following rules:

$$\text{IF } ((a \text{ is } A_1) \text{ and } ((b \text{ is } B_1) \text{ or } (c \text{ is } C_1)) \text{ and } ((d \text{ is } D_1) \text{ or } (e \text{ is } E_1))) \text{ THEN } o \text{ is } O \quad (5.2)$$

$$\text{IF } ((a \text{ is } A_1) \text{ and } ((b \text{ is } B_2) \text{ or } (c \text{ is } C_2)) \text{ and } ((d \text{ is } D_1) \text{ or } (e \text{ is } E_2))) \text{ THEN } o \text{ is } O \quad (5.3)$$

$$\text{IF } ((a \text{ is } A_2) \text{ and } ((b \text{ is } B_2) \text{ or } (c \text{ is } C_2)) \text{ and } ((d \text{ is } D_2) \text{ or } (e \text{ is } E_3))) \text{ THEN } o \text{ is } O \quad (5.4)$$

The firing strengths of these rules are calculated as follows:

$$s_1 = min(\mu_{A_1},\ max(\mu_{B_1},\ \mu_{C_1}),\ max(\mu_{D_1},\ \mu_{E_1}))$$
$$s_2 = min(\mu_{A_1},\ max(\mu_{B_2},\ \mu_{C_2}),\ max(\mu_{D_1},\ \mu_{E_2}))$$
$$s_3 = min(\mu_{A_2},\ max(\mu_{B_2},\ \mu_{C_2}),\ max(\mu_{D_2},\ \mu_{E_3}))$$

The union of the outcomes of these rules is $s = max(s_1, s_2, s_3)$. Figure 5.11 shows the graphs that would be used if these rules were represented and evaluated separately. Our proposition is that rules with same output can be combined into a single rule. The following is the combined rule:

IF ((a is $A_1\{1,2\} \mid A_2\{3\}$) and
($(b$ is $B_1\{1\} \mid B_2\{2,3\})$ or $(c$ is $C_1\{1\} \mid C_2\{2,3\}))$ and
$((d$ is $D_1\{1,2\} \mid D_2\{3\})$ or $(e$ is $E_1\{1\} \mid E_2\{2\} \mid E_3\{3\}))$) THEN o is O

The numbers that are seen next to fuzzy set literals are used to preserve the relationships between propositions. They correspond to rule numbers. If all input combinations are covered by a set of rules which have identical output, there would not be a need for these numbers. Possible input combinations for the above event pattern is:

$$\{(A_1, B_1, C_1, D_1, E_1),\ (A_1, B_1, C_1, D_1, E_2),\ (A_1, B_1, C_1, D_1, E_3),\ \cdots,\ (A_2, B_2, C_2, D_2, E_3)\}.$$

However, for the above rules, $(A_1, B_1, C_1, D_1, E_1)$, $(A_1, B_2, C_2, D_1, E_2)$ and $(A_2, B_2, C_2, D_2, E_3)$ are the only combinations. In general, union of input combinations of a set of rules that can be combined constitutes a subset of all possible input combinations.

Definition 5.5.1 *A fuzzy composition graph for a combined rule (FCG-CR) is an edge-labeled wDAG that is characterized by a 9-tuple* $G = (V,\ E,\ s,\ f,\ T,\ L,\ F_w,\ F_t,\ F_l)$.

- V *is a set of vertices* $\{v_1,\ v_2,\ \cdots,\ v_n\}$,

- *E is a set of edges* $\{e_1,\ e_2,\ \cdots,\ e_m\}$,
- *s is a source vertex,* $s \in V$,
- *f is a sink vertex,* $f \in V$,
- *T is a set of edge types* $\{t_1,\ t_2,\ \cdots,\ t_k\}$,
- *L is a set of labels* $\{l_1,\ l_2,\ \cdots,\ l_r\}$ *that represents path identifiers,*
- F_w *is a function* $F_w : E \rightarrow [0, 1]$ *that maps an edge in E to a real number between zero and one,*
- F_t *is a function* $F_t : E \rightarrow T$ *that maps an edge in E to a type,*
- F_l *is a function* $F_l : E \rightarrow L^l$ *that maps an edge in E to a set of labels.*

Figure 5.12 shows the FCG-CR that can be used for the above combined rule. Edges in the graph have two labels:

i. The blue label above an edge corresponds to the numbers next to fuzzy literals, and it identifies a specific path from the source to the sink vertex. Therefore, it is called the *path identifier*. A path identifier plays an important role during traversing graphs. If a vertex has an incoming edge with a specific path identifier, the outgoing edge that traversing can carry on should have that identifier as well. Considering a vertex, if there are multiple incoming edges and/or the incoming edge has multiple identifiers, than the union of path identifiers in the outgoing edge(s) should contain all of the identifiers in the incoming edge(s) as well.

ii. The red label below an edge represent the fuzzy membership value for the proposition represented by the vertex that this edge is an incoming edge of.

A vertex in such a graph keeps the results of fuzzy compositions carried out up until to that vertex. It may have multiple values assigned to itself. Actually, a value need to be computed for each path identifier that is present in the incoming edges of that vertex.

In order to determine the firing strength of a combined rule using the FCG-CR created for that rule, breadth first searching is a reasonable strategy. Breadth first search algorithm uses each

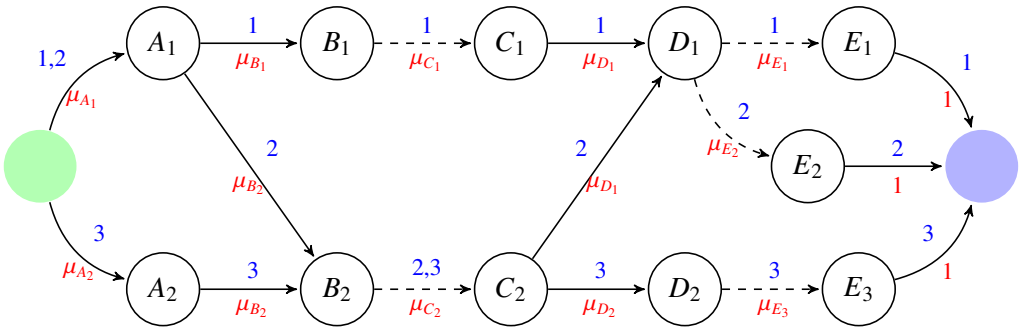


Figure 5.12: Fuzzy composition graph for a combined rule

vertex and edge only once. Therefore, the time complexity of such a strategy is $O(|V| + |E|)$ where $|V|$ is the number of vertices and $|E|$ is the number of edges. If a depth first search strategy were employed, the time complexity for the algorithm would to be $\Omega(|V| + |E|)$; i.e., $(|V| + |E|)$ would be the lower bound. This is due to the fact that a vertex or an edge has to be processed at least once but some of them may be processed more than once, when at least one edge has more than one path identifier assigned to it.

Algorithm 6 shows the pseudo code for determining the degree of grade of a combined rule. Starting with the source vertex, each vertex is visited in a breadth first manner. For this purpose a queue is used. The vertices that could be reached from a vertex is put into the queue. After the processing of the current vertex, the next vertex to be visited is dequeued from that queue. During the traversal of FCG-CR, an array is used to store the results of evaluations for each path identifier. If a dashed edge is met, an intermediate array is used to store the maximum membership value along the path with dashed edges. When a straight line is encountered once again, these maximum values are used to update the corresponding values in the initial array. When the sink vertex has finally been reached, the array contains the results of the fuzzy composition for each path identifier. Maximum of these values determines the firing strength of the combined rule.

Figure 5.13 is an illustration that shows the steps of the computation done to determine the firing strength of a combined rule. For this example, the following fuzzy membership values

Algorithm 6 Computing Firing Strength of a Combined Rule

INPUT: $G = (V, E)$

OUTPUT: RS = firing strength of the rule

FUNCTION *void enqueue*(q, v): insert v to q

FUNCTION *vertex dequeue*(q): pull next entry from q and return it

FUNCTION *edge incoming_edge*(v, p): return incoming edge having label p

```
1:  enqueue(q,"source vertex")
2:  RS ← 0
3:  ∀i, m[i] ← 0
4:  ∀i, val[i] ← 1
5:  while v ← dequeue(q) is not "sink vertex" do
6:      for all outgoing edge e_out of v do
7:          if v.next(e_out) is not in q then
8:              enqueue(q, v.next(e_out))
9:          end if
10:         if v is "source vertex" then
11:             Continue
12:         end if
13:         for all identifier p of e_out do
14:             if e_out.type is dashed then
15:                 m[p] ← max(incoming_edge(v, p).weight, m[p])
16:             else if incoming_edge(v, p).type is dashed then
17:                 m[p] ← max(incoming_edge(v, p).weight, m[p])
18:                 val[p] = min(m[p], val[p])
19:             else
20:                 val[p] = min(incoming_edge(v, p).weight, val[p])
21:             end if
22:         end for
23:     end for
24: end while
25: for p = 1 to n do
26:     RS ← max(val[p], RS)
27: end for
```

