# The Integration of Explanation-Based Learning and Fuzzy Control in the Context of Software Assurance as Applied to Modular Avionics

by

Matthew James Timperley

A thesis submitted in partial fulfilment for the requirements for the degree of Doctor of Philosophy at the University of Central Lancashire

October 2015

# Student Declaration Form

**Concurrent registration for two or more academic awards**

I declare that while registered as a candidate for the research degree, I have not been a registered candidate or enrolled student for another award of the University or other academic or professional institution.

**Material submitted for another award**

I declare that no material contained in the thesis has been used in any other submission for an academic award and is solely my own work.

**Collaboration**

Where a candidate's research programme is part of a collaborative project, the thesis must indicate in addition clearly the candidate's individual contribution and the extent of the collaboration. Please state below:

**Signature of Candidate:** *Matthew Timperley*

**Type of Award:** Doctor of Philosophy.

**School:** Computing, Engineering and Physical Sciences.

**Abstract**

A Modular Power Management System (MPMS) is an energy management system intended for highly modular applications, able to adapt to changing hardware intelligently. There is a dearth in the literature on Integrated Modular Avionics (IMA), which has previously not addressed the implications for software operating within this architecture. Namely, the adaptation of control laws to changing hardware. This work proposes some approaches to address this issue.

Control laws may require adaptation to overcome hardware degradation, or system upgrades. There is also a growing interest in the ability to change hardware configurations of UASs (Unmanned Aerial Systems) between missions, to better fit the characteristics of each one. Hardware changes in the aviation industry come with an additional caveat: in order for a software system to be used in aviation it must be certified as part of a platform. This certification process has no clear guidelines for adaptive systems. Adapting to a changing platform, as well as addressing the necessary certification effort, motivated the development of the MPMS.

The aim of the work is twofold. Firstly, to modify existing control strategies for new hardware. This is achieved with generalisation and transfer learning. Secondly, to reduce the workload involved with maintaining a safety argument for an adaptive controller. Three areas of work are used to demonstrate the satisfaction of this aim.

Explanation-Based Learning (EBL) is proposed for the derivation of new control laws. The EBL domain theory embodies general control strategies, which are specialised to form fuzzy rules. A method for translating explanation structures into fuzzy rules is presented. The generation of specific rules, from a general control strategy, is one way to adapt to controlling a modular platform. A fuzzy controller executes the rules derived by EBL. This maintains fast rule execution as well as the separation of strategy and application. The ability of EBL to generate rules which are useful when executed by a fuzzy controller is demonstrated by an experiment. A domain theory is given to control throttle output, which is used to generate fuzzy rules. These rules have a positive impact on energy consumption in simulated flight.

EBL is proposed, for rule derivation, because it focuses on generalisation. Generalisations can apply knowledge from one situation, or hardware, to another. This can be preferable to re-derivation of similar control laws. Furthermore, EBL can be augmented to include analogical reasoning when reaching an impasse. An algorithm which integrates analogy into EBL has been developed as part of this work. The inclusion of analogical reasoning facilitates transfer learning, which furthers the flexibility of the MPMS in adapting to new hardware. The adaptive capability of the MPMS is demonstrated by application to multiple simulated platforms.

EBL produces explanation structures. Augmenting these explanation structures with a safety-specific domain theory can produce skeletal safety cases. A technique to achieve this has been developed. Example structures are generated for previously derived fuzzy rules. Generating safety cases from explanation structures can form the basis for an adaptive safety argument.

# Contents

# List of Figures

# List of Tables

# Acknowledgements

This work would not have been possible without the funding received as part of my CASE award (EP/I501312/1). The industrial partner in this project is BAE Systems. I was given over to Ian Harrington to manage and would like to thank him for helping to keep me focused, making me feel supported, suggesting some useful reading and putting me in touch with Dr Gorry. I would like to thank Dr Gorry for suggesting insightful criticisms which shaped my work in the early stages.

My supervisory team have been brilliant. I would like to thank Joe Howe for always looking out for me, for getting me to see both the large and small pictures and always giving me good advice. I want to thank Gareth Bellaby for offering a sympathetic ear and a critical eye. I want to thank Maizura Mokhtar, who has made time all over the place to make sure that my work is always improving. She has been instrumental, letting me run with ideas freely, pulling me back to the task in hand when I needed it. Furthermore, I want to thank the whole team for all of their help and support.

Thank you to my family for supporting me. To my mum and dad for always being there for me and encouraging me. Especially to Jonny and Alex, for looking after me and inspiring me to think differently. Thanks to all of my friends for putting up with my ramblings and for distracting me. Thanks to Caitlin for a rigorous proofreading and for being an agile discussion partner. A special thanks to Adam and Louise Foster for buoying me up, looking after me post surgery, and for proofreading. Thanks are also given to Adam Bedford whose exhilarating discussions convinced me to not be shy about my thoughts. Thanks too to Saed for having my back. I also want to thank my cohort of fellow students, it has been a lot less lonely with you all in the same boat as me. A final thank you to Fong Wa Ha for making me smile again.

# List of Acronyms

**ANN** Artificial Neural Network

**ASI** Airspeed Indicator

**CBR** Case-Based Reasoning

**EBL** Explanation-Based Learning

**FLCHG** Flight Level Change

**FMS** Flight Management System

**GSN** Goal-Structuring Notation

**ICE** Internal Combustion Engine

**ILP** Inductive Logic Programming

**IMA** Integrated Modular Avionics

**LRU** Line Replaceable Unit

**MPMS** Modular Power Management System

**SDK** Software Development Kit

**UAS** Unmanned Aerial System

**VNAV** Vertical Navigation

# Chapter 1

# Introduction

A Modular Power Management System (MPMS) will be implemented to address an aim via the integration of Explanation-Based Learning (EBL) and fuzzy control. The MPMS is applied to modular avionics and also considers the software assurance of such a system. The aim of this project is to: *reduce the software and certification efforts engendered by the management of a modular platform.*

The project is to develop an MPMS for use on an Unmanned Aerial System (UAS). Power management is the intended domain. However, the MPMS is mainly concerned with addressing the issues of managing a modular system, hence Modular Power Management System. Another important feature of the MPMS is the inclusion of machine learning techniques to address the problem of power management for a hardware system whose components are modular.

A power system is comprised of the hardware required to provide and regulate power to other components or systems. Power systems are an example of a system which can be made in a modular manner. A modular system is one that is constructed in such a way that hardware items may be swapped out without significant system redesign [2].

A power management system is a software system which manages the hardware in a power system. Artificial Intelligence techniques such as expert systems [3] have already been applied to power management. Modular power systems have been designed, one example is given in [2]. The focus of the software for modular avionics systems is on supporting the reuse of code. When a change in the hardware of a platform occurs a change in the control software may be required. Code reuse allows for changes in hardware configuration to potentially be managed by a reconfiguration of software.

The challenge addressed in this work stems from the control of a modular system. Modular power management is taken as a specific example of the general problems which arise from the control of a modular platform.

Traditionally avionics platforms employed a federated architecture, but Integrated Modular Avionics (IMA) is gaining popularity [4]. Federated and IMA approaches are discussed in [5]. Federated and IMA systems will be further expounded upon in Chapter 2. Federated systems maintain as much separation between components as is possible, whereas IMA aims to share resources. This is motivated partially by the weight-saving potential from removing duplicate parts, which is a

major concern in avionics [6]. However, modular units still require their own power conversion components but may share computational and communicative resources. Separation is maintained logically for computational resources, as indicated in [7]. This is explained further in Chapter 2.

The MPMS is a software control system intended to operate within an IMA architecture. It is intended to run using the shared computational resources that are part of an IMA platform. The MPMS will take advantage of the greater integration of systems to manage the platform on a higher level than single-component control. However, by controlling more than one component the MPMS becomes more likely to be impacted by a change in hardware.

Current literature relating to IMA concentrates on the architecture and operating system considerations; for example, maintaining a logical separation between modules and the sharing of resources. This is illustrated in [7] and [2]. Changes to an IMA hardware platform may require changes to the control software. The impact of hardware changes on the software system forms a gap in the existing literature, as do coping strategies for this scenario.

Changes in hardware may require changes in software. When controlling a modular platform, these changes may require that the software system change its control laws in order to accommodate the change in hardware. The consequent change to the software system that is considered in this work. Reduction of the workload engendered would be beneficial. A machine learning technique could use adaptation to better accommodate changing hardware by altering its own control laws.

The major issue of applying a machine learning technique to modular avionics is the difficulty in pursuing software assurance. Changes in hardware, in aviation, require recertification of the platform. Different configurations are each certified in order to deal with this issue. Changes to the hardware are then permitted if it can be shown that an individual platform corresponds to a certified configuration [8]. Therefore, one challenge of managing a modular system comes from the recertification workload. A benefit of the MPMS is that it will reduce the recertification effort for the software system necessitated by a hardware change.

The specific objectives for this project are:

- *To investigate the integration of EBL and fuzzy controllers.* The EBL domain theory can be used to form structures which could be specialised to form new fuzzy rules. This will assist in the management of a modular system as changes in hardware may already be accounted for in the domain theory, but not the specific control rules. This process could account for changes in hardware where the specific hardware has changed, but the general control strategy still applies. An example of this kind of change would be upgrading a component to one which uses less energy.

- *To investigate the extension of EBL to perform transfer learning.* In order to support greater modularity it would be useful to extend the domain theory when no deductive option is possible. This could be achieved by the incorporation of analogical reasoning. The need to extend the domain theory could arise when the platform is altered by the addition of a piece of hardware that was not considered when engineering the domain theory.

- *To investigate the extension of EBL to produce certification artefacts.* Certification is required for a software system, as part of a platform. Changes to software require recertification. EBL could be extended, by the inclusion of a safety-specific domain theory, to generate safety cases alongside each control rule generated.