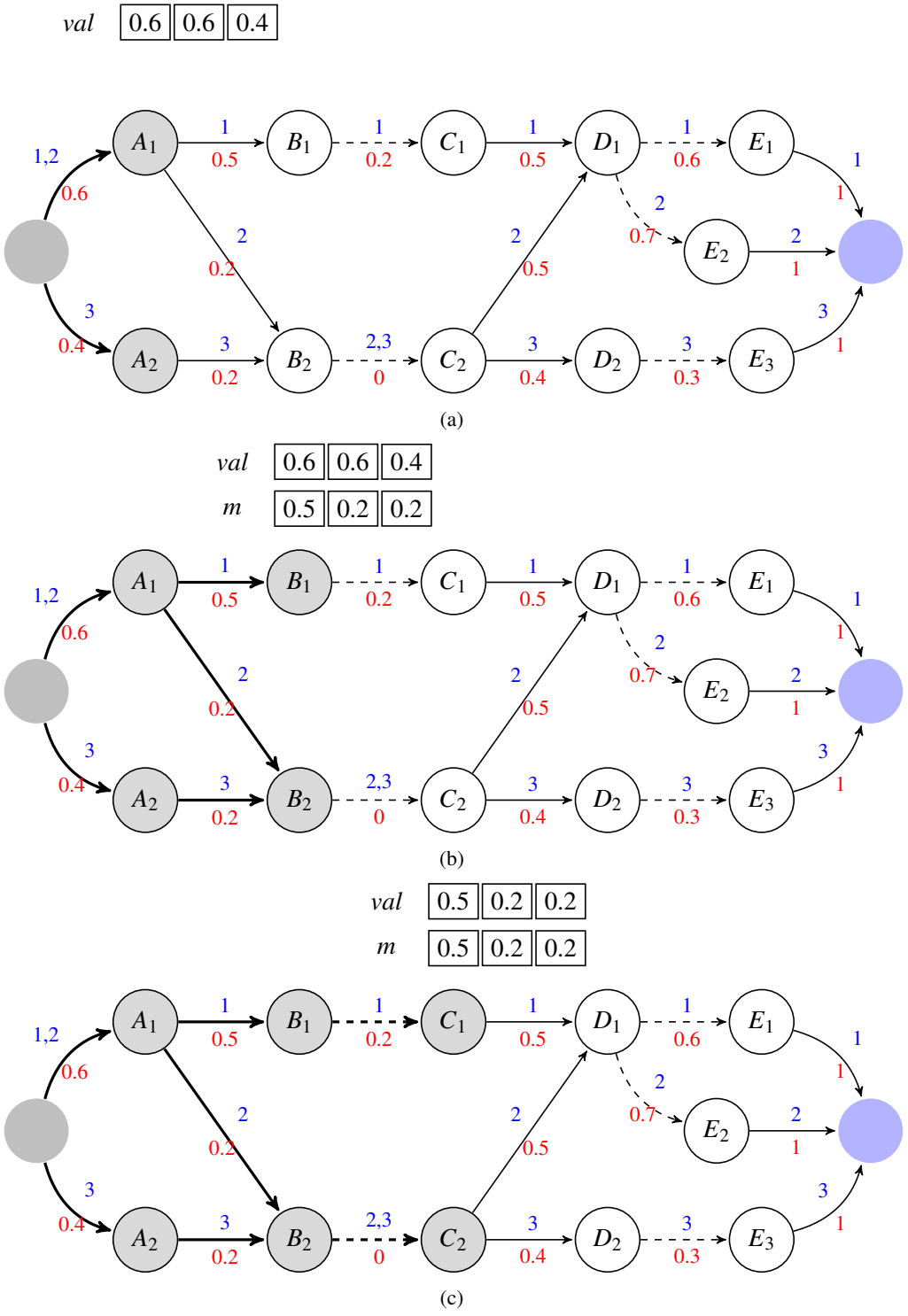
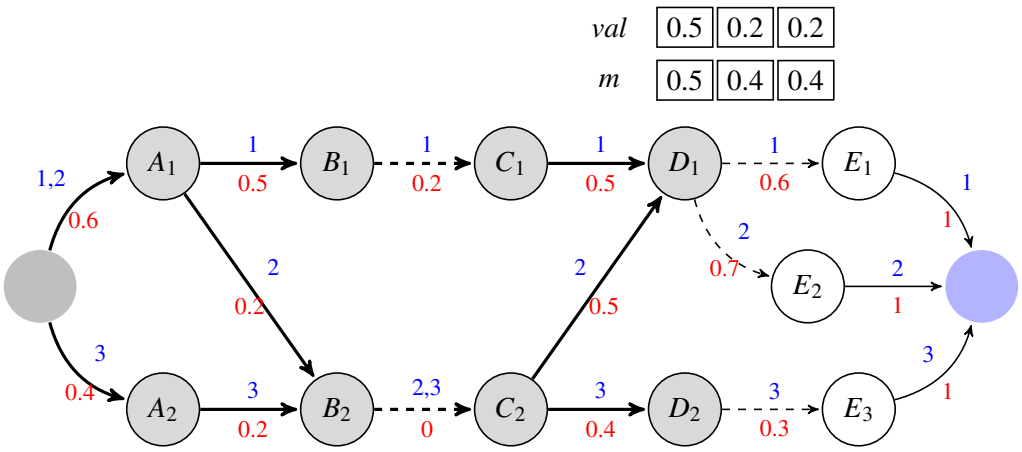
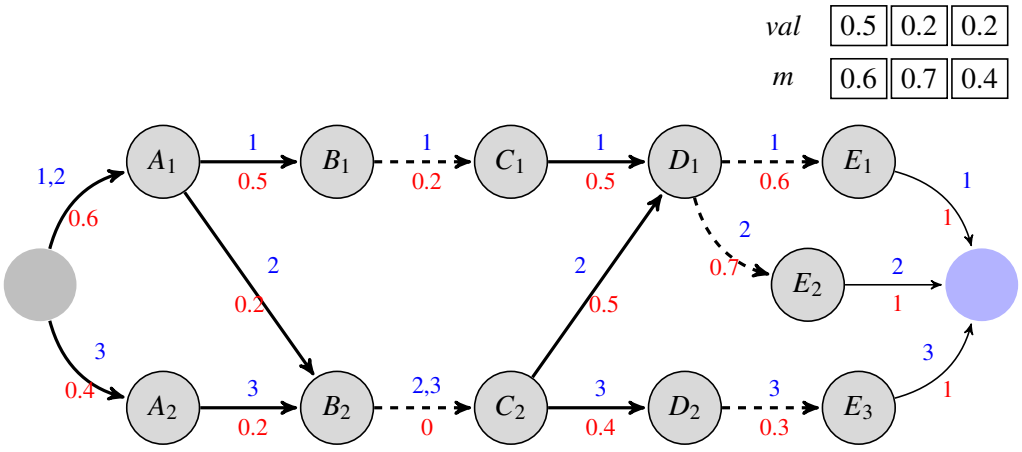


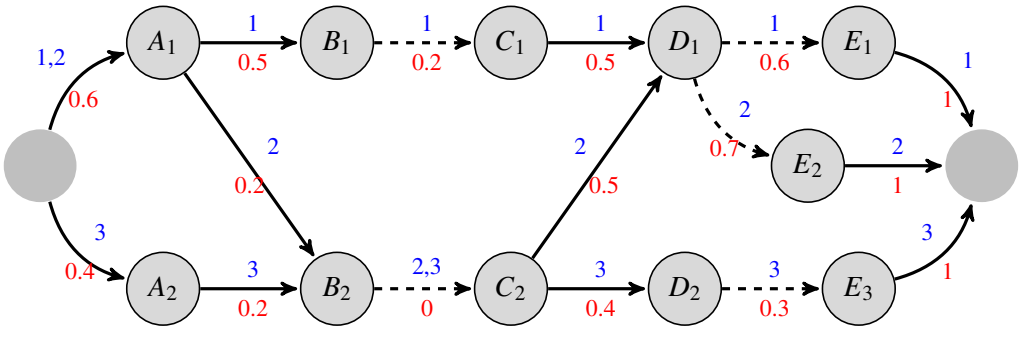
Figure 5.13: Fuzzy composition for a combined rule



(d)



(e)



$Firing\ Strength = max\{0.5, 0.2, 0.2\} = 0.5$

(f)

Figure 5.13: Fuzzy composition for a combined rule (cont.)

are used for the rule that is formed by combining (5.2), (5.3) and (5.4):

$$\mu_{A_1} = 0.6 \qquad \mu_{A_2} = 0.4$$
$$\mu_{B_1} = 0.5 \qquad \mu_{B_2} = 0.2$$
$$\mu_{C_1} = 0.2 \qquad \mu_{C_2} = 0$$
$$\mu_{D_1} = 0.5 \qquad \mu_{D_2} = 0.4$$
$$\mu_{E_1} = 0.6 \qquad \mu_{E_2} = 0.7 \qquad \mu_{E_3} = 0.3$$

Starting with the source vertex, fuzzy inference engine first visits A_1, and as the outgoing edge of A_1 is straight, the array that keeps the latest value of the fuzzy composition completed up to that vertex is set to [0.6, 0.6, -]. As the traversed edge contains two path identifiers, two entries in the array are updated that correspond to the path identifiers. In the second step, A_2 is visited and the array is modified as [0.6, 0.6, 0.4]. After that, B_1 is visited, but it has a dashed outgoing edge, which means the predicate represented by this vertex is part of a disjunctive clause. So instead of modifying the first array, a temporary array is used to store the computations for the disjunctive clause. The temporary array m is modified as [0.5, -, -]. Then vertex B_2 is visited, and in a similar manner, m is modified as [0.5, 0.2, 0.2]. The next vertex to be visited is C_1, and the first element of m is set to maximum of the value of the first entry of m and the weight associated with the traversed edge, which is 0.5. So, m is not modified, and as the outgoing edge signifies the end of the disjunctive clause, the initial array is modified using the values in m. The first entry of the initial array is set to the minimum of the first entry of the initial array and the first entry of m. After this step, the initial array looks like [0.5, 0.6, 0.4]. Next, C_2 is visited and performing the steps similar to the steps described for C_1, the fuzzy inference engine modifies the initial array as [0.5, 0.2, 0.2]. Continuing on the graph, when the fuzzy inference engine reaches the sink vertex, the final look of the array becomes [0.5, 0.2, 0.2]. After reaching the sink vertex, the final step is to find the maximum of the values in the array, which is the firing strength of the combined rule.

5.5.1 Distributed Fuzzy Composition

For a combined rule that is centrally processed, there is only one source and one sink vertex. The sink vertex represents the state that the rule antecedent is satisfied, and when the sink is reached, the fuzzy inference engine selects the maximum rule firing strength among the

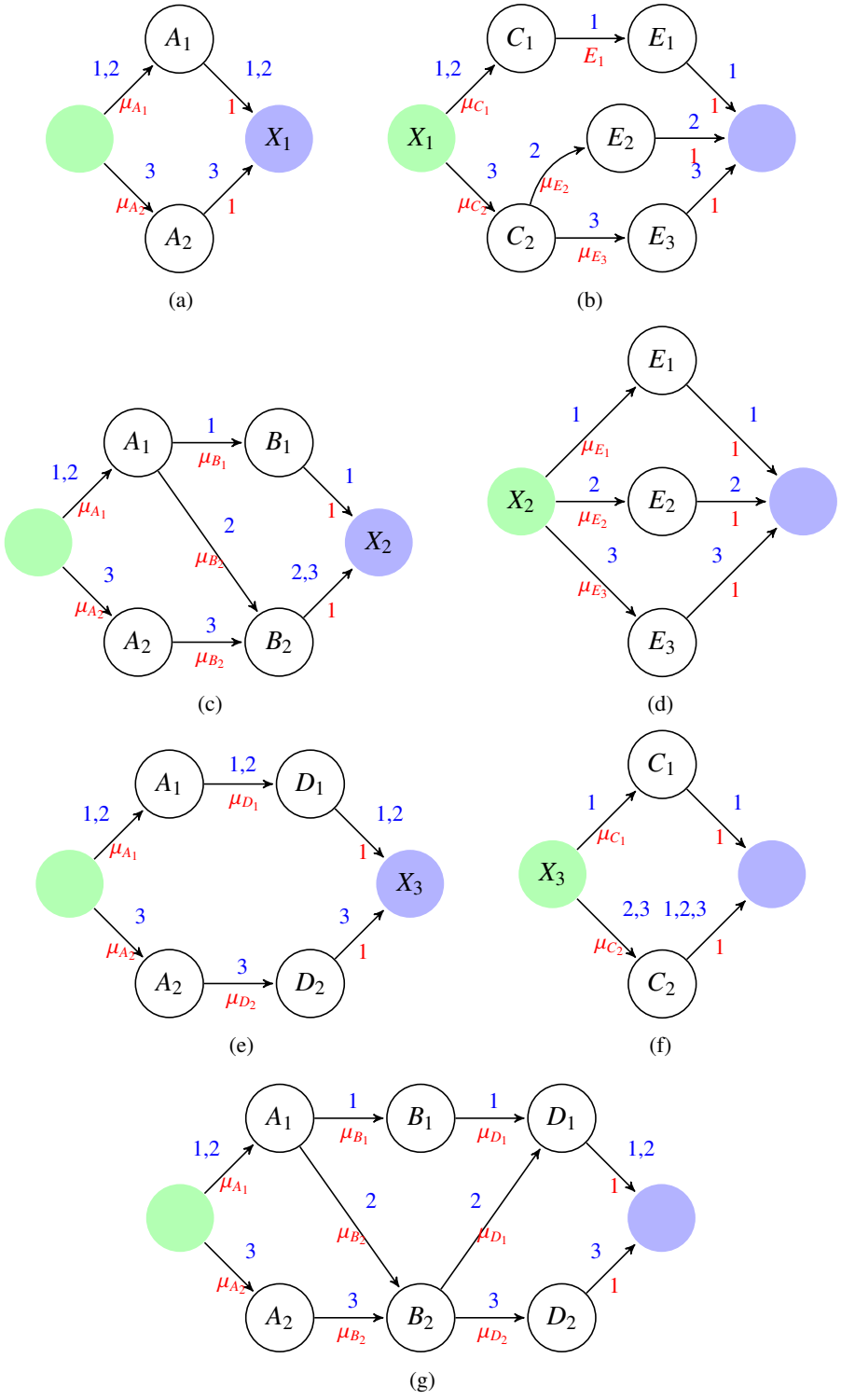


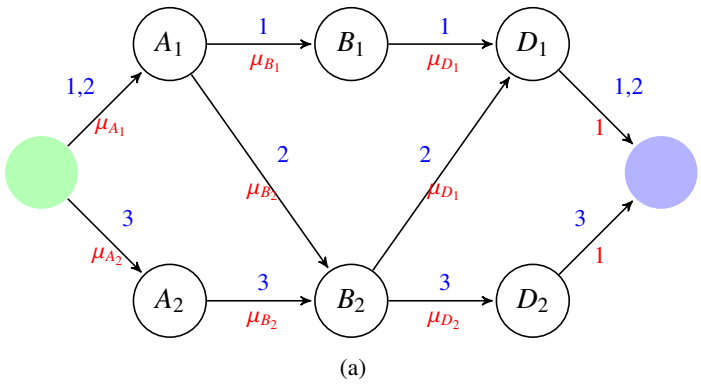
Figure 5.14: Fuzzy composition graphs for sub-rules (2 layer decomposition): (a), (c), (e), (g) are for the first layer, and (b), (d), (f) are for the second layer

set of computed values. However, when decomposition of rules is considered, a sink vertex for a sub-rule does not necessarily imply the satisfaction of the original rule. Moreover, if this is the case, the sink vertex should not unite the fuzzy membership values by taking the maximum of them, because the sub-rules that are decomposed from the rule capture partial states of processing, not the whole processing of the rule, and this partial state should be made available to the upper layers where the processing could carry on. Considering the example in the graph in Figure 5.12, let us assume that the inputs a, b and d can be processed at layer one, and c and d can be processed at layer two. Let us further assume that the inputs a and b have produced non-zero fuzzy membership values for the fuzzy sets A_1, A_2, B_1 and B_2. Then it is possible to reach the vertices B_1 and B_2, which keep the results of partial evaluations up until to these vertices. However, as the input d in the third conjunct does not yield a usable result, the partial evaluations need to be transferred to the next layer in the WSN network hierarchy. At the upper layer, the reachability of the non-decomposed rule's consequent is determined based on the fuzzy membership values of the fuzzy sets E_1, E_2 and E_3.