EBL, a machine learning algorithm, is proposed as a technique to meet the aim. The more seamless control of modular systems, by reusing strategies from similar components as a basis for the control of previously unseen ones, is posited to address the project aim. Rather than constructing a new strategy there is the opportunity for transfer learning. Rather than learning from scratch, transfer learning uses knowledge out of the context in which it was derived to solve problems [9]. In this case, it is control laws which are being derived. Transfer learning is realised by the application of machine learning techniques, such as Case-Based Reasoning (CBR) [10] and Inductive Logic Programming (ILP) [9]. The proposed technique is based on EBL.

EBL aims to define a concept by extracting as much information from each training example presented as possible [11]. EBL generates an explanation structure to explain how a training example satisfies a given goal concept. EBL can be considered as a search through the space of all possible explanations. This structure connects a given goal to training data, which acts as an inductive bias. An explanation structure can be flattened and generalised, generating a new rule that explains the goal. The rule is generalised so that future examples do not necessarily require additional explanation.

EBL was chosen because it may be of use in minimising the software overhead. Firstly, generalisation is an inherent part of the technique. Generalisation is one way to apply knowledge, formed in one situation, to another situation. Utilising general rules reduces the overhead of generating rules for every situation. Secondly, a domain theory can contain general strategies. The strategies could be specialised to form specific control rules. This allows for the development of a small domain theory which can be used to generate rules for a range of situations.

These control rules will take the form of fuzzy rules, for execution by a fuzzy controller. Fuzzy controllers are a viable solution as they are human-readable and execute rapidly. Human-readability may reduce the potential certification issues of an adaptive controller. Rapid execution is necessary as the computational resources on board a UAS are limited.

EBL was chosen, in terms of minimising the certification overhead, for two reasons. Firstly, its human-readability can be an aid to certification in the aerospace industry. Secondly, the domain theory contains information in addition to that required to define a control rule. This additional contextual information is useful in understanding a given control rule. This would also be an aid to certification.

The remainder of the thesis is structured as laid out in the following section.

## 1.1 Structure of the Thesis

**Chapter 2**
Chapter 2 provides a review of the literature in order to inform the design of the MPMS. The work is also contextualised within the existing literature. Namely, there is a gap in the current literature associated with modular control into which this work fits. The gap is in managing the impact that modular hardware has on its control software. The choice of EBL and fuzzy controllers, as well as the notion of adaptive safety cases, are also framed within the existing literature.

**Chapter 3**
Chapter 3 proposes the integration of EBL and fuzzy controllers. The communication between each

system is considered as well as the role that each technique performs. The approach is presented and discussed. Additionally, the approach is implemented and applied initially to two separate hardware platforms. The need for further experimentation to validate the technique is highlighted. Further consideration of the applicability of the technique to modular control is also proposed.

**Chapter 4**
Chapter 4 builds on the work in Chapter 3. This chapter seeks to analyse the applicability of the approach to modular control. Two experiments are conducted in order to show that the integration of EBL and fuzzy controllers can lead to rules which are beneficial. The first experiment compares the effect of generated rules on throttle control with the default autopilot behaviour in the simulator. This experiment seeks to show that useful rules can be generated by the proposed approach. In order to consider the impact of modular control a second experiment is conducted. The second experiment differs from the first by controlling a significantly different platform. This is an extreme case of a change in hardware, where all the items of hardware have changed, but the control strategy still applies to the throttle.

**Chapter 5**
Chapter 5 establishes the method for including analogical reasoning in EBL. The role of analogy within the MPMS is presented. Analogy is used only to solve an impasse. This is in line with solving problems which are not covered explicitly by the domain theory. An existing strategy is altered in order to cope with the new problem. An experiment is conducted where pitch is controlled by indirectly affecting the throttle. The control of a different piece of hardware is not catered for by the domain theory. The generation of rules sits in support of the possibility of using the technique to adapt a domain theory to fit new pieces of hardware by employing past strategies.

**Chapter 6**
Chapter 6 presents a position relating to the certification of adaptive controllers in the aviation industry, that of an adaptive argument. An adaptive safety argument takes advantage of the characteristics of machine learning techniques to mirror controller development. This is realised by the generation of (skeletal) safety cases. A method for augmenting EBL to generate safety cases is described. Some rules which are derived in Chapter Three are used to generate safety cases for consideration. These rule derivations are augmented to form skeletal safety cases, which are presented. The generation of potential safety cases sits in support of the proposed method, further aligning EBL within the aviation domain.

**Chapter 7**
This chapter concludes this project with a reflective analysis. The analysis highlights the need for further work in several areas. The main areas for further work are:

- Further development and investigation into goal selection for fuzzy rule generation via EBL.
- The investigation of a technique to further specialise and evaluate analogically generated rules.
- The evaluation of EBL to generate safety cases which are usable in the aviation domain, as well as the flexibility of the approach.

## 1.2 Unique Aspects of the Project

The aim of the project, to reduce the software and certification efforts engendered by the management of a modular system, will be addressed via the objectives. The unique aspects of the project to satisfy the aim are:

1. The application of EBL to modular management.
2. The manner in which EBL is extended to incorporate analogical reasoning.
3. The integration of EBL and fuzzy controllers.
4. The extension of EBL to automatically generate safety cases.

# Chapter 2

# Background and Related Work

The MPMS is designed using a particular assumption: this differentiates it from other power management systems. The assumption is that the hardware being managed will change often. It is possible to use configuration to accommodate changing hardware. Various configurations, which define a particular combination of hardware and software, are considered flightworthy. Each one has passed through regulatory processes. An aircraft can undergo a hardware change as long as it can be shown to correspond to a particular configuration [8]. In this case the software components must already have been developed to control the particular hardware items within the configuration.

However, inclusion of a trustworthy learning mechanism could mean that changes in hardware would not require changes to the system, avoiding the cost of pre-certifying specific hardware configurations. A trustworthy learning element could also give the advantage that changes such as hardware degradation can be accounted for automatically.

One way for controllers to adapt to changes in hardware is by configuration. Configuration allows non-adaptive power management systems to have tailored behaviour. However, effort is required to generate each new configuration, especially as new hardware is developed. This may be particularly pronounced for a UAS since there is interest in frequent changes in hardware to accommodate changing roles [12]. Changes occurring over time, such as damage or wear, will also alter the nature of the system so that pre-computed ideal behaviours may no longer be appropriate [13]. It is possible that the hardware being added did not exist when the control laws were first elicited. This is a source of additional work as different configurations could need to be created and updated as new hardware is developed.

Inclusion of a learning element is an alternative to configuration. Learning is less brittle than configuration, allowing as it does for adaptation to unexpected changes in the system. The MPMS will incorporate learning in order to reduce the workload engendered by a change in hardware on control systems.

Machine learning can alleviate this problem. Examples of power management systems with learning are presented in [14],[15],[16],[17]. This learning element will work towards deriving an appropriate power management strategy. However, alterations to the control laws may still be required, depending on their specificity with respect to the original platform. In order to reduce this workload it should be possible to re-purpose general control strategies, from similar components, for previously unseen hardware. This is an example of transfer learning.

EBL will be extended in order to provide transfer learning, applicable to management of a modular system. There are advantages in combining more than one learning method, such as EBL and analogical learning. The inclusion of multiple types of learning algorithm into a single agent-based system was studied in [18]. The work concludes that the inclusion of additional learning modes is likely to be beneficial, particularly in complex environments.

## 2.1 Modular Avionics

Traditionally, avionics platforms employed a federated architecture, but IMA is gaining popularity [4]. In the aviation industry there is interest in moving from the federated architecture to a more modular systems architecture [6]. This is partly driven by the potential benefits of greater integration. Modularity in aviation has been gaining interest because of both the additional future-proofing inherent in having easily replaceable modules and the flexibility of role that this engenders [19]. This would allow the tailoring of platforms to fit specific missions, as advocated in [12].

Federated and IMA approaches are discussed in [5], which focuses on the transition between the older federated architecture and the adoption of the newer IMA architecture. Federated systems maintain as much separation between components as is possible with each LRU (Line Replaceable Unit) being self contained. LRUs have their own power conversion and computational resources included, which increases weight when compared to IMA. IMA aims to share resources whilst maintaining separation. This can reduce weight since modular units can be dumb terminals [5]. Weight reduction is a serious concern in avionics [6]. The modules will have a common form factor(s) and share power and communication buses, as well as computational resources. However, modular units still require their own power conversion components. Separation is maintained logically for computational resources, as indicated in [7].

The software for individual modular units run on the same platform and shares computational resources. In the federated architecture, the software was physically separated, being part of each LRU. One benefit to physical separation is that failures are separated. IMA computing platforms therefore maintain a definite logical separation. The processes for each IMA are kept separate through programmatic techniques, rather than physical distance.

Current literature relating to IMA concentrates on architecture and operating system considerations; for example, sharing resources and maintaining a logical separation between modules. This is embodied in [7] and [2]. Software modules operate within this architecture, such as the MPMS. A more modular platform means that changes to the controlled hardware will be more frequent. The literature on IMA sets out a framework where hardware can be altered more freely, as well as sharing resources. It is the seeming dearth in literature regarding the impact of hardware changes on the controlling software which has motivated this work.

Within an IMA architecture changing hardware can have a more significant and less predictable impact, where its software controls multiple components within the system. Existing technology copes well where the control software only applies to a single system component in isolation. It is more difficult to reason about software which controls multiple components due to the potential for interaction between the components being controlled. A traditional approach requires a significant amount of test and analysis to ensure this change does not produce undesirable side effect behaviour. It is hypothesised that using EBL may allow significant reduction in this ef-

fort to establish the same demanding standards of evidence required by a traditional federated architecture.

Using EBL for generating control laws, because of its focus on generalisation, could aid in reducing the workload engendered by a change in hardware.

## 2.2 Explanation-Based Learning

The explanation effect states that students who explain examples more thoroughly to themselves tend to learn more [20]. Explanation-Based Generalization, an early proposition of EBL, aimed to use a single example, rather than many, to draw a conclusion [21]. EBL now aims to define a concept by extracting as much information from each training example presented as is possible [11]. One or more definitions may be acquired. Adaptations have been developed that concern the generalisations between examples, though each can potentially gain useful knowledge from a single example [22].