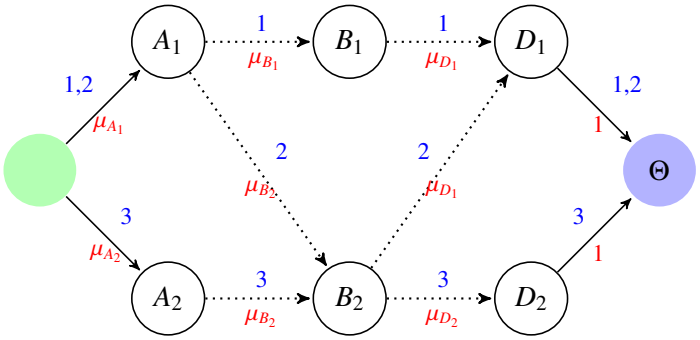
The fuzzy composition graph used for the sub-rules that are decomposed from a rule is similar to FCG-SR with the following differences:

- A sink vertex in a FCG-SR indicate the state that a conclusion is drawn. However the fuzzy composition graph used for the decomposed rules may or may not model such a state. If it is not, a fuzzy inference engine need to transfer the results of its computations to the fuzzy inference engine in the next layer.
- A source vertex in a FCG-SR is just an empty state, a starting point for the fuzzy composition. However in the new case, it may possibly indicate the state that has been reached by lower layers.

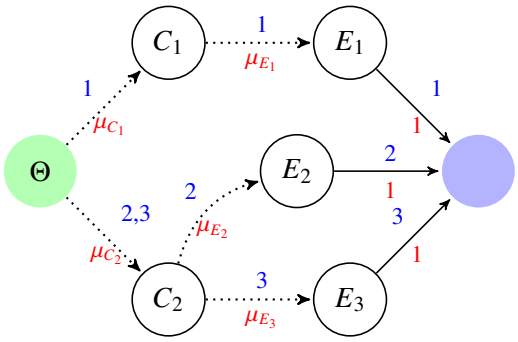
Figure 5.14 shows fuzzy composition graphs for the sub-rules that are produced as a result of two-layer decomposition of the combined rule from the previous sub-section, and for the decomposition, it is assumed that only c and e cannot be processed at the first layer. The steps required for fuzzy composition is similar to the steps described in Algorithm 6. The case with sub-rules only requires the data structures to be initialized with the proper values if the source vertex does not represent an empty state, and the selection of the maximum of computed firing strength values if the rule is a final rule; i.e., it has the consequent of the rule that it is decomposed from.



(a)



(b)



(c)

Figure 5.15: Fuzzy Composition Graph for sub-rules (2-layer decomposition with variables): (a) and (b) for the first layer, (c) for the second layer

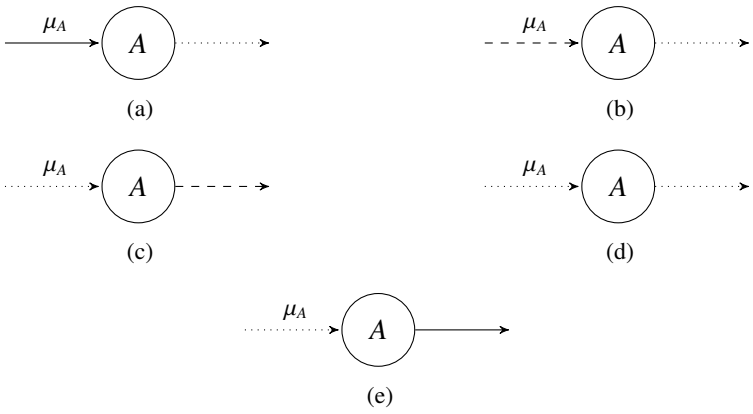


Figure 5.16: Possible edge-vertex-edge triplets for a FCG-CR with variables

5.5.2 Fuzzy Composition of Distributed Rules with Variables

The fuzzy composition process that has been described so far cannot be used for the computations required for rules with variables. All possible conjuncts need to be represented in a single fuzzy composition graph for a rule containing variables. If a conjunct in a sub-rule, which is a disjunction of predicates, is part of a bigger disjunctive clause in the non-decomposed rule, it is treated as an element of the variable for the sub-rule. However, for a rule with variables, the satisfaction of the rule does not necessitate the satisfaction of the elements of the variable. Therefore, in the fuzzy composition graph, the edges that have the fuzzy membership values of the elements of the variables as their weights should be traversed even if the weight is zero. For this reason, we define a new type of edge, a dotted edge, which connects one of the elements of a variable with other conjuncts, which may be other variable elements, or *P*-part or *R*-part of a rule.

Actually, there is not any change in the formal definition of the fuzzy composition graph. It is the same as the FCG-CR. Figure 5.15 shows examples of FCG-CRs for three sub-rules. Nevertheless, with the newly added edge type, how fuzzy inference engines process the FCG-CR needs to change. With the newly added edge type, in addition to the four edge-vertex-edge triplets shown in Figure 5.9, there may be five more triplets that can be present in a FCG-CR modeling a rule with variables. Figure 5.16 shows these five new triplets.

- Figure 5.16(a) shows a vertex representing a predicate in the *P*-part of the rule, and it forms a conjunct on its own.

- Figure 5.16(b) shows a vertex representing a predicate that is the last element of a disjunctive clause that is in the P-part of the rule.

- Figure 5.16(c) shows a vertex representing a predicate that is a component of a disjunctive clause, and this disjunctive clause is an entry of the vector that keeps variables.

- Figure 5.16(d) shows a vertex representing a predicate that is an entry of the variable vector on its own.

- Figure 5.16(e) shows a vertex representing a predicate that is the last entry of the variable vector.

Algorithm 7 shows how fuzzy inference engine computes the firing strength of a decomposed rule. In the case that there are no variables, that is, there are no dotted edges in a fuzzy composition graph, a fuzzy inference engine behaves exactly the same as in Algorithm 6. However, when the inference engine encounters a vertex that has an incoming edge with a dotted line, which model an element of a variable, it stores the evaluations related to the variables in a two dimensional array. The number of rules that are combined to form the rule, which is decomposed, constitutes one dimension of this array. The other dimension is related to the number of elements of the set q. For each path identifier, the elements of the set q is evaluated and stored in that array. After that point, unless the fuzzy inference engine does not encounter a straight-lined edge, it continues to store its results in that array. A straight edge after a dotted edge is possible either if the straight edge is connected to the sink vertex or it is connected to the R part of the rule. If the latter is the case, the sink vertex represents the satisfaction of the original, non-decomposed rule.

Figures 5.17 and 5.18 show an example for how the algorithm works for rules with variables. Starting with the source vertex, fuzzy inference engine first visits A_1, and as the outgoing edge of A_1 is straight, the array that keeps the latest value of the fuzzy composition completed up to that vertex is set to [0.6, 0.6, -]. As the traversed edge contains two path identifiers, two entries in the array are updated that correspond to the path identifiers. In the second step, A_2 is visited and the array is modified as [0.6, 0.6, 0.4]. After that, B_1 is visited, but as the traversed edge is a dotted edge, this implies that B_1 is an element of a variable. So, the weight associated with the traversed edge is stored in a new two-dimensional array: [[0.5, -, -]]. When B_2 is visited, the second array is modified as [[0.5, 0.2, 0.2]]. The next steps are

to visit D_1 and D_2, and in a similar manner to the previous case, the second array is updated to become [[0.5, 0.2, 0.2], [0.5, 0.5, 0.4]]. When the final state is reached in the first layer, the initial array is [0.6, 0.6, 0.4] and the array holding the results related to variable entries is [[0.5, 0.2, 0.2], [0.5, 0.5, 0.4]]. These values form the initial state of the computation done in the second layer. When C_1 is visited, the first element of the first vector in the variable array is compared with the weight of the traversed edge, a dotted edge, and the bigger one is assigned as the first entry of the first vector of the variable array. In a similar manner, when C_2 is visited, the second and third entries of the first vector of the variable array is updated: [[0.5, 0.2, 0.2], [0.5, 0.5, 0.4]]. Next E_1 is visited, and the first entry of the second vector of the variable array is assigned 0.6, since it is bigger than the previous value of 0.5. When E_2 is visited, the second entry of the same vector is set to 0.7, and when E_3 is visited, the previous value 0.4 is preserved because it is bigger than the weight associated with the traversed edge, 0.3. When the sink vertex is reached, the firing strength of the combined rule is calculated as following: For each entry of the initial array, which stores the results of computations for different path identifiers, this value is compared with the corresponding entries of the variable array. The minimum of these values determines the firing strength associated with the path identifier. After determining the values for each path identifier, which are min(0.6, 0.5, 0.6) = 0.5, min(0.6, 0.2, 0.7) = 0.2 and min(0.4, 0.2, 0.4) = 0.2, maximum of these give the final firing strength of the rule: 0.5.

Algorithm 7 Computing Firing Strength of a Combined Rule with Variables - Part 1

INPUT: $G = (V, E)$

OUTPUT: RS = firing strength of the rule

FUNCTION *void enqueue*(q, v): insert v to q

FUNCTION *vertex dequeue*(q): pull next entry from q and return it

FUNCTION *edge incoming_edge*(v, p): return incoming edge having label p

```
1:  enqueue(q,"source vertex")
2:  RS ← 0
3:  ∀i, m[i] ← 0
4:  ∀i, val[i] ← 1
5:  ∀i, ind[i] ← 1
6:  flag ← false
7:  while v ← dequeue(q) is not "sink vertex" do
8:      for all outgoing edge e_out of v do
9:          if v.next(e_out) is not in q then
10:             enqueue(q, v.next(e_out))
11:         end if
12:         if v is "source vertex" then
13:             if v is not an empty state then
14:                 Initialize val with lower layer's results
15:             end if
16:             Continue
17:         end if
18:         for all identifier p of e_out do
19:             if incoming_edge(v, p) is dotted then
20:                 flag ← true
21:             end if
22:             if flag is false then
23:                 if e_out.type is dashed then
24:                     m[p] ← max(incoming_edge(v, p).weight, m[p])
25:                 else if incoming_edge(v, p).type is dashed then
26:                     m[p] ← max(incoming_edge(v, p).weight, m[p])
27:                     val[p] = min(m[p], val[p])
```

Algorithm 8 Computing Firing Strength of a Combined Rule with Variables - Part 2

```
28:                 else
29:                     val[p] = min(incoming_edge(v, p).weight, val[p])
30:                 end if
31:             else                                              ▷ flag is true
32:                 if e_out.type is dashed then
33:                     var[ind[p]][p] ← max(incoming_edge(v, p).weight, var[ind[p]][p])
34:                 else if incoming_edge(v, p).type is dashed then
35:                     var[ind[p]][p] ← max(incoming_edge(v, p).weight, var[ind[p]][p])
36:                     ind[p] = ind[p] + 1
37:                     if e_out.type is not dotted then
38:                         flag ← false
39:                     end if
40:                 else if e_out.type is dotted then
41:                     var[ind[p]][p] ← max(incoming_edge(v, p).weight, var[ind[p]][p])
42:                     ind[p] = ind[p] + 1
43:                 else if v.next is not "sink vertex" then
44:                     flag ← false
45:                     var[ind[p]][p] ← max(incoming_edge(v, p).weight, var[ind[p]][p])
46:                 end if
47:             end if
48:         end for
49:     end for
50: end while
51: if rule is a final rule then
52:     for p = 1 to n do
53:         for i = 1 to ind[p] do
54:             val[p] = min(var[i][p], val[p])
55:         end for
56:         RS ← max(val[p], RS)
57:     end for
58: end if
```

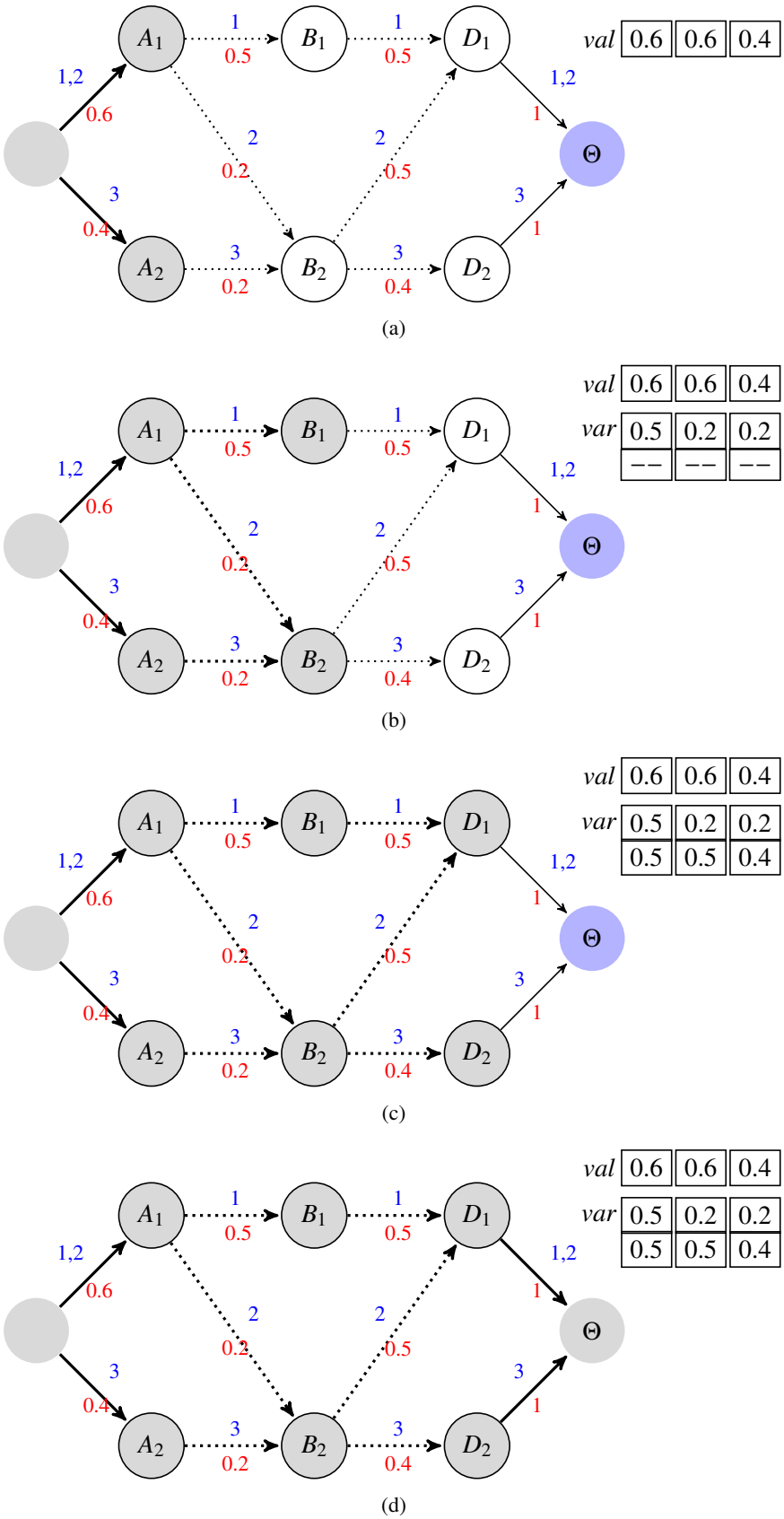


Figure 5.17: Fuzzy composition for a sub-rule in layer 1

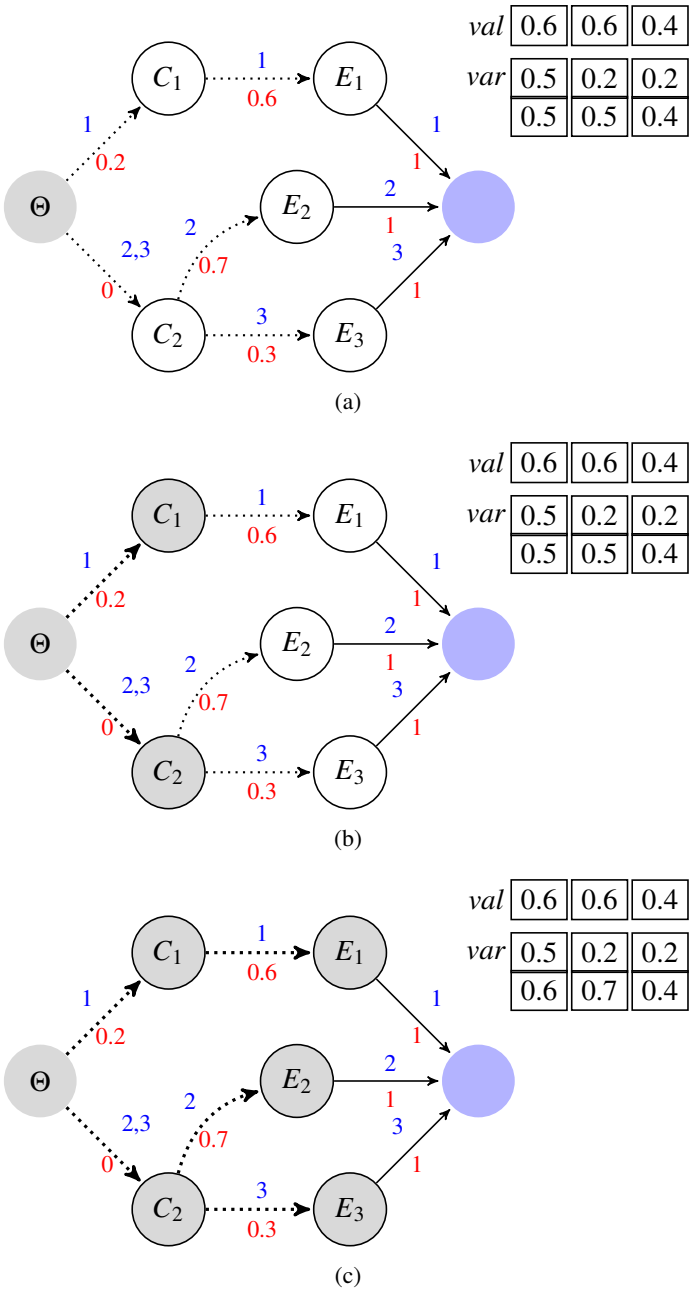
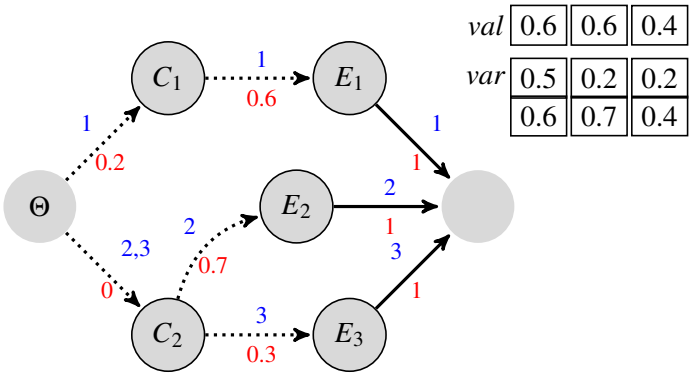


Figure 5.18: Fuzzy composition for a sub-rule in layer 2



RS = max(min(0.6,0.5,0.6),min(0.6,0.2,0.7),min(0.4,0.2,0.4))
RS = max(0.5,0.2,0.2) = 0.5

Figure 5.18: Fuzzy composition for a sub-rule in layer 2

CHAPTER 6

APPLICATION SCENARIO

In-network processing of data is especially useful for event-driven applications where the focus is on events and reactions to them. Although some applications might require all raw sensory data to be recorded, such as research-oriented applications where the aim is to extract knowledge about the inner workings of an unexplored real-world phenomenon, there are many application scenarios where the only interest is in the high-level knowledge of whether a certain event happens or not. Early detection of forest fires and healthcare monitoring are typical examples for such applications.

We choose to discuss a healthcare monitoring application as a scenario that can benefit from our approach. In the following subsections, we give the details of the properties and requirements of such an application and then describe a system architecture that can be used for the purpose.

6.1 Properties and Requirements

Healthcare monitoring has become an important application area for wireless sensor networks. The benefits of implementing such an application have been discussed in [9, 26]. Nevertheless, there is still a lot to be done in the development of hardware and software before we see widespread implementations. For example, individuals would expect wearable sensors to be small, unobtrusive, harmless, reliable and long-lasting. In addition to these hardware issues, the requirements of timely and efficient data processing necessitate improvements in the underlying information processing and communication architectures.

In a healthcare monitoring application, a person's physiological signals, such as pulse rate,

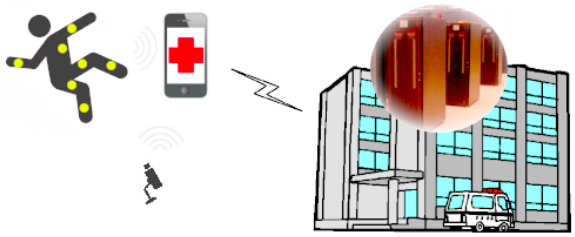


Figure 6.1: Healthcare monitoring network

blood pressure, respiration rate, body temperature, etc., together with physical activities and the state of individual and environmental conditions are assessed to detect and react to emergency cases. Take pulse rate as an example. A person's pulse rate while he is exercising might be twice its rate while he is resting. Besides, normal values of vital signs differ according to a person's age or sex. Because of this capacity for change, it is not enough to employ simple threshold-based filters to eliminate unnecessary network traffic. On the other hand, it is very important to relay only relevant information to the medical center as the person responsible for monitoring and managing alerts might be overwhelmed by the number of alerts and miss important ones.

Another aspect of healthcare monitoring is that it requires continuous evaluation of sensor values; an emergency can happen at any time and in any place. If in-network processing is not used, all sensor readings would need to be transported to the data center.

Finally, time is very critical and immediate reaction is required in the case of an emergency. Therefore, the delay between the detection of an event and the reaction to it needs to be small. In this respect, in-network processing again performs better than central processing.

6.2 System Architecture

The architecture of a healthcare monitoring application consists of a body area sensor network (BASN), a home network and a medical center network.

Body Area Sensor Network. This consists of wearable medical sensors that sense the physiological signals of a person and sensor nodes that detect the posture and movement and/or other relevant physical activities or characteristics of the person. Medical sensors can detect

pulse rate, blood pressure, respiration rate, body temperature, blood oxygen saturation and similar physiological signals. Additionally, physical motion sensors, such as the accelerometer and the gyroscope, are used to detect the current physical condition of the person [33]. An accelerometer is used to measure forward or upward acceleration so that it is possible to determine if the person is running, walking, falling down or stationary. A gyroscope measures orientation, such as sitting, lying or standing. Wearable sensor nodes have limited processing capabilities and they only check if the sensed value is above or below a threshold value.