EBL can be considered as a search through the space of all possible explanations. EBL generates an explanation structure to explain how a training example satisfies a given goal concept [23]. Explanation structures often take the form of trees with a goal at the top, training data at the bottom and domain theory connecting them. Terminal nodes can be referred to as leaf nodes.

Explanation structures can be formed by backward-chained unification from the goal to the training data through the domain theory. Backward-chaining refers to the fact that the explanations supporting a goal are explored, rather than the implications a goal has, as in forward-chaining.

The resolution principle is proposed in [24] to perform unification. Unification is a basis for deductive reasoning in machine learning. The Robinson algorithm is employed to find a substitution, if one exists, under which two given terms match. As an example *can_supply(X,C)* and *can_supply(fuel_cell_1, ASI_demand)* are unifiable under the substitution *{X=fuel_cell_1, C=ASI_demand}*. This is used to determine when two rules can be connected in the explanation structure. Whilst there are other resolution algorithms, use of the Robinson algorithm is supported by practical performance data described in [25].

The domain theory is the set of rules which in conjunction define the search space [26]. Alternatively, the domain theory is a set of *a priori* beliefs that the system holds. The domain theory usually takes the form of a set of facts and rules, often provided by a human expert, relating to a specific domain. These rules and facts may be stored in a rule-base, or other data structure. The domain theory encoded provides an inductive bias, by restricting the explanations that fit within the model defined by the domain theory. This search is further restricted by a training example to give a specific explanation. An inductive bias is information used to restrict the possible inductive leaps. Many may be permitted by a domain theory but only one might fit the particular situation.

Training examples are a structured form of input and can be thought of as observations. They are structured such that they fit with the internal model expressed by the domain theory. Training examples lead to specific conclusions. The training examples are used to constrain the search for a concept definition and in conjunction with the domain theory language form an inductive bias. A vocabulary can be too restrictive, giving rise to trivial concept definitions. It can also be overly general, which increases the complexity of the problem. A vocabulary is a set of all predicates

within a program.

Explanation structures can be generalised and chunked into a new rule so that future examples do not necessarily require additional explanation. Chunking often takes the form of computing the most general preconditions under which the explanation holds. One way to achieve this is to flatten the explanation structure, take the lowest nodes, remove their variable bindings, then use these in concert to infer the goal [22]. This, if all the nodes used are operational, gives the new rule.

Operationality is expressed with an operationality criterion. Operational, in terms of EBL, has several posited definitions but often relates to computational efficiency, [21],[27]. This can be considered as a filter for rules whose inclusion improves the performance of the controller. Generating additional rules will increase computational costs when selecting or executing rules. Therefore, these rules need to have a positive impact in order to justify their generation. Rules with greater generality also increase the selection costs as they potentially apply to many situations, but may not always prove beneficial to execute. An operationality criterion is used to determine when to stop explanation. This will need to be considered by any application of EBL.

### 2.2.1 EBL Example

EBL begins with a goal. In this case: *bigger(large,small).* This goal asks whether the term large is bigger than the term small. Resolution is used to identify rules which imply this goal. In this case: $!same(X,Y) \wedge bigger(X,Z) \wedge bigger(Z,Y)$ implies $bigger(X,Y)$. A substitution $\theta$ is found between the original goal and the antecedent of the new rule. In this instance $\theta$ is *X=large, Y=small.* Under the substitution $\theta$ both instances of *bigger* will match. This node is added to the tree under the goal. The precedents of the new rule then become goals. Note also that the substitution is used to bind the variables in the new rule. This provides a search bias. This continues until the nodes added are operational. In this case, inputs from the fuzzy system will be considered operational.

If, at some point, no new rule or operational node can be found, then one of two things happens. Either the explanation fails or backtracking occurs. When backtracking, the explanation structure returns to the state it was in at the last choice point. A choice point is a step where multiple inferences could be made i.e. there was a choice of nodes to add to the tree, each with a viable substitution. Deduction can then continue using one of the alternative rules from the domain theory.

Once an explanation structure has been generated, it can be generalised (chunked). Generalisation results in new, more general rules. This can simply take the form of undoing variable bindings [22]. The leaf nodes are then taken to imply the goal. This is the most common, but not the only method of generalisation.

The explanation structures in Figure 2.1 depict some basic reasoning about relative orders of magnitude. Two copies of the same explanation structure are shown. The top structure shows variable bindings, the bottom shows the unbound nodes. Note that operational nodes remain bound, but their parents would be used in generalisation. The terminal nodes match the domain theory nodes when specialised (top image). For brevity explanation structures in this thesis exclude the sub-tree under the negated node.

The implementation includes negation as failure. Negation as failure is when an explanation of a

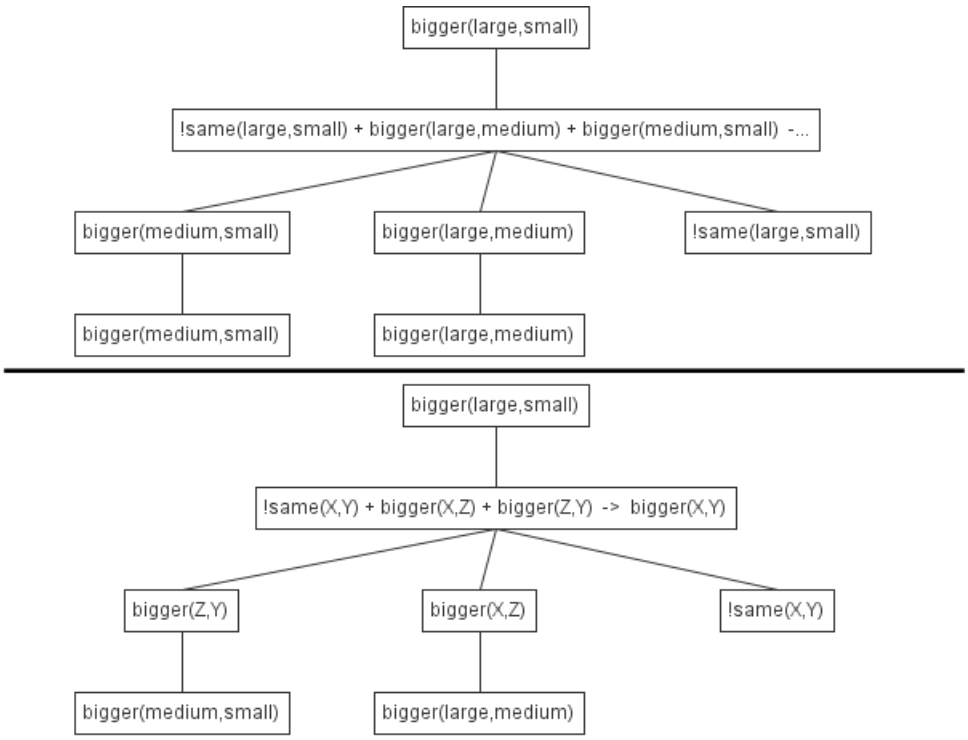


**Figure 2.1: Two explanation structures, the upper shows variable bindings, the lower unbound nodes.**

goal fails it is construed as a successful explanation for the negation of that goal. In this example, the failure to show that two values are the same supports the supposition that they are different. This is the goal *!Same(large,small)* in Figure 2.1.

EBL uses generalisation to generate new rules. Each explanation structure results in a single EBL rule. These rules prevent regeneration of the same structure in similar situations. This is achieved by implication of the goal by the leaf nodes, alongside the introduction of variables. The introduction of variables transfers knowledge from one situation to another. However, EBL is computationally expensive and were it to be used directly to control a UAS, there would be difficulty in dealing with crisp values. Alternatively, EBL could be used to generate control rules for a fuzzy controller. Fuzzy controllers are computationally cheaper than EBL and are also human-readable.

Crisp sets are those who are entirely defined by their members, traditional sets [28]. Membership of a crisp set is binary. In this work the system inputs come as crisp values as each number is a traditional set of one. Fuzzy sets are an alternative to crisp sets.

## 2.3 Fuzzy Systems

Fuzzy sets are different to the sets from set theory. They were first introduced in [29], then proposed as a method of capturing numerical data as linguistic variables in [30]. These are variables whose values are words, rather than numbers. Altitude, a linguistic variable, could be described by mapping a continuum of numerical values onto several fuzzy sets. High and low could be linguistic variables. The continuum of values will fall into either one, none or both sets. It is important to note that altitude will always be described using all sets and a membership value, which shows

to what degree that value epitomises that fuzzy set. So, when a value falls into no fuzzy set it is more accurate to say that it has a membership value of zero for each set. A linguistic variable is therefore defined by four things [30], given below:

- The name of the variable, such as altitude.
- The linguistic values it may have, such as high and low.
- The universe of discourse, which is the numerical values.
- The membership functions, which map the linguistic values onto the universe of discourse.

Each set has a membership function ($\mu$). The membership function defines the shape, or distribution, of the set over the universe of discourse. These membership functions take a crisp value and output the degree of membership the value has for a corresponding set [29]. Membership values are in the range [0,1]. It is worth noting that membership functions do not assign probabilities [29]. A fuzzy system is visualised in Figure 2.2.