Gateway Node. This involves sensor nodes that communicate with a PDA or a special device that is used to collect and process the nodes' readings, act as a gateway between the BASN, home network and the medical center network, and react to emergencies. The gateway node also has capabilities to interact with the person being monitored. For example, if the sensor outputs are not enough to reach an accurate conclusion, an audio-visual alarm or an application might be activated that demands a confirmation response from the person. The inputs provided or not provided by the person can be recognized by the rule-processing engine as events, which then lead to other actions. Sensor nodes communicate with that device using 802.15.4 or a similar low power and a low data-rate protocol.

Home Network. Apart from wearable sensors, temperature or light sensors, IP cameras placed in the home might be used to evaluate environmental conditions, which can help clarify a person's state. They communicate with the gateway node.

Medical Center Network. This contains a central server that stores and processes the information from individuals. Typical data that can be stored in central servers might be a person's medical history or results of a face-to-face examination. In addition to information processing systems, there are also operators who are responsible for monitoring and managing the incoming information. The gateway node in a home network communicates with the medical center network using GSM, UMTS or similar mobile technologies.

6.3 Healthcare Monitoring Rules

Rules that are used in monitoring the healthcare should be developed by domain experts. Although we are not domain experts, the following rules are given for illustration purpose. These rules are used to show how we distribute the processing so that not all data is collected

at a central place.

```
IF ((Blood_Oxygen_Saturation < 80) &
    (Blood_Pressure < 100/70) & (Respiration_Rate > 20))
THEN Send_Alert("Shock")

IF ((Pulse_Rate > 100) & (Blood_Pressure < 100/70) &
    ((Blood_Oxygen_Saturation < 80) |
     (Respiration_Rate < 15)))
THEN Send_Alert("Hearth Attack")

IF (Speed >= 6 km/h)
THEN Running

IF ((Pulse_Rate > 100) & !Running &
    (Respiration_Rate > 20))
THEN Send_Alert("Abnormal situation")

IF (Blood_Pressure > 120/80)
THEN Increment(High_Blood_Pressure_Count)

IF ((Blood_Pressure > 120/80) &
    (High_Blood_Pressure_Count > 5) &
    (High_Blood_Pressure_in_Family = true))
THEN Warn("See Doctor")
```

The above rules contain parts that can be processed at different places in the network and if we follow our decomposition algorithm we come up with several rule-bases for ordinary sensor nodes, a rule-base for the PDA/special device and a final one for central server at the medical center. Sensor nodes compare their measurements with the appropriate threshold values in order to detect events that might be interesting for the application. The following is a collection of rule-bases for different types of sensor nodes:

```
IF (Blood_Oxygen_Saturation < 80)
```

```
THEN low_oxygen_saturation

IF (Blood_Pressure < 100/70)
THEN low_blood_pressure
IF (Blood_Pressure > 120/80)
THEN high_blood_pressure

IF (Respiration_Rate > 20))
THEN high_respiration_rate
IF (Respiration_Rate < 15)
THEN low_respiration_rate

IF (Pulse_Rate > 100)
THEN fast_pulse_rate

IF (Speed >= 6 km/h)
THEN Running
```

Simple events from sensor nodes are gathered and fused in the PDA/special device and the rules used for this purpose are as follows:

```
IF (low_oxygen_saturation & low_blood_pressure &
    high_respiration_rate)
THEN Send_Alert("Shock")

IF (fast_pulse_rate & low_blood_pressure &
    (low_oxygen_saturation | (low_respiration_rate))
THEN Send_Alert("Hearth Attack")

IF (high_pulse_rate & !Running &
    high_respiration_rate)
THEN Send_Alert("Abnormal situation")
```

```
IF (high_blood_pressure)
THEN Increment(High_Blood_Pressure_Count)

IF (high_blood_pressure & High_Blood_Pressure_Count > 5)
THEN High_Blood_Pressure_Alarm
```

We assume that the information about whether there is high blood pressure problem in a family member is stored in a database in medical center network. The rule-base for the central server at the medical center is as follows:

```
IF (High_Blood_Pressure_Alarm &
    (High_Blood_Pressure_in_Family = true))
THEN Warn("See Doctor")
```

The above rule-bases make sure that the processing is distributed in the network and the data is transported only if there is an interest in it. Sensor readings are transported if they satisfy a filtering rule. Similarly, PDA eliminates false positives to be sent to the central server. For example, heart rate goes up while exercising and respiration rate decreases while sleeping, and these should be considered normal. Actually, for a typical person, we expect a large value for the ratio of these false positives to the real problems.

CHAPTER 7

PERFORMANCE EVALUATION

This chapter presents the experiments that are conducted to demonstrate how the proposed algorithms, RBDA and RBDA-V, behave with different parameters. Furthermore, simulations have been performed to show the amount of reduction in total transmitted packets and the energy consumption of the sensor nodes when employing the proposed approaches in contrast to employing a centralized approach.

In the following sub-section, we give the results of the experiments carried out to see the efficiency of the algorithms. After that, we explain the energy model that we adopt for this study and we discuss about the simulation setup and parameters. Finally in the last sub-section, we present the simulation results.

7.1 Performance of the Algorithms

We implemented the proposed rule decomposition algorithms in a UNIX environment using C language. In Figure 7.1 we see how our algorithms behave for a varying number of rules in the initial rule-base. For this experiment we generated random rule-bases having between 10 and 100 rules, for a two-layer decomposition. After repeating each experiment five times, each with different rule-bases, we plot the average of the sum of the number of all sub-rules for both layers. It is seen that the number of sub-rules generated by the RBDA-V is more predictable. The line is almost straight. Furthermore, the number of rules for a layer is close to the original number of rules. On the other hand, the number of the sub-rules generated by the RBDA depends entirely on the nature of the rules. Although random rule-bases have not generated the worst-case, the difference between the results is still huge, especially if we

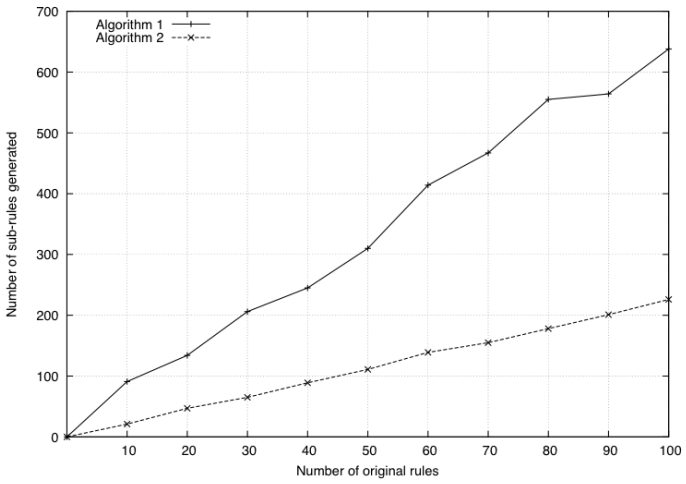


Figure 7.1: Number of initial rules vs. number of sub-rules generated

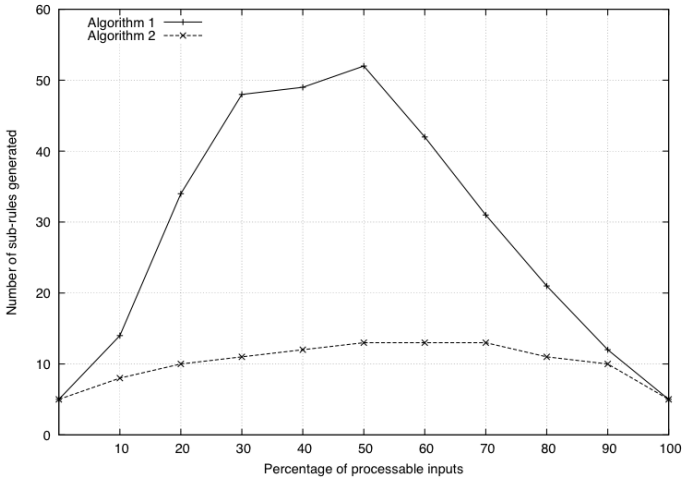


Figure 7.2: Percentage of processable inputs vs. number of sub-rules generated

consider the sensor nodes where energy conservation is of utmost importance.

In Figure 7.2, we present the effect of percent of processable inputs of a node to total available inputs on the number of sub-rules generated. For this purpose we used five different randomly generated rule-bases, each containing five rules, which may have at most five conjuncts and each of those conjuncts may contain at most four disjuncts. Throughout the experiments we varied the input set of the node for each percentage value and took the average of the results for plotting. A percent of 0 means no processable inputs are present and 100 means all inputs can be processed. In these cases the original number of rules is preserved. We can see from the results that there is a major difference in the number of sub-rules generated by the RBDA-V

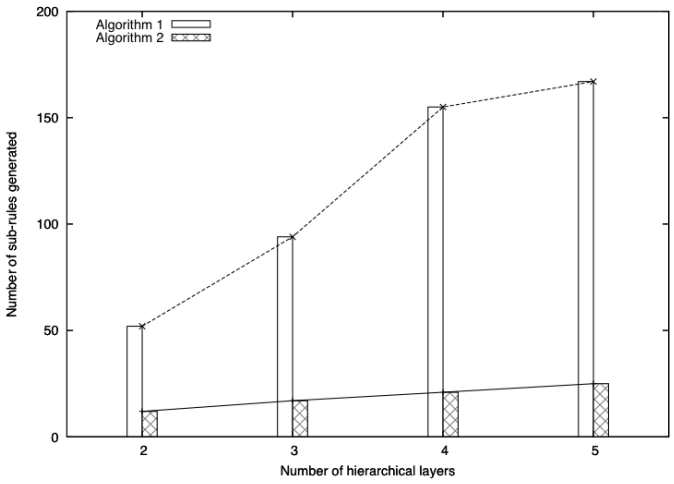


Figure 7.3: Number of different layers vs. number of sub-rules generated

and the RBDA when the percentage is between 20 and 70.

In Figure 7.3, we see the effect of the number of different layers, which is equal to the number of different classifications, on the number of sub-rules generated. We used the same rules that we used in the second experiment, and we uniformly distributed inputs to the layers. The results lead us to a similar conclusion as the previous experiment, where, with everything being equal, the RBDA-V produces fewer rules, and compared to the RBDA, the number of layers has less influence and the number of sub-rules generated is more predictable.

In Figures 7.4 and 7.5, the ratio of the reduction in the number of rules for different percentages of combinable rules is seen. For this experiment, starting with a 15 non-combinable randomly generated fuzzy rules, a new rule is added into the set of rules that can be combined with one of the initial rules so that the percentage of the combinable rules is increased. The ratio of the number of fuzzy rules when nothing is performed to the number of rules when rules having the same event pattern and consequent is combined into a single rule is plotted in the figures. The rule-base is decomposed into two sub-rule-bases. In Figure 7.4, the ratio for the rules of layer one and in Figure 7.5, the ratio for layer two are seen. As it can be seen from the figures, in the presence of 50% or more combinable rules, for the RBDA-V, the number of rules is decreased by nearly a factor of 2 by combining fuzzy rules that have the similar event pattern and the same output. This value increases to a factor of 9 for the RBDA.

Figure 7.6 shows the number of bytes transferred when an event is occurred. For this experi-

Table 7.1: Space needed by rule-bases

	RB-1 (bytes)	*RB-2 (bytes)*
RBDA	47	43
RBDA-V	17	10

ment, we use a set of rules with varying number of $|q|$, the size of the set q. These rules have the same event pattern and the same consequents, so they can be combined. While changing the number of rules, the ratio of the number of bytes transferred when they are not combined to the number of bytes transferred when they are combined is plotted in the figure. It is obvious from the figure that combining fuzzy rules decreases the number of bytes that are transferred. Although the size of data transmitted in a single packet is increased for a combined rule, as the number of packets declines, less packet header overhead results in less number of bytes transmitted. In the case that the size of the set q increases, so does the size of the payload. Therefore, the ratio decreases when $|q|$ increases.

We used the rule-bases of the last two experiments for evaluating the storage requirements of the algorithms. In order to calculate how much space is needed by each algorithm, we made the following assumptions. First, we assume that the predicates in the rules contain a pointer to the actual implementation of the predicate function. This knowledge is stored in two bytes (16 bits). Similar to predicates in the antecedent, we assume that the consequent part of the rules contains a pointer, two bytes, to the actual code that takes the necessary actions. A PMV requires as much space as the total space required by each predicate that could be matched by that variable. Finally, a KTV requires just two bytes, since a single bit is enough to represent the state of the element, and 16 bits is more than enough for a rule. In Table 7.1, *RB*-1 and *RB*-2 refer to the sub-rule-bases for layer 1 and layer 2 accordingly. The numbers seen in the table are the average of the number of bytes required for sub-rule-bases that are generated from the five original rule-bases. The RBDA requires 90 bytes in total whereas the RBDA-V requires 27 bytes in total, which is 30 percent of the space required for the RBDA.

7.2 Simulations for In-Network Processing

In this section, we first discuss our energy consumption model for a WSN, then present the network and application setup of our simulations, and finally show and discuss our simulation

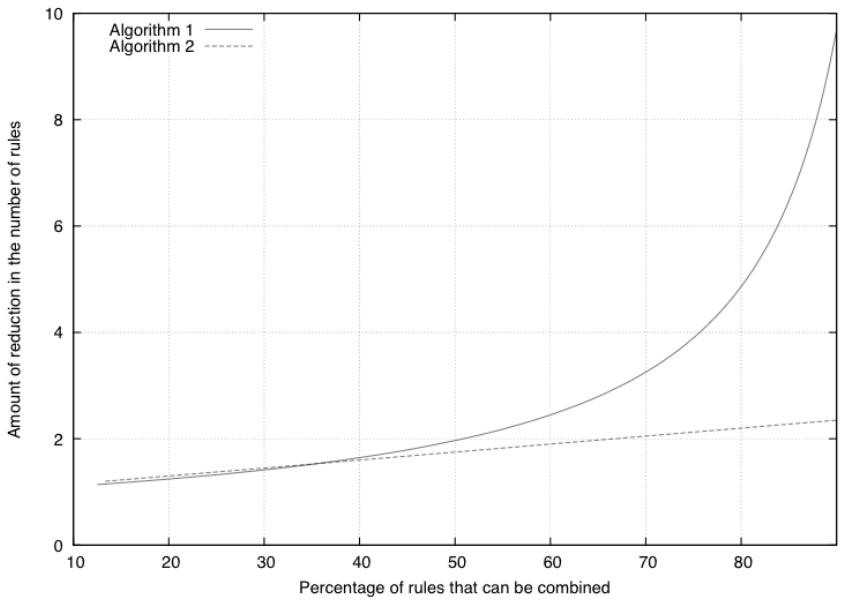


Figure 7.4: Ratio of reduction in the number of rules when fuzzy rule combining performed - Layer 1

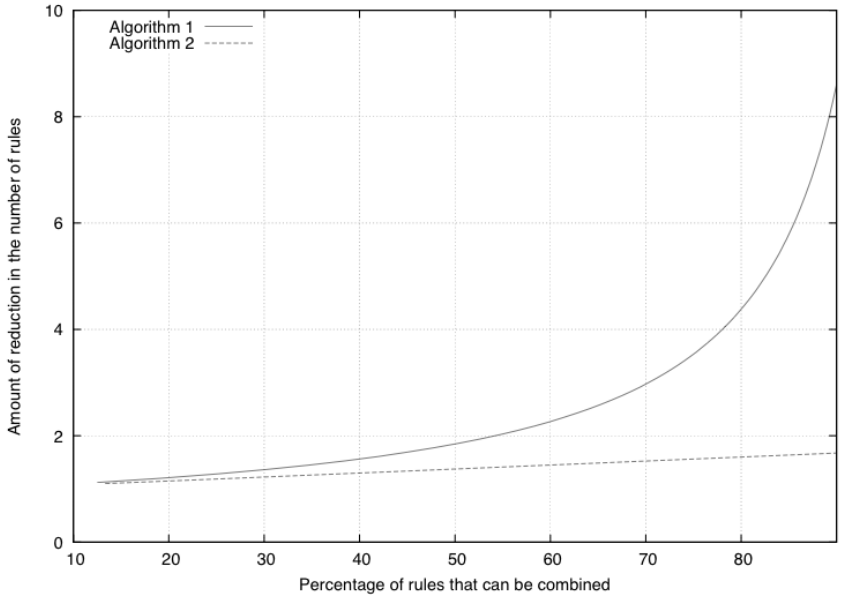


Figure 7.5: Ratio of reduction in the number of rules when fuzzy rule combining performed - Layer 2

results for evaluating our approach against centralized approach. In our simulations we used Castalia simulator [12], which is a sensor platform independent WSN simulator based on OMNet++ platform.

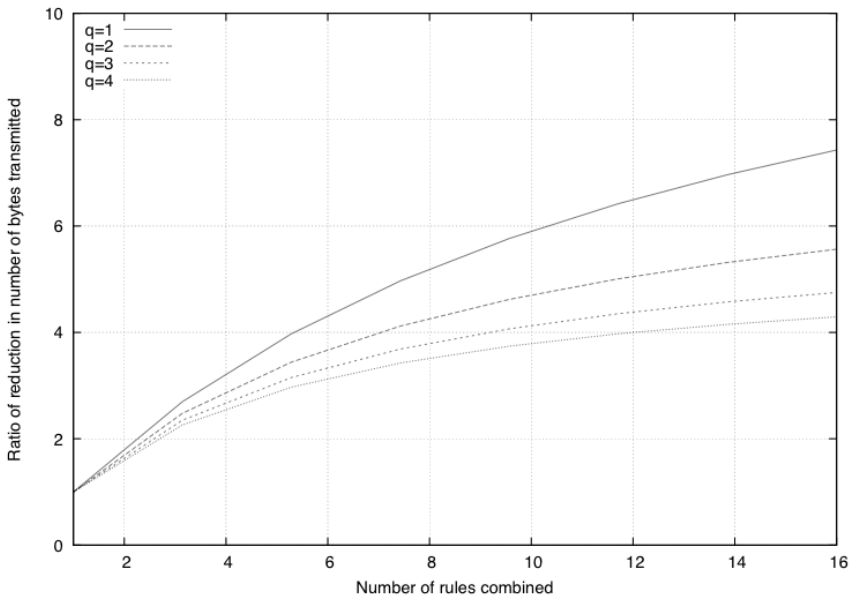


Figure 7.6: Ratio of reduction in the number of bytes transmitted for rules with varying number of size of the set q

7.2.1 Energy Model

We consider sensor network with n sensor nodes, where the number of events detected by a sensor node has a poisson distribution with a mean event arrival rate of λ. Let h be the average number of hops for a node to reach the nearest cluster-head, b be the number of bits in a network packet and c be the transmission rate in bits per second.

In a time interval $[0, t)$, λt events occur at a single node and $n\lambda t$ events occur throughout the entire network. The occurrence of a single event results in h transmission and $(h - 1)$ receive operations that are carried out by the sensor nodes. In order to calculate the energy consumed for these communication operations, we can use the formula

$$E = V \cdot I \cdot T, \tag{7.1}$$

where V is the potential difference, I is the electrical current and T is the time in seconds. In [57], the electrical current that flows through the Mica2's radio circuitry is listed under the presence of a 3V power supply. The receive operation results in 7.0 mA current flow whereas the transmission operation causes between 3.7 to 21.5 mA, depending on the power level used for the transmission. We can calculate the time required to transmit or receive a packet using the formula $T = b/c$.

Table 7.2: Power required for sensor node's operations

	Power (mW)
Receive	22.2
Idle	22.2
Transmit	80.1 - 15.9
Sense	0.02
Sleep	0.0006

The total energy consumed by the sensor nodes is determined by multiplying the energy required for the transmission and receive operations when an event occurs by the total number of events that occur in the sensor network:

$$E = \frac{3.n.\lambda.t.b}{c} \left(h \cdot I_t + (h-1) \cdot I_r\right), \tag{7.2}$$

where I_t and I_r are the electrical currents that flow through the radio circuitry when transmitting and receiving packets, respectively.

In the above formula, the energy consumption that occurs while sensor nodes listen to the wireless channel is not taken into account. However, sensor nodes also need to power the radio electronics in a listening or idle state to detect the presence of radio signals. As mentioned in [62], the energy consumed during the listening state is big enough so as not to be ignored in the energy consumption analysis. Actually, we see from [57] that the same amount of current flows through the radio circuitry during the listen and receive operations.

If we assume that the same amount of energy is consumed while receiving and listening, then the total energy consumed by the sensor nodes will be:

$$E = 3.t. \left(\frac{n.\lambda.b.h.I_t}{c} + l.I_r \right), \tag{7.3}$$

where l is the ratio of listen time to the sum of sleep and listen times in a unit time interval. We choose to use this energy model in our simulations where we ignore the energy consumed during processing and sensing operations.

7.2.2 Network Setup

In our setup, the sensor network consists of ordinary sensor nodes and highly capable cluster-heads. Sensor nodes use cluster-heads for the transmission of their data to the sink and it

is assumed that they can reach the cluster-heads in just one hop. They choose the nearest cluster-head as their next hop to sink. Similarly, cluster-heads reach the sink in one hop. This means that, any node in the sensor network can reach the sink in at most two hops.

The sensor field used in the simulations is 400m x 400m and it is divided into 16 100m x 100m sub-regions. Each sub-region contains one cluster-head randomly deployed inside it. Simulations were run for 50, 100, 150 and 200 sensor nodes that are deployed uniformly in the field.

Table 7.2 shows the amount of power required for the operations carried out by Mica2 motes. As it can be seen from the table, a node in receive and idle states requires the same amount of power.

7.2.3 Application Setup

Rules describing the application logic for this experiment are as follows:

$$(x \geq 50) \rightarrow A_1$$

$$((\textit{last } 5 \textit{ readings of } y \geq 100) \textit{ and } (v \geq 500)) \rightarrow A_2$$

$$((x \geq 35) \textit{ and } (y \geq 50)) \rightarrow A_3$$

$$((x \leq 0) \textit{ and } (z \leq 0) \textit{ and } (w == 0)) \rightarrow A_4$$

$$((x \leq 0) \textit{ and } (z \leq 0) \textit{ and } (w == 1)) \rightarrow A_5$$

$$((y \geq 125) \textit{ and } ((w == 2) \textit{ or } (v < 400))) \rightarrow A_6$$

These rules form the central rule-base. Ordinary sensor nodes sense x, y and z whereas cluster-heads sense v and w. The rules do not have any conditions.

We consider three different scenarios. In the first scenario, the sensory data is transmitted directly to the sink without any processing by the sensor nodes. Cluster-heads act as gateways and they do not process data either. In order to reduce the total packet overhead and the number of packets, the x, y and z values are packed into a single packet. In the second scenario, both sensor nodes and cluster-heads send data to the sink only if it is above or below some threshold value. This is similar to the case in directed diffusion [30], where data is sent if there is an interest in it. Sensory data meeting the threshold conditions are transmitted in a single network packet in this scenario too. Finally, in the last scenario, sensor nodes and cluster-heads process data according to the rules in their rule-bases and only the results of the

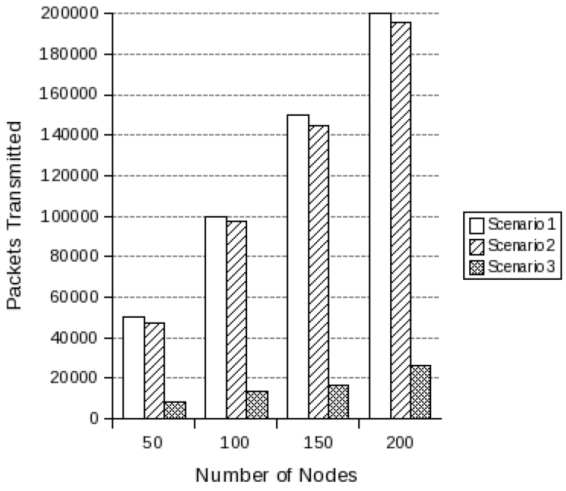


Figure 7.7: Total number of packets transmitted by sensor nodes

processing are sent to the sink.