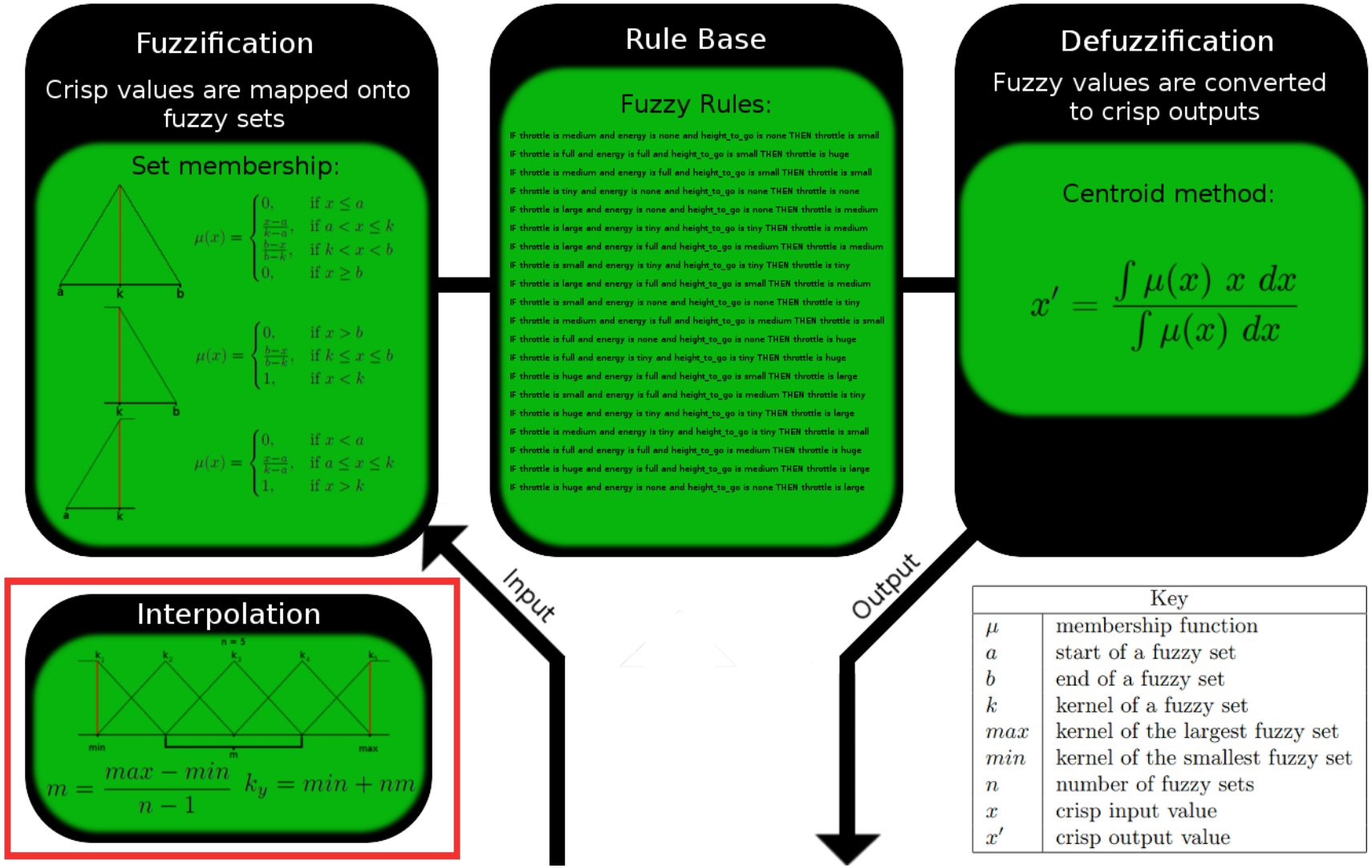


**Figure 2.2: The proposed fuzzy control system is depicted in this figure. The area in red is an addition proposed during Chapter 3.**

The input representing altitude might have a crisp value of 800m. However, it is not clear intuitively whether this means that the altitude is high or low; it likely depends on the context e.g. platform. The altitude can be represented as the linguistic values *low 0.6* and *high 0.4*. The membership values, 0.4 and 0.6, represent how much the crisp value embodies the fuzzy set. This results in the crisp input being described by membership values for each fuzzy set. As an example of how a continuum can be described in terms of a fuzzy variable, see Figure 2.3. The possible shapes defined by membership functions, as well as the ways in which they overlap, are varied. The particular characteristics of the shapes and distributions of membership functions are usually considered expert knowledge [31].

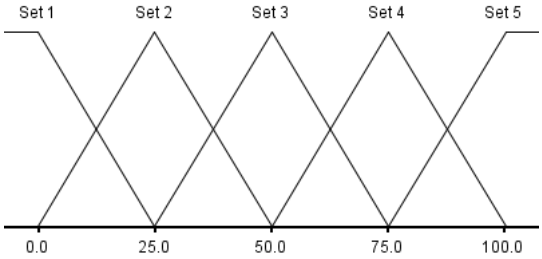


**Figure 2.3: An example linguistic variable, describing the universe of discourse [0,100] using the linguistic values of sets 1 through 5.**

Fuzzy logic also includes operators such as conjunction, disjunction, negation and hedges [30]. Hedges are terms which affect the meaning of another term e.g. very. Fuzzy logic has been applied alone [32], included into first order fuzzy logic [33], integrated Artificial Neural Networks (ANNs) and other techniques [34]. The main application for fuzzy logic, however, is in fuzzy controllers. The first fuzzy logic controller was used to control a steam engine in [35]. Since then, there have been many applications of fuzzy control [34].

Any given fuzzy controller broadly falls into one or more of four categories, based on the concepts [36]:

- Expert knowledge and control engineering.
- Existing controllers.
- A mathematical system model.
- Self-learning.

Input data is fuzzified, which is the process of describing the crisp input as a linguistic variable. A linguistic variable is one which has linguistic, rather than numeric values [30]. Fuzzification is facilitated by membership functions. The controller has a rule base which comprises all of the control rules. The fuzzy rules often express expert knowledge, but may be derived by other techniques [31]. The inputs are used to identify which rules are triggered. The appropriate rules are used to derive the value of output variables, via an inference mechanism. An algorithm is then used to obtain a crisp output from fuzzy antecedents.

The two dominant fuzzy controller designs are the Mamdani-type and the Sugeno-type. Mamdani controllers use a process called defuzzification, which involves fuzzy output sets. Sugeno controllers, considered to be computationally faster, use a weighted average and do not use fuzzy output sets. Furthermore, the two design paradigms also differ in how they represent the fuzzy rules. Mamdani

controllers are often chosen due to their readability and power of expression, but the representation used in Sugeno controllers is more amenable to optimisation techniques. Further information, including a comparison of the two designs can be found in [37].

In aviation, readability and transparency are important attributes as they can aid in certification. Using fuzzy controllers on a UAS can support readability. Fuzzy rules are expressed in human-readable language. Fuzzification also provides a way to convert crisp values sensibly into linguistic variables. Linguistic variables could be useful in EBL, for the generation of new fuzzy rules. To compare values in EBL a system of inferring magnitude would be appropriate. Rules are needed to express this concept within the domain theory. Without the fuzzification of inputs EBL could require many rules. Either each number would require a rule to state its relation to other numbers, or numbers could be split into components and the domain theory could have rules which relate them, alongside a smaller number of rules for the magnitude of specific numbers. This would result in more rules than relating a few linguistic variables, as each one represents a range of values. When executing the fuzzy rules the membership value can still be used to modify the intensity of the output, without having to consider this during rule generation. By having terms which represent a range of values, rules can be derived which fit a number of specific cases. When these cases can be interpolated to fit multiple ranges then such rules can provide generalisation.

The integration of EBL and fuzzy controllers comprises a surprising dearth in the literature. A contribution to filling this gap in the literature will be proposed as part of this work. The generation of fuzzy rules by EBL allows for rules to be generated from general strategies. However, this generalisation relies upon the substitution of specifics for variables. The predicates themselves do not change. It may be the case that, in order to apply to a new piece of hardware, predicates need to be substituted. In the event that a deductive resolution is not possible, an extension of EBL to incorporate analogy could be useful.

## 2.4 EBL and Analogy

If the components and architecture of a power system are modular and could change at any point, then the energy management system should be able to derive appropriate control laws for previously unseen components. It is likely that a new component will share both similarities to other components and an energy management strategy. This transfer of knowledge between situations can be achieved by derivational analogy [38]. However, the transfer of knowledge occurs when there is a similarity in plans. In the given context it may be useful to establish abstraction relationships between concepts. This can then be used to guide EBL when making inductive leaps, as in analogical reasoning. Analogical reasoning uses the correspondence between two ideas to make inferences about the target/unfamiliar idea. Analogy has been used to map different situations to one another, in order to apply prior knowledge to a new situation [39]; thereby performing transfer learning.

There are four stages in analogical reasoning [40].

1. *Access* locates a body of knowledge (the base) that may be analogous to the current situation (the target) and prunes irrelevant features from the analogy.

2. *Mapping* finds similarities between the base and target and possibly to infer new features

from the base to the target by analogical inference.

3. *Evaluation* considers the quality of the match and its consistency with general domain knowledge.

4. *Use*, application of a valid analogy.

An application of analogical learning with EBL [26] was able to construct explanations from incomplete domain theories where the combination of domains made a deductive explanation possible. This involved drawing analogues between rules and transferring them to a new situation. For example *"If there are atoms in a domain that imply A and there are analogous atoms in another domain, then an analogue of A can be determined in the other domain"*. This is a transfer of knowledge between domains, which is an application of analogy. A common language is assumed in this work as the analogies are drawn between atoms sharing a predicate. Predicates are assumed to appear in multiple concept definitions since the domain theory is expert knowledge. The experts share a language for communication as well as a subset of technical language; it therefore is reasonable to assume that a given predicate will occur in multiple concept definitions.

This work proposes that less exact analogues can be used to derive relations between terms with different predicates. A similarity in language is still assumed. Rather than establishing a link between the same terms in different domains, the emphasis is on relating different terms within the same domain. These two terms are used to derive a new concept to which both belong, such is the nature of their relation. This can allow inductive leaps, by replacing one concept in a rule with an analogue. These analogues are not deductively entailed without additions to the domain theory stating when concepts are similar. Another aspect of the proposed technique is that training examples encountered can provide an inductive bias for disregarding terms that are not relevant from a definition of analogical similarity.