7.2.4 Simulation Results

Simulations were run for 5000 seconds for each of the application scenarios under exactly the same conditions; that is, the same network topology, the same energy and the physical event models were used in all of the application scenarios. Figures 7.7 and 7.8 show the simulation results.

Figure 7.7 shows the total number of packets sent by the sensor nodes. The results in the figures do not include the packets transmitted by cluster-heads. As it can be seen from the figure, the total amount of communication is drastically reduced by employing our approach, used in the third scenario. In addition to the decrease in the number of packet transmissions, the packet size for the third scenario is smaller compared to the packet sizes for the other two scenarios. In the first two scenarios, raw sensory data, together with its temporal and spatial properties, are transported inside the network; in the third scenario only the current state of the processing is propagated to the appropriate nodes. An eight-bit data can represent 2^8 different states. Additionally, due to spatial and temporal locality properties of information processing in WSNs, spatial and temporal information is either not transported or is transported in a short path in the network. Therefore, the reduction in the total number of bytes transmitted is much lower than the reduction in the total number of packets. In the simulations we assumed that

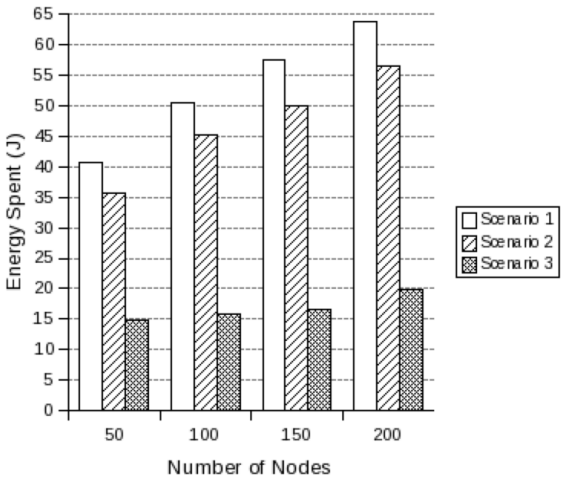


Figure 7.8: Average energy consumption by sensor nodes

no collisions occur in the network. Had collisions been taken into account, the results for the first and second applications would have been worse due to the fact that more network traffic would lead to higher probabilities of packet collisions and retransmissions.

Figure 7.8 shows the average amount of energy consumed by each sensor node. As seen from the figure, the total spent energy is decreased by nearly a factor of three when the data is processed inside the network using our approach. However, the number of transmitted packets is reduced by nearly a factor of seven. The nodes' energy consumption is not a corresponding seven times lower because they use the same amount of energy in the idle state as in the receive state.

In all of the scenarios, only one-hop clusters exist and cluster-heads reach the sink in just one hop. In the case of multi-hop clusters, we anticipate better results with in-network processing compared to other approaches, because in such a case, nodes need to route other nodes' packets as well. The more network packets put into the sensor network, the more a node needs to transmit the packets of others. In addition to the obvious implications of power consumption, more packets cause more collisions and more retransmissions.

7.2.5 Healthcare Monitoring

In addition to the above simulations, we also conducted experiments for the healthcare monitoring application described in Chapter 6 to show that in-network processing is indispensable

for event-driven applications. Here, the concern is not only minimizing network traffic and power consumption, but also generating timely and accurate reactions to important events.

We used the following rules:

$$(not\_exercising \ \& \ (pr \leftarrow high \mid rr \leftarrow high \mid bp \leftarrow high)) \rightarrow \text{"}record\_abnormal\_event\text{"}$$
$$(exercising \ \& \ (pr \leftarrow low \mid rr \leftarrow low \mid bp \leftarrow low)) \rightarrow \text{"}record\_abnormal\_event\text{"}$$
$$(number\_of\_abnormal\_events \geq 3) \rightarrow \text{"}inform\_person\_and\_medical\_personnel\text{"}$$
$$(pr \leftarrow very\ low \mid rr \leftarrow very\ low) \rightarrow \text{"}inform\_medical\_personnel\text{"}$$
$$(pr \leftarrow high \ \& \ bp \leftarrow low) \rightarrow \text{"}ask\_person\_if\_he\_is\_OK\text{"}$$
$$(not\_OK\_signal \mid no\_response\_for\_active\_question) \rightarrow \text{"}inform\_medical\_personnel\text{"},$$

where pr is pulse rate, rr is respiration rate, and bp is systolic blood pressure. The first four rules are self-explanatory. If the fifth rule fires, an application that gets the person's attention by audio-visual stimuli and asks him to provide his status is activated. That person is assumed to provide the necessary information if he is all right. Rule 6 covers the cases where there are no replies after rule has requested status information, or there is an explicit notification of a bad condition.

Wearable sensor nodes have a sampling interval of 30 seconds and the simulation duration is 86400 seconds, i.e., 1 day. Similar to the previous simulation, we consider both central and distributed processing. We consider three scenarios. In the first scenario, most of the time a person has normal values as his vital signs, but occasionally his pulse rate increases while his blood pressure has a low value. In the second scenario, a person has high blood pressure values throughout the day. Finally, in the third scenario, a person having normal vital signs stops breathing for two minutes.

The simulation results can be seen in Table 7.3. Link 1 refers to communication between the sensor nodes and the gateway node, and Link 2 is for the communication between the gateway and the central network. The results confirm the expectations that in-network processing makes a big difference. Instead of transmitting thousands of event notifications to a central network, only necessary events are transmitted.

Another parameter to look for is the delay in reacting to emergency cases. If the fifth rule is evaluated at a centralized system and it fires, the reaction requires getting back to home network and activating an application that interacts with the person. Table 7.4 shows how many hops are required for such a rule to reach a conclusion that the situation requires emergency

Table 7.3: Number of packet transmissions for healthcare monitoring

Scenario	Centralized		Distributed	
	Link 1	Link 2	Link 1	Link 2
Case 1	11520	2896	2862	3
Case 2	11520	3926	3025	173
Case 3	11520	2880	2508	4

Table 7.4: Number of hops required for a decision

Location		*Number of hops*	
Firing	*Action*	*Centralized*	*Distributed*
Center	Center	2	2
Center	Gateway	4	3
Gateway	Center	2	2
Gateway	Gateway	4	1

personnel's response. It is wise to support decisions reached by bare sensor readings with a confirmation response from people in order to reduce false alarms. The case in the fourth row in the table would be frequently encountered in such a scenario. That row indicates that a decision can be made at the gateway node, but with central processing data need to be transported to central network. Furthermore, action, i.e., soliciting a confirmation request from the user, can only be taken at the gateway node. For central processing, transferring data from the sensor nodes to the gateway node, from the gateway node to the central system, from the central system back to the gateway (user confirmation request), and finally from the gateway to the central system (user's response) sums up to four hops. On the other hand, if distributed processing is allowed, sensor readings are transferred only to the gateway, and the rest of the processing is carried out at this node.

CHAPTER 8

CONCLUSIONS AND FUTURE WORK

In this thesis we present new methods that distribute information processing into different nodes in a WSN for rule-based event-driven applications. We present how we decompose the central rule-base expressing the logic of an application into multiple sub-rule-bases that are employed in appropriate places inside the network. Our first method shows a simple, easy-to-grasp and rational way of distributing information processing. However, it possibly suffers from an exponential increase in the number of rules generated. Therefore, we introduce a new way of representing rules with variables and rearrange the distribution method accordingly. We show the results of experiments that confirm that the second method removes the deficiencies of the first method.

Concerning the information processing in sensor networks, imprecision of sensor readings is an important challenge. We think that a fuzzy rule-based system is the most suitable solution to tackle this problem. However, the number of rules in a fuzzy rule-based system is the most important obstacle for it to be employed in WSNs. For this reason, we present a mechanism that reduces the number of rules in a fuzzy rule-based system, and show how fuzzy inference can be done in a distributed manner in line with our previous rule distribution schemes. Based on the rule reduction strategy, we propose a new algorithm for fuzzy composition that can be used by fuzzy rule engines.

Our motivation was to process data inside WSNs as much as possible since communication operations are responsible for most of the energy consumption in sensor nodes. Our simulations show that in a sensor network application scenario employing in-network processing based on our methods, the average energy consumption of a sensor node might be reduced to nearly one-third of the energy consumption that is used in central processing. We describe

an application scenario that might benefit from our approach and simulations based on this scenario prove the benefits of using in-network processing.

Since the rule-base decomposition is carried out outside of sensor networks and on rare occasions like initial deployment or when there is a need for changing the logic of an application, its execution time is less important compared to the space requirements of sub-rule-bases. Although technological advancements make storage capacity being a smaller concern, it is still a limitation for small sensor nodes. Therefore, there is a need for having lightweight representations of rules to be employed in sensor nodes. This can be a topic for future study.

Cooperation of sensor nodes may provide more reliable decisions. Of course, cooperative processing may explicitly be specified by rules. But in order to mitigate the complexity introduced by such rules, developing systems that do this job implicitly might be an area that can be worked on in future studies. Fuzzy logic seems to be a flexible and cheap solution for combining the data of multiple sensor nodes.

The truth value of a predicate might change over time. For example, a temperature value of 30° Celcius measured may be considered normal in summer but it should not be counted as a normal value if measured in winter. In addition to that, the parameters associated with rule-based systems, such as fuzzy membership functions or the priorities of rules, or the parameters associated with sensor nodes, such as sleeping interval or transmission range, may need to be changed in the course of time. In order to adapt to such changes automatically, systems that have learning capabilities, such as neuro-fuzzy systems, may be used. This may be a study that can extend the work presented here.

In rule-bases that contain automatically generated rules, which are the rules generated as a result of decomposition, there might be redundant rules. The determination and elimination of such rules should be considered in a future study as well.

REFERENCES

[1] S. Ahn, and D. Kim, *"Proactive Context-Aware Sensor Networks"*, Lecture Notes in Computer Science, Vol. 3868/2006, pp. 38-53, 2006.

[2] I. F. Akyildiz, S. Weilian, Y. Sankarasubramaniam, and E. Cayirci, *"A Survey on Sensor Networks"*, IEEE Communications Magazine, Vol. 40, No. 8, pp. 102-114, August 2002.

[3] J. F. Allen, *"Maintaining Knowledge about Temporal Intervals"*, Communications of the ACM, Vol. 26, No. 11, pp. 832-843, November 1983.

[4] Arcsight Enterprise Security Manager (ESM), http://www.arcsight.com/products/products-esm/, Last Visited June 2011.

[5] F. Babich, L. Deotto, *"Formal Methods for Specification and Analysis of Communication Protocols,"* IEEE Communications Surveys and Tutorials, Vol. 4, No. 1, Dec 2002.

[6] J. Balasubramaniam, *"Rule Reduction for Efficient Inferencing in Similarity Based Reasoning"*, International Journal of Approximate Reasoning, Vol. 48, Issue 1, pp.156-173, 2008.

[7] B. Berstel, P. Bonnard, F. Bry, M. Eckert, and P. L. Patranjan, *"Reactive Rules on the Web"*, Reasoning Web, Lecture Notes in Computer Science, Vol. 4636, pp. 183-239, 2007.

[8] M. Bhardwaj, and A. Chandrakasan, *"Bounding the Lifetime of Sensor Networks via Optimal Role Assignment"*, In Proc. of 21st. Annual Joint Conference of the IEEE Computer and Communication Socities, New York, NY, USA, 2002.

[9] P. Bonato, *"Advances in Wearable Technology and its Medical Applications"*, Proc. of 32nd. Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), Buenos Aires, Argentina, September 2010.

[10] B. Bonfils, and P. Bonnet, *"Adaptive and Decentralized Operator Placement for In-Network Query Processing"*, IPSN'03, Palo Alto, CA, USA, April 2003.

[11] BTnode Platform. Available online at http://www.btnode.ethz.ch/, Last Visited June 2011.

[12] Castalia: A simulator for WSNs. Available online at http://castalia.npc.nicta.com.au/, Last Visited June 2011.

[13] Chipcon. CC1000 - Single Chip Very Low Power RF Transceiver. Chipcon AS, Oslo, Norway, April 2002.

[14] Chipcon AS SmartRF. CC2420 2.4 GHz RF Transceiver CC2420 - Zigbee RF Transceiver. Oslo, Norway, April 2004.