When two concepts share an abstract similarity, it is plausible to suppose that conclusions that apply to one could be mapped to the other. There is an issue with assuming that an abstract similarity exists. Not all analogical inferences are equally likely [41] and where the similarity is based on features, not all will be relevant. This is partially ameliorated by EBL disregarding features not relevant to the example. When trying to derive an abstract similarity between two concepts, even using an example as an inductive bias to disregard features, it is likely that not all features in the example are relevant to the analogue being derived. As well as using an example as an inductive bias empirical validation could help disregard spurious analogies.

This type of analogy, from information in [40], could be stated as using similarity-based generalisation in order to perform a form of analogical reasoning. The analogical reasoning is restricted to generalising information to hold to more concepts than when derived. This differs from [26] by increasing slightly the deductive closure of the program in order to apply analogical reasoning in more restricted form. This increase in deductive closure is based on the assumption that the addition of a new concept increases the number of facts that can be deduced within the system; though these may be only semantic or erroneous differences, rather than resulting in a meaningful increase in closure.

In other applications of analogical reasoning (e.g. [38], [42]) a known solution to a similar problem is found and mutated to be applicable to the current problem. CBR was the method of analogical reasoning in [38] and [42]. CBR uses similarities between previous solutions to various problems

to solve the one at hand, the target [43]. Closely similar solutions are modified to solve the target problem. Similarity metrics are used to guide searches for similar solutions.

In order to more easily compare to EBL, a solution can be thought of as being a particular path through the domain theory, or a subset of the domain theory. This may take the form of a single rule such as: *battery(bat_1) + can_supply(bat_1, demand_1)* $\rightarrow$ *supply(bat_1, demand_1).* Therefore a domain theory can be considered as a set of solutions i.e. all the solutions reachable within the domain theory's deductive closure. So, to apply knowledge from another situation, characterised by different outer predicates, a link between predicates could be elicited.

During explanation, adopting an analogy could occur by temporarily swapping outer predicates. This will necessarily change the rules that follow to ones using the new predicate. This will cause different rules to match the given term which will lead to still other rules which are part of a different solution. This adoption of a different predicate can be viewed as similar to mutating a given solution to fit a situation or, alternatively, mutating the given situation to entail a different solution. This can allow inductive leaps, by replacing one concept in a rule with an analogue, which are not deductively entailed without additions to the domain theory stating when concepts are similar.

If comparing the technique proposed with that of derivational analogy [38], the analogue is a concept rather than a situation (although this is mostly a superficial difference, as a situation may be a concept). Another difference is, the presented technique, rather than tailoring a previous solution to a new situation, generalises a previous line of reasoning, within a solution, to more concepts. Applying analogy to only a single line within an example is similar to when students refer to a specific line of a previous example to justify some reasoning rather than the example as a whole [20]. The technique also shares some conceptual similarities with [44], though the specifics and application differ.

Both this work and that of Könic et al [45] map between concepts in order to apply knowledge from one situation to another. There are some differences between this work and [45]. This technique is proposed to derive links between previously unrelated concepts. These links are embodied in a new, more abstract, concept definition. This technique has a different method of application, as an alternative to failure. Both works use a goal and explanation, as a bias, to constrain the possible analogies. This technique also maps between potentially overlapping concepts rather than like terms.

The computational resources of a UAS are limited. Under a federated architecture the computational hardware needs to be developed and included in each Line-Replaceable Unit (LRU) [5], in order to maintain the separation of systems. Even in an IMA system, as assumed to be the case for the purpose of this paper, processing resources likely need to be carefully negotiated between disparate developers and the IMA system integrator [4]. The separation of rule generation, the more computationally expensive task, and rule execution is advocated to address this issue. It is reasonable to assume that a UAS has both a line of communication and a ground station with which to communicate, as in works such as [7] and [12]. Rule generation can therefore take place at the ground station, with rule execution on-board the UAS (see Figure 2.4). To achieve this separation, EBL is used to generate control laws to be executed by a fuzzy controller.

The reason for extending EBL to include analogy, rather than adopting CBR is that EBL primarily uses deduction. This can produce rules for which determinism is easier to demonstrate. It may

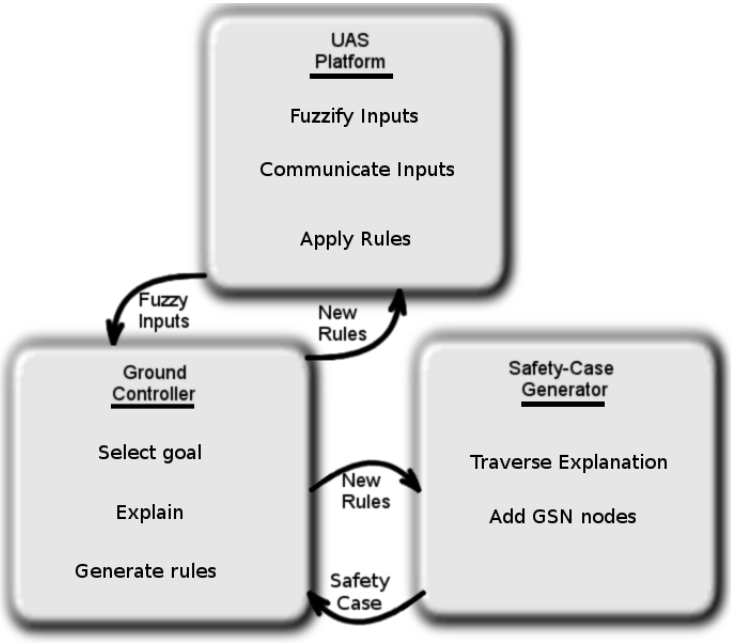


**Figure 2.4: Architectural Overview.**

also be easier to gain acceptance with regards to aviation certification to have rules which are primarily derived using deduction. The application of EBL with regard to certification in aviation is considered further in the following Section.

## 2.5 Certification in Aviation

Certification aims to give an appropriate level of confidence that a particular system / platform does not pose an unacceptable risk of causing harm [46]. Commercial aviation regulations are defined by an appropriate governing body such as FAA(US), EASA and CAA(UK). These bodies produce standards to describe the means by which aircraft and their components are judged acceptable for use. These standards are often augmented by guidelines, which between them define "an acceptable means of compliance" to the standard.

Software is not inherently hazardous, except in terms of the consequences of intended or unintended control it exerts over the system it controls. In order to understand the contribution software can make either on its own or in combination with another system component to the occurrence of a hazardous event it is necessary to construct a model of all known hazards and then track the reliance being placed on the software to ensure the prevention of the hazard. Obviously the integrity expected of the software increases as the severity of the identified hazard increases. Similarly the requirement for integrity increases where the software can cause the hazard on its own as opposed to a situation where an independent concurrent failure must also exist.

A safety case is a representation of the hazard space and the contribution that software can make to the occurrence of each hazard. A safety case links the claims that form the safety argument to the evidence that supports its acceptability. Safety cases for components can be combined to form substantiated arguments for larger system components or whole aircraft. Simplistically, it is often argued that the effort required to certify a system increases as the number of components increases.

In a modular system there are more components and they integrate to form a variety of allowed configurations. On the contrary, however it may be that the systematic combination of component based safety cases may in fact be less effort due to the controlled interfaces through which they operate and the re-use of safety case fragments and evidence across a range of components. Research into this would be beneficial.

The two elements of this process "standards compliance" and "safety case construction" are discussed in [47],[48].

### 2.5.1 DO-178

DO-178 (US), or ED12 in Europe, is a guideline on how to develop a software-based system. This guideline defines an acceptable means of compliance for software based systems as defined by the civil aviation authorities. Even though this document is only a guideline it is almost always adopted as it is easier to demonstrate compliance than it is to persuade the aviation authorities of the acceptability of an alternative means of compliance.

Since these documents are guidelines they are not absolute and arguments may be presented in other ways, though in practice these guidelines are frequently followed [49]. Flight critical software must demonstrate compliance to airworthiness standards. DO-178 is identified as the method of choice for this. Alternatives are permitted but not defined. So, in practice, DO-178 is treated more like a standard [13].

At the time of writing DO-178 is at revision C. The document guides the software life-cycle used by establishing aims, processes and evidence required to demonstrate compliance at each stage [49].

These guidelines define an acceptable means of compliance for DO-178 for a range of software levels. These levels are proportionately more demanding of rigour and supporting evidence as the hazard severity and contribution made by software to its occurrence increases. Level A software has the highest integrity obligation. Typically level A is designated for software for which a failure would lead directly to a catastrophic hazard occurring. Software would only be assigned to level B to D where software failure can result either in a less severe hazard or where other independent failure mitigations exist.

Adherence to DO-178 can be indicated by fulfilling process objectives. Process objectives are the suggested ways, for a given process, to show that a system has been developed to the standard. Process objectives relate to showing in what ways the system fulfils its specification, or its compliance to specification. Higher software levels have more process objectives, shown in 6.1. Some argue that the demonstration of mathematical correctness should be the goal for the highest integrity level, however it is not obvious whether greater correctness to specification also increases confidence in the safety of the system [46]. This is embodied in the quote below:

*"Part of the argument that should be supplied for correctness-based software guidelines such as DO-178B is a resolution of the conundrum mentioned earlier: how more assurance of correctness (as required by the higher DALs of DO-178B, or the SILs of IEC-61508) renders software more suitable for applications that require lower likelihood of failure (i.e., where failure conditions are more serious). This is a significant topic that has received scant attention."* [46]

| Failure Condition | Software Level | Process Objectives |
|---|---|---|
| Catastrophic | A | 66 |
| Hazardous | B | 65 |
| Major | C | 57 |
| Minor | D | 28 |
| No Effect | E | 0 |

**Table 2.1: Failure conditions, software levels and objectives. Table is adapted from [49].**

An alternative to compliance based certification is to present a rigorous argument which supports the claim that the system as developed is safe, rather than relying on the use of approved processes alone to certify novel systems. This is allowed by the UK Military Airworthiness Authority (MAA) under def-stan 00-56 issue 3 [50].

### 2.5.2 UAS Certification

It is worth noting that not all cases are covered by the document, for example UAS specific software. This is because DO-178, the de facto guideline to obtaining certification for aviation software, has no UAS-specific provisions. The ASTRAEA project [51] has an aim to establish an approach to UAS certification, through considering the certification of a virtual platform [52]. The virtual approach is also adopted in [53]. There is an impasse with regards to UAS certification: regulatory bodies want existing prototypes to base certification guidelines on, but such a prototype needs to be certifiable before it can be used to collect data in most airspaces [53]. This helps to illustrate the importance of making a strong safety argument when dealing with aviation software and certification.

Whilst revision C of DO-178 addresses techniques such as formal methods and Object Oriented (OO) programming, a longtime standard in Software Engineering, adaptive systems remain a certification challenge because of the absence of a suggested way of obtaining compliance via DO-178. In theory, DO-178 is applicable to all flight control software but for adaptive software there are additional challenges [50],[13] since the gains, or situation-action bindings can change after deployment. An approach that gives partial ability to alter gains but avoids the main issue with adaptive controllers, of not being deterministic, is gain Scheduling [54]. These issues affect the MPMS, which will be mainly applied to an all-electric IMA UAS. The dominant issue is, however, the certification of adaptive controllers.

### 2.5.3 Certifying Adaptive Flight Control

Adaptive flight control software changes its gains, which could be conceived as its situation-action bindings, after deployment. This is not allowed by current certification guidelines. The main areas that adaptive systems may require additional work to conform to DO-178B, according to [13], are:

1. defining software performance requirements.

2. providing a software verification plan.

3. defining software requirements and derived requirements.

4. providing software verification test cases and procedures.

5. providing a Plan for Software Aspects of Certification (PSAC).

6. providing software life cycle data.

Adaptive systems, such as expert systems and ANNs, have been adopted in safety-related systems but are typically restricted to advisory roles [55] [56]. Approaches for certifying adaptive systems have been proposed. Two main types of safety argument used for software systems are process and product based [55]. Process based arguments provide assurance by employing processes during development. The limitation is that there is no clear link between the development processes employed and the safe behaviour of an ANN. Product based arguments provide assurance based on functional behaviour. Hybrid ANNs, which represent symbolic knowledge within an ANN, are considered in [55]. A hybrid ANN was used to illustrate an approach for certifying ANNs based on their functional behaviour.

Non-adaptive controllers cannot alter their gains to match evolving situations, such as decay in system responsiveness. Adaptive controllers can alter gains after deployment automatically but this comes with the issue that current certification guidelines do not allow this. An approach that gives partial ability to alter gains whilst remaining deterministic is gain scheduling. Gain scheduling is a method that has been used to certify controllers that are adaptive by pre-computing behaviour in the full flight regime. The gains that would normally be adaptive are pre-computed for given operating conditions and stored in lookup tables [13] which cover the entire flight regime. This makes the adaptive controller predictable in any given situation that has been pre-computed. This could be viewed as a graceful degradation of adaptability in order to become deterministic. Gain scheduling allows one to demonstrate that certain situations, ideally all possible situations, are covered since the response is explicitly calculated and stored.

The use of gain scheduling implies an important point; determinism is a key issue in putting forward a safety argument. In this case determinism refers to the fact that for a given situation the reaction will be known. Test coverage, demonstrating that all possible situations are covered will also need to be shown in order to obtain certification [54]. The aforementioned limitations in applying DO-178 to adaptive control imply that safety case development might be the more appropriate method for certifying this system.

### 2.5.4 Goal-Structuring Notation and EBL

Goal-Structuring Notation (GSN) is a notation for presenting safety cases. Safety cases represent a safety argument and substantiate claims with evidence. They have been adopted in the aviation industry for representing arguments [1]. GSN is a tree structure comprised of the nodes given in Figure 2.5 stemming from the goal. It is worth noting that individual applications may differ as the standard is flexible. GSN is a graphical notation used to present safety cases in a manner that aims to minimise ambiguity and avoid poor writing. Safety cases are often formed by hand or using tools [57], so a tool for (partial) automatic safety case generation would be beneficial.

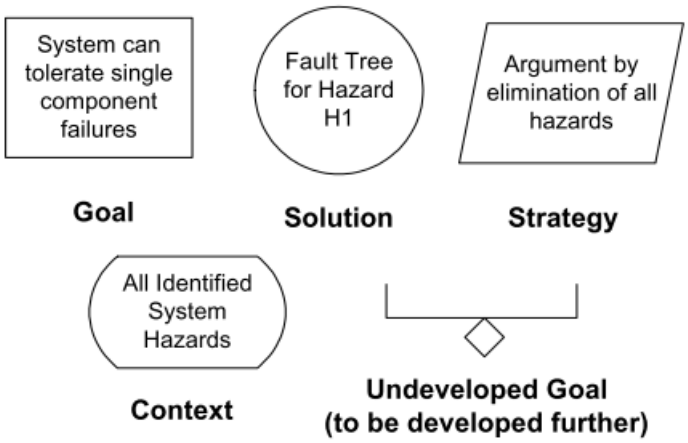


**Figure 2.5: GSN nodes and their purposes. Taken from [1]**

Since both GSN and EBL explanation structures are goal structured and attempt to tie claims to evidence, there is a parallel between their representations. This parallel forms the basis of an argument for the use of EBL in aviation. It is plausible that EBL can be used to elicit both control laws and (at least skeletal) safety cases, a suite of which could form a larger argument. There are some advantages to this approach, such as highlighting potential flaws in a domain theory by presenting justification for rules that have additional motivational information - that of safety.

If these generated arguments could be verified at runtime, then it may be possible to argue the safety of introducing a new rule in a manner equivalent to a human-derived argument. Equivalence to human-produced safety cases is fundamental to current approaches to establish certification guidelines for UASs [52],[53]. Non-adaptive elements of a system can be certified using the well established standards-based approach. Adaptive safety cases can potentially present an argument justifying their control laws. The initial domain theory can be used as a baseline, for comparison with the system after learning occurs. For any later state the adaptive system reaches through learning, where a verifiable safety case can be constructed, it seems reasonable to argue that the overall argument holds. The validity of a safety case can be established, possibly by a static checker or mathematical assessment. This argument is presented in [58], which also reinforces some of the views expressed in [59], [60]. It is worth noting that if this argument holds it would apply to more than just the technique discussed, i.e. the EBL; but it could potentially be applied to other human-transparent techniques such as rational agents, other rule-based techniques, and fuzzy logic controllers to name a few.

Changes to an adaptive system's behaviour potentially require recertification. This can prove to be a large overhead. This issue suggests that the ability to operate an adaptive controller off-line would be an advantage. Learning could still occur as long as the situation-action bindings are not altered without recertification.

Tools exist to help with the construction of safety cases [57]. AdvoCATE is a tool which aims to automatically generate and assemble safety cases [61]. AdvoCATE is capable of integrating heterogeneous elements to form GSN structures [62]. Safety case fragments relating to software can be generated using formal methods. However, this approach may not be ideal for considering fuzzy rules. A reason for this is that fuzzy rules contain no contextual information, such as information relating to their derivation. Generating fuzzy rules and safety case fragments using EBL could provide this additional information.

There are two main benefits to using EBL as a basis for the generation of safety cases. Firstly, the generation of safety cases is useful in itself. Safety cases are usually generated manually and therefore the process can be time consuming and error-prone [62]. Automation could alleviate these issues and it is the focus of the work in [62], which brings the automatic generation of safety cases incrementally closer. Secondly, only the domain theory need be certified. The EBL algorithm itself needs to be certified, but not for every application. The domain theory may well be smaller than a more traditional software solution.

## 2.6 Conclusions

This thesis addresses a gap in the literature. This is the issue of managing of a modular hardware platform, specifically with regards to aviation. This work concentrates on the control and certification of adaptive control on an all-electric IMA UAS.

When considering the extension of EBL to include analogical reasoning there are a few issues which remain. Whilst there are similar works for techniques such as case-based reasoning [63], [44] , these systems tend to use only analogy for reasoning. In this work, EBL will primarily be used deductively, to generate fuzzy rules. However, changes in hardware motivate the additional need for transfer learning. There is an extension of EBL to perform analogical reasoning when a deductive option is possible with the combination of domain theories [26]. Correspondences in language can be exploited to infer analogical terms. The terms deemed to be common are used to produce a structure which is a combination of both domain theories, related to a particular goal. Generalisation of this structure can be used to extend a domain theory. In [26] deductive explanation and analogical reasoning are run simultaneously. The work in this project is positioned differently as deduction remains the primary focus, and there is only one domain theory. This guides the research down a certain path, namely the extension of EBL to include analogical reasoning.

Fuzzy-EBL integration is similar in concept to [32], where explanation is used to elicit new fuzzy rules. However, there is a dearth in the literature relating to the integration of fuzzy controllers and EBL. As UASs are the target platform for energy management, the integration of two human-readable techniques is advantageous, as it aids certification. Additionally, the integration of two techniques - one with a greater wealth of contextual information, one with a lower computational overhead - is appropriate to the target platform. Therefore, a second research topic is to integrate EBL and fuzzy controllers, appropriately to the target application.

The literature relating to certification, within the aviation domain, highlights issues with the current approach when applied to adaptive control. There are notions such as just-in-time certification [59] which would benefit from automation. There is some existing work in automatic safety case development, as in [64]. However, in this case the expressive power of EBL could be extended so that skeletal safety cases are generated by the same technique which derives the actual control laws. How to achieve this is the final research question.

Several research topics have been elicited from the body of existing literature. The following chapters will deal with achieving these propositions.