[15] C. Collet, and T. Coupaye, *"Primitive and Composite Events in NAOS"*, Actes des 12e Journées Bases de Données Avancées, pp. 331-349, Cassis (France), August 1996.

[16] K. Dasgupta, M. Kukreja, and K. Kalpakis, *"Topology-Aware Placement and Role Assignment for Energy-Efficient Information Gathering in Sensor Networks"*, In Proc. if the eight IEEE International Symposium on Computers and Communication (ISCC'03), Antalya, Turkey, July 2003.

[17] A. P. Dempster, *"A generalization of Bayesian Inference"*, Journal of the Royal Statistical Society, Vol. 30, No. 2, 1968.

[18] M. Eckert, *"Complex Event Processing with XChange^{EQ}: Language Design, Formal Semantics, and Incremental Evaluation for Querying Events"*, Ph.D. Dissertation, October 2008.

[19] C. Frank, and K. Romer, *"Algorithms for Generic Role Assignment in Wireless Sensor Networks"*, ACM International Conference on Embedded Networked Sensor Systems (SenSys'05), San Diego, CA, USA, November 2005.

[20] C. Frank, and K. Romer, *"Solving Generic Role Assignment Exactly"*, WP-DRTS/IPDPS, Rhodes Island, Greece, April 2006.

[21] Fuzzy Logic Fundamentals, Available online at http://ptgmedia.pearsoncmg.com/images/0135705991/samplechapter/0135705991.pdf, Last Visited June 2011.

[22] S. Gatziu, K. R. Dittrich, *"Events in an Active Object Oriented Database System"*, In Proc. of 1st. International Workshop on Rules in Database Systems, pp. 23-39, Edinburg, Scotland, September 1993.

[23] S. Gatziu, K. R. Dittrich, *"Detecting Composite Events in Active Databases Using Petri Nets,"* Proceedings of the Fourth International Workshop on Research Issues in Data engineering: Active Database Systems, pp. 2-9, Feb 1994.

[24] N. H. Gehani, H. V. Jagadish, and O. Shmueli, *"Composite Event Specification in Active Databases: Model and Implementation"*, In Proc. of the 18th. International Conference on Very Large Databases, 1992.

[25] J. Hajek, *"Basic Fuzzy Logic and BL-Algebras"*, Soft Computing - A Fusion of Foundations, Methodologies and Applications, Vol. 2, No. 3, pp. 124-128, 1998.

[26] M.A. Hanson, H.C. Powell, A.T. Barth, K. Ringgenberg, B.H. Calhoun, J.H. Aylor, J. Lach, *"Body Area Sensor Networks: Challenges and Opportunities"*, Computer, Vol.42, No.1, pp. 58-65, 2009.

[27] M. Hempstead, M. J. Lyons, D. Brooks, G. Wei, *"Survey of Hardware Systems for Wireless Sensor Networks"*, ASP Journal of Low Power Electronics, Vol. 4, No. 1, 1-10, April 2008.

[28] IEEE 802.15 WPAN Task Group 4. http://www.ieee802.org/15/pub/TG4.html, Last Visited June 2011.

[29] Infineon Technology AG, Munich, Germany, Energy Efficient Sensor Networks: eyesIFX Wireless Sensor Network Deployment Kit, 1.0.1 edition, November 2005.

[30] C. Intanagonwiwat, R. Govindan, and D. Estrin, *"Directed Diffusion: a scalable and robust communication paradigm"*, Proceedings of the 6th. Annual International Conference on Mobile Computing and Networking, pp. 56-67, Boston, 2000.

[31] B. Jiao, S. Son, and J. Stankovic, *"GEM: Generic Event Service Middleware for Wireless Sensor Networks"*, Proceedings of the 2nd International Workshop on Networked Sensing Systems (INSS), June 2005.

[32] K. Kapitanova, S. H. Son, K.-D. Kang, *"Event Detection in Wireless Sensor Networks - Can Fuzzy Values Be Accurate?"*, Ad Hoc Networks - Second International Conference, ADHOCNETS 2010, Victoria, BC, Canada, August 2010.

[33] A. M. Khan, Y.-K. Lee, S. Y. Lee, T.-S. Kim, *"A Triaxial Accelerometer-Based Physical-Activity Recognition via Augmented-Signal Features and a Hierarchical Recognizer"*, IEEE Transactions on Information Technology in Biomedicine, Vol. 14, No. 5, pp. 1166-1172, Sep 2010.

[34] M. Kochhal, L. Schwiebert, and S. Gupta, *"Role-based Hierarchical Self Organization for Wireless Ad Hoc Sensor Networks"*, WCNA'03, San Diego, CA, USA, September 2003.

[35] B. B.-Korpeoglu, A. Yazici, I. Korpeoglu, R. George, *"A New Approach for Information Processing in Wireless Sensor Networks"*, 22nd International Conference on Data Engineering Workshops (ICDEW'06), Atlanta, GA, 2006.

[36] B. Krishnamachari, D. Estrin, and S. Wicker, *"Modeling Data-Centric Routing in Wireless Sensor Networks"*, In. Proc. of IEEE INFOCOM'02, 2002.

[37] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, *"TinyDB: an Acquisitional Query Processing System for Sensor Networks"*, ACM Trans. Database Syst., Vol. 30, No. 1, pp. 122-173, 2005.

[38] E. H. Mamdani, and S. Assilian, *"An experiment in linguistic synthesis with a fuzzy logic controller"*, International Journal of Man-Machine Studies, Vol. 7, No. 1, pp. 1-13, 1975.

[39] G. P. Meo, and S. Ceri, *"Composite Events in Chimera"*, In Proc. of 5th. International Conference on Extending Database Technology (EDBT'96), LNCS 1057, pp. 56-78, 1996.

[40] Hardware platforms available for TinyOS. Available online at http://www.tinyos.net/scoop/special/hardware.html, Last Visited June 2011.

[41] I. Motakis, and C. Zaniolo, *"Formal Semantics for Composite Temporal Events in Active Database Rules"*, Journal of System Integration, Vol. 7, No. 3-4, pp. 291-325, 1997.

[42] T. Murata, *"Petri Nets: Properties, Analysis and Applications"*, Proceedings of IEEE, Vol. 77, No. 4, pp. 541-580, 1989.

[43] L. Nachman, R. King, R. Adler, J. Huang, V. Hummel, *"The Intel®Mote Platform: a Bluetooth-based sensor network for industrial monitoring"*, Proceedings of 4th International Symposium on Information Processing in Sensor Networks, 2005.

[44] E. F. Nakamura, A.A.F. Loureiro, A. C. Frery, *"Information Fusion for Wireless Sensor Networks: Methods, Models, and Classifications"* ACM Computing Surveys, Vol. 39, No. 3, Article 9, August 2007.

[45] Nordic VLSI ASA. nRF2401 Single Chip 2.4 GHz Radio Transceiver. Tiller, Norway, March 2003.

[46] A. Paschke, and A. Kozlenkov, *"Rule-Based Event Processing and Reaction Rules"*, Lecture Notes in Computer Science (LNCS), Vol. 5858/2009, pp. 53-66, 2009.

[47] M. M. Perianu, and P. Havinga, *"D-FLER: Distributed Fuzzy Logic Engine for Rule-Based Wireless Sensor Networks"*, Ubiquitous Computing Systems, Vol. 4836/2007, pp. 86-101, 2007.

[48] P. R. Pietzuch, B. Shand, and J. Bacon, *"Composite Event Detection as a Generic Middleware Extension"*, IEEE Network Magazine, Special Issue on Middleware Technologies for Future Communication Networks, Jan/Feb 2004.

[49] G. J. Pottie, and W. J. Kaiser, *"Wireless Integrated Network Sensors"*, Communications of the ACM, Vol. 43, No. 5, pp. 51-58, 2000.

[50] RFM Monolithics, Inc., Dallas, TX, USA. 916.50 MHz Hybrid Transceiver, 1999.

[51] K. Romer, and F. Mattern, *"The Design Space of Wireless Sensor Networks"*, IEEE Wireless Communications, Vol. 11, No. 6, pp 54-61, December 2004.

[52] RSA enVision, http://www.rsa.com/node.aspx?id=3170, Last Visited June 2011.

[53] N. Sadagopan, B. Krishnamachari, A. Helmy, *"The ACQUIRE Mechanism for Efficient Querying in Sensor Networks"*, Proceedings of the First IEEE International Workshop on Sensor Network Protocols and Applications, pp. 149-155, May 2003.

[54] G. Shafer, *"A mathematical theory of evidence"*, Princeton University Press, 1976.

[55] Shimmer Hardware Guide. Available online at http://www.eecs.harvard.edu/ konrad/projects/shimmer/references/SHIMMER_HWGuide_REV1P3.pdf, Last Visited June 2011.

[56] Y. Shi, *"A Deep Study of Fuzzy Implications"*, Ph.D. Dissertation, September 2009.

[57] V. Shnayder, M. Hempstead, B. R. Chen, G. W. Allen, and M. Welsh, *"Simulating the power consumption of large-scale sensor network applications"*, Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems, pp. 188-200, 2004.

[58] T. Takagi, M. Sugeno, *"Fuzzy identification of systems and its application to modeling and control"*, IEEE Transaction on Systems, Man and Cybernetics, Vol. 15, pp. 116-132, 1985.

[59] M. Sugeno, and G. T. Kang, *"Structure identification of fuzzy model"*, Fuzzy Sets and Systems, Vol. 28, pp. 15-33, 1988.

[60] K. Terfloth, *"A Rule-Based Programming Model for Wireless Sensor Networks"*, Ph.D. Dissertation, June 2009.

[61] N. Tezcan, and W. Wang, *"A lightweight classification algorithm for energy conservation in wireless sensor networks"*, In Proc. of 14th. International Conference no Computer Communications and Networks, San Diego, CA, USA, October 2005.

[62] Y. Xu, J. Heidemann, and D. Estrin, *"Geography-informed energy conservation for Ad Hoc routing"*, Proceedings of the 7th Annual International Conference on Mobile Computing and Networking, pp. 70-84, Rome, Italy, 2001.

[63] Y. Yao, and J. Gehrke, *"The Cougar Approach to In-Network Query Processing in Sensor Networks"*, SIGMOD, 2002.

[64] J. Yick, B. Mukherjee, and D. Ghosal, *"Wireless Sensor Network Survey"* Computer Networks, Vol. 52, pp. 2292-2330, 2008.

[65] L. A. Zadeh, *"Fuzzy Sets"*, Information and Control, Vol. 8, Issue 3, pp. 338-353, June 1965.

[66] Zigbee Alliance. http://www.zigbee.org/, Last Visited June 2011.

[67] D. Zimmer, A. Meckenstock, R. Unland, *"A General Model for Event Specification in Active Database Management Systems"*, In Proc. of the 5th. International Conference on Deductive and Object-Oriented Databases (DOOD'97), LNCS 1341, Montreux, Switzerland, December 1997.

[68] M. Zoumboulakis, G. Roussos, and A. Poulovassilis, *"Active Rules for Sensor Databases"*, Proceedings of the First Workshop on Data Management for Sensor Networks (DMSN'04), August 2004.

CURRICULUM VITAE

PERSONAL INFORMATION

Surname, Name: Şanlı, Özgür

Date of Birth: 15.06.1978

Place of Birth: Ankara

Marital Status: Single

Phone: +90 312 507-5976

Email: ozgur.sanli@ceng.metu.edu.tr

EDUCATION

Degree	Institution	Year of Graduation
MS	METU Computer Engineering	2003
BS	METU Computer Engineering	2000

WORK EXPERIENCE

Year	Place	Enrollment
2000-present	T.C. Merkez Bankası	IT Security Specialist
1999	Aselsan	Intern
1998	METU CC	Intern

PUBLICATIONS

- Ö. Şanlı, İ. Körpeoğlu, A. Yazıcı, *"Rule-Based In-Network Processing in Wireless Sensor Networks"*, IEEE Multi-conference on Systems and Control, pp.660-665, Saint Petersburg, Russia, July 2009.

PERSONAL INTERESTS AND HOBBIES

Tennis, Parachuting, Travelling, Electronics, Linguistics

FOREIGN LANGUAGES

Fluent English, Basic German and Spanish