# Chapter 3

# Explanation-Based Learning for Fuzzy Rule Generation

## 3.1 Introduction

The first objective of this project is to investigate the integration of EBL and fuzzy controllers. The fulfilment of this objective commences by proposing an approach to integrate the two techniques. The ability EBL has to generalise knowledge can be used to derive specific fuzzy rules from a more general domain theory. This allows a small domain theory to generate rules appropriate to many situations. By utilising generalisation in this way, a general strategy can be used to generate rules which are platform-specific. When changes are made to a platform a new set of rules can be generated without changing the domain theory. Rule generation is an alternative to manually altering a controller to adapt to a change in hardware, which partially fulfils the aim of this project. A reduction in the effort required to alter a control system to a changing platform is the aim of this work.

This chapter proposes an approach for reducing the workload engendered by the management of a modular system: the integration of EBL and fuzzy control. The fuzzy controller is concerned with rule execution. EBL provides for fuzzy rule generation.

As the controller(s) are rule based, the rules may be specialisations of more general strategies. The general strategies themselves may remain appropriate when the need for adaptation arises. A change in hardware may only require that specific rules be changed whist employing the same strategy.

Fuzzy controllers are often expert systems, incorporating expert knowledge through human-derived rules. Fuzzy rules may also be generated by a learning technique [31]. If the rules are generated by a learning technique, the fuzzy controller may not include expert knowledge. EBL helps overcome this limitation. EBL incorporates expert knowledge in the form of a domain theory. Since EBL incorporates expert knowledge, the fuzzy rules generated by it implicitly embody such knowledge.

The domain theory is a collection of rules, embodying the knowledge of the system, which EBL connects to form explanation structures. The rules are specialised, by binding variables within

clauses to input values. This specialisation acts as a search bias, which restricts the search to certain rules when many may be possible. Since EBL constrains its search space using specifics (variable bindings) and expert knowledge (domain theory), then explanation structures are generated from the specifics of situations that are encountered. These explanation structures can be generalised to form new rules. These rules, derived from such explanation structures, match the specifics of platform, given its inputs, to the situations it encounters.

Rule generation and execution are separated. The separation abstracts execution details from rule derivation, such as which crisp values map to a linguistic variable. By abstracting the execution details from rule derivation, the same derivation can be specialised to elicit multiple rules.

An approach to facilitate the interfacing between EBL and fuzzy controllers is suggested to allow their integration. This is akin to establishing a way for both techniques to interpret a shared vocabulary, each in the manner appropriate to its purpose. An issue with interfacing EBL and fuzzy controllers is the communication of inputs, from the platform to EBL. Inputs are fuzzified to map the crisp values onto linguistic variables. These linguistic variables can interpreted by both techniques. Interpretation allows EBL to incorporate linguistic variables into its domain theory, rather than the greater range of crisp values. The fuzzy controller interprets references to linguistic variables, in rules generated by EBL, as normal. This means that input and output can take a crisp form without being required within EBL.

Linguistic variables allow the consideration of a range of values. Each input will have a crisp value. Each input will be described in the fuzzy system by a series of fuzzy sets and membership values. These membership values are calculated by membership functions, each one defining a fuzzy set [29]. The distribution of fuzzy sets, to describe an input, is usually expert knowledge [31].

The fuzzy sets which define linguistic variables have their membership functions interpolated. Interpolation allows one set of linguistic variables to be applied differently to any range of values. The interpolation of these linguistic variables over any range of values limits the vocabulary required to describe any input value. Interpolation allows for hardware providing different input values to share the same linguistic variables and therefore control strategy, possibly without further derivation.

The interpretation of each crisp value for comparison could be time consuming if the relationship between each number had to be enumerated or calculated. The comparison of a small number of linguistic variables could be a simpler task. The shared vocabulary allows EBL to reason about the relative orders of magnitude of different inputs. In using the same fuzzy set names to describe the magnitudes of all inputs, EBL can reason about the relative size of the terms without needing to convert between units or deal with the different scales used by the crisp inputs. A large altitude for an aircraft might be 10000ft, whereas the indicated climb might be large at 90 degrees. However, both could be labelled as being *large* and when both are *large* it might be appropriate to take some action. In this way disparate inputs can be easily compared.

Fuzzified inputs have corresponding statements in EBL and are considered to be operational. Operational nodes in an explanation structure are nodes which are allowed to be leaves. This originally came from concerns about performance [27]. It is possible for explanation to continue to the point that the rules derived have a net negative effect. That is, the rules take longer to generate and execute than the time saved by utilising them. This was initially a concern when using EBL as a technique to speed up other algorithms. In this work system inputs are viewed

as the point at which explanation may end. They are considered suitably true to forestall further explanation.

The chapter is structured as follows: Section 3.2 describes the architecture for rule generation and execution, as well as outlining the issues to be further expounded upon in later sections. The manner in which input data is marshalled by the fuzzy system, for use by the rule generator, as well as the adaptation of fuzzy sets over time are proposed in Section 3.3. Section 3.4 considers the conversion of EBL outputs into fuzzy rules. The system is applied to two different platforms, in simulation. The application is presented in Section 3.5. The chapter concludes with Section 6.5.

## 3.2 The Architecture

The separation of rule generation and execution is advocated. The separation supports the management of a modular environment by taking a general management strategy then specialising it based on the situations experienced. Fuzzy controllers are proposed for rule execution, EBL for rule generation. Fuzzy controllers will execute rules on the UAS. EBL will generate fuzzy rules on the ground station. Two concerns to interfacing the techniques are prevalent. Firstly, inputs need to be communicated between the fuzzy controller to EBL. Secondly, a method for the production of fuzzy rules from explanation structures is required.

The communication of inputs from the UAS, simulated or otherwise, to EBL needs to be considered. The crisp inputs from the simulator need to be encoded, or fuzzified, for interpretation by the fuzzy controller. For example, the input representing altitude might have a crisp value of 800m but be represented as a linguistic variable. Altitude might have the linguistic values *low 0.6* and *high 0.4*. The membership values, 0.4 and 0.6, represent how much the crisp value embodies the fuzzy set. This results in the crisp input being described by membership values for each fuzzy set. In order for EBL to generate rules it will require these inputs. This is because fuzzy rules will need to link inputs to outputs in linguistic terms so this information must be available to the EBL system.

Each input variable is described by a series of fuzzy sets, each with a membership function. These membership functions take a crisp value and output the degree of membership the value has for a corresponding set [29]. An approach to reporting fuzzified inputs to the rule generator is required i.e. setting the value of the input to be the name of the fuzzy set with the highest membership value then converting to a predicate logic syntax, such as First Order Logic (FOL). As an example, the altitude input, reading 800m, is first fuzzified into *low 0.6* and *high 0.4*. This can be converted into FOL: *altitude(low)* by using the fuzzy set with the highest membership value.

One consequence of making the inputs available to EBL is that they can provide the inductive bias which guides the explanation process. Different situations are described by the state of the inputs. The situations encountered restrict the explanations and therefore the fuzzy rules which are generated.

To ensure that these shared terms have meaning for all inputs one linguistic variable can be used to describe every input. Each input may have a different range of crisp values. In order to share the same linguistic values across any range of crisp inputs the fuzzy sets can be scaled whilst maintaining a pre-established distribution. In this way rules can continue to be re-interpreted to function over a changing flight envelope. Consider a rule whose precedents note a high altitude. If

the UAS were adapted between missions then it may be able to operate at higher altitudes. This would mean that the interpretation of high altitude either needs to scale to the new flight envelope, or risk triggering at a prematurely low altitude.

Scaling the inputs brings some intuitive advantages in adapting existing rules to a changing environment and hardware platform. Existing rules are automatically reinterpreted as the membership functions take account of a changing range of crisp values. Re-interpretation of control laws could negate the advantages that they were to bring if large changes apply the rule in states that differ from the intent but share the same linguistic values. This suggests a non-monotonic learning approach in order to continually assess existing rules to find ones that are interpreted in a manner detrimental to their original intent. This can be avoided by establishing the fuzzy set distributions before allowing rule generation, which is the case in this chapter. However, this adaptive capacity could prove to be a certification barrier. These issues could be avoided by simulation of the new flight envelope, such that no adaptation takes place during flight. Simulation is also advocated for updating control laws between hardware configurations, to avoid online adaptation.

Rule generation using fuzzified inputs needs to be considered. Given a goal, EBL will use a domain theory to link evidence, or operational statements, to a goal. This forms an explanation structure, often a tree. The tree has the goal as the root node, domain theory rules form the body of the tree, and leaf nodes are nodes defined as operational. In this case fuzzy inputs can be viewed as evidence and therefore taken to be operational. Nodes within the tree are usually linked by an inference method, such as unification by resolution [24]. Unification is the process of finding a substitution, between terms and variables, under which two expressions equate. If this is possible then the two terms can be linked in the explanation structure. Resolution, in this case, refers to the approach to unification.

The design of the domain theory needs to consider the states that the fuzzy inputs can take. The values which they are designed to take influence the variable binding process. The structures that can be formed by EBL are guided by the specific values of variable bindings. Domain theory rules, and indeed goals, may include variables. By including variables in the domain theory, different bindings can result in different explanation structures. For example, there could be a goal which is implied by two different rules. The rules themselves might specify different variable bindings for the goal which they explain. Therefore different bindings of the variables within a goal can result in different rules being used to infer said goal.

When forming explanation structures, the variables are bound based on the specifics given to EBL. These variable bindings propagate through the explanation structure. This process is visualised in Figure 3.1, which was presented in the previous section. Figure 3.1 depicts some basic reasoning about relative orders of magnitude. Two copies of the same explanation structure are shown. The top structure shows variable bindings, the bottom shows the unbound nodes. The terminal nodes from the top structure match the domain theory nodes when specialised. The bottom structure shows which nodes contain specifics and which, within the domain theory, contain variables and can therefore apply to multiple specific situations.

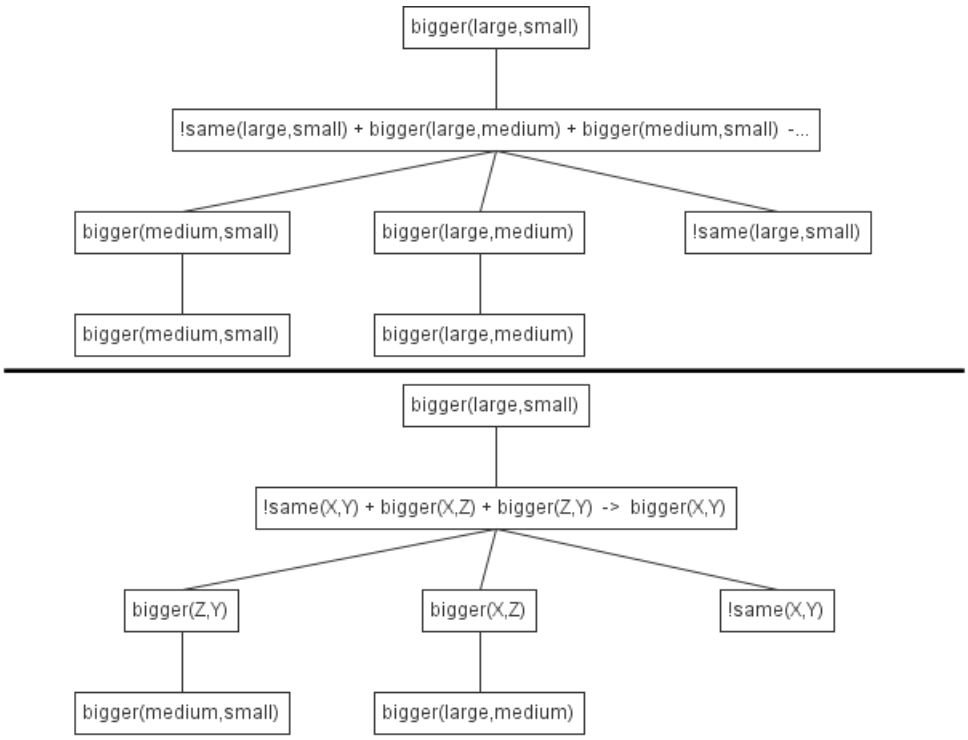


**Figure 3.1: Two explanation structures, the upper shows variable bindings, the lower unbound nodes. This figure has been repeated from Chapter 2.**

Figure 3.1 includes an example of negation in the node with the predicate *!same*. The implementation includes negation as failure. When a negation, denoted by '!', is encountered it is used as a goal for further explanation. The goal is taken without the negation symbol i.e. the non-negation. If the explanation fails then the negation is taken to be true and the original explanation continues.

EBL generalises the explanation structures into new, more general rules. This can simply take the form of undoing variable bindings [22]. Generalisation, in EBL, is often concerned with the sensible replacement of terms in a clause with variables [21]. In this way, specifics are abstracted from the structure. The specific, unabstracted, explanation structure can be used as the basis of a fuzzy rule.

Generating fuzzy rules from explanation structures has one major requirement. The system needs to be designed to assure that the necessary information to generate a new fuzzy rule is available within a given explanation structure. Information regarding system state needs to be linked to information regarding an output state. The goal is designed to represent an output. The leaf nodes are structured to represent an input state. The domain theory must be designed appropriately so that appropriate inputs link to appropriate goals. The fuzzy inputs are taken together represent a situation or state. If the goal is structured such that it contains enough information to describe an action then, taken with the fuzzy inputs, a fuzzy rule could be generated.

In this thesis the fuzzy rules take a form similar to the rules within the domain theory, with a conjunction of precedents and a single antecedent. More detail on the conversion process is given in Section 3.4.

EBL specialises variables within a domain theory in order to limit the search space of explanations. This means that the domain theory can be designed such that it can encapsulate general strategies,

which are converted to specific rules only when those situations are actually encountered. Simulation can control the situations encountered by EBL and therefore used to train the fuzzy controller by influencing the rules which can be generated. In this way simulation can be a powerful training tool.

## 3.3 Fuzzification of Input Data

The manner in which inputs are fuzzified and interpreted needs to be considered further. For example, an engine may be replaced with one that has a higher RPM. It may be the case that almost all crisp values reported, using the old engine's vocabulary, are shown as being dangerously high. This could lead to rules which worked on the old engine being used inappropriately on the new engine. An example could be continually slowing the RPM of the engine to the point that it no longer operates correctly, or efficiently. This continual reduction could be the result of an engine appearing to operate at an overly high RPM when it is just a characteristic of the new engine. It would therefore be advantageous if the sets reconfigured themselves rather than requiring the analysis, removal and re-derivation of applicable rules.

This issue can be addressed by interpolating the membership functions. Interpolation will result in the same linguistic values applying to slightly different crisp ranges based on the inputs being reported. When fuzzy rules are being executed the linguistic variables will automatically reflect a state appropriate to the inputs being recorded. A large value will change depending on the range of values encountered.

Maximum (max) and minimum (min) values are stored; these provide the kernels for the lowest and highest sets respectively. An even distribution of kernels for the other sets can then be interpolated. Given the kernels, and a pre-defined distribution, the membership functions can likewise be interpolated. This can apply to a variable number of sets, though a sensible upper limit of 5 to 9 sets was established in [65].

The distribution could simply be triangular sets prefixed and suffixed with trapezoidal sets. This distribution supports transparency [36], which can ease the difficulties in demonstrating determinism. Determinism is an important property in the certification process. An example of the distribution is given in Figure 3.2. If, through changes in input, the maximum and minimum values were altered then the sets would effectively be changed, shown in Figure 3.3. Both figures show 5 sets encompassing different ranges of crisp values. Values outside this range are encompassed in the outermost sets. Interpolation of these sets occurs to fit the distribution, such that the sets are meaningful. In this case meaningful sets define linguistic values which are proportional to the range of data they apply to. Another way to consider this notion is that a meaningful set is a set in the context of a particular input/output.

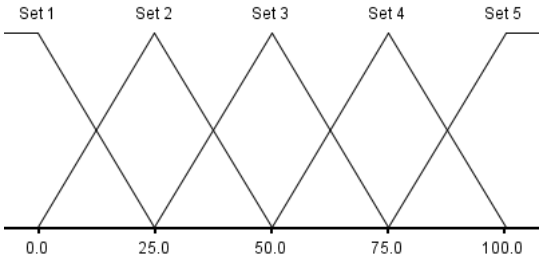


**Figure 3.2: Example of an interpolated fuzzy set distribution over the values 0 to 100. This figure has been repeated from Chapter 2.**

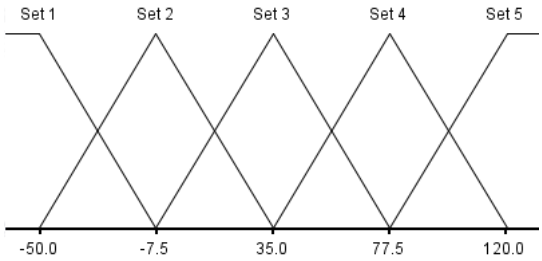


**Figure 3.3: Example of an interpolated fuzzy set distribution over the values -50 to 120.**

Given $n$ sets, each one denoted by $\{s_1, s_2, s_3 ... s_n\}$, the distance between the kernels ($m$) is:

$$m = \frac{max - min}{n - 1} \tag{3.1}$$

The kernels can be obtained by adding $m \times n$ to the minimum value. As the sets all overlap, $m$ is also the width of each membership function ($\mu$). Triangular sets are therefore given as:

$$\mu(x) = \begin{cases} 0, & \text{if } x \leq a \\ \frac{x-a}{k-a}, & \text{if } a < x \leq k \\ \frac{b-x}{b-k}, & \text{if } k < x < b \\ 0, & \text{if } x \geq b \end{cases} \tag{3.2}$$

Where $k$ is the kernel of any particular set and $a$ is the point on the distribution that membership first rises above 0. The point where the membership value returns to 0 is denoted as $b$. The crisp input value is $x$. The initial trapezoidal set differs by having no $a$ value and the membership function is therefore:

$$\mu(x) = \begin{cases} 0, & \text{if } x > b \\ \frac{b-x}{b-k}, & \text{if } k \leq x \leq b \\ 1, & \text{if } x < k \end{cases} \tag{3.3}$$

The final trapezoidal set likewise has no $b$ and is defined by:

$$\mu(x) = \begin{cases} 0, & \text{if } x < a \\ \frac{x-a}{k-a}, & \text{if } a \leq x \leq k \\ 1, & \text{if } x > k \end{cases} \tag{3.4}$$

In calculating the minimum and maximum values, a naive approach might be to simply take the largest and smallest encountered values and use these as the bounds. However, anomalous data could severely curtail the usefulness of this. A large outlier could cause all of the actual inputs appear to be too small. This could be alleviated using an averaging technique, keeping a buffer of the highest and lowest X results to report the average. However, in order to make the average converge i.e. discard anomalous results, data ($q$) is allowed into these buffers based on the range of previous values:

- The range of values in the buffer, $r$, is calculated as follows: $(max_b - min_b)$
- For values to be added to the minimum buffer, i.e. values to be considered when calculating the average minimum: $q \leq min_{avg} + r$
- For values to be added to the maximum buffer, i.e. values to be considered when calculating the average maximum: $q \geq max_{avg} - r$

The subscript $avg$ refers to the previously calculated average maximum or minimum value for its respective buffer. The subscript $b$ refers to the maximum and minimum values that can be found in the buffer.

Values are allowed into a buffer when they are within a range of the average, this range is itself determined from the buffers. This allows values smaller than the anomaly to be added to the buffer, rather than just adding values larger than the maximum. Enough examples will result in the anomaly being replaced by normal data, as the buffer is of a limited size. This causes the average to converge as the range decreases, until the range is 0 so only values larger than the average are accepted. This process applies to both buffers conversely.

Using range to control entry into the buffers is acceptable for capturing the inputs from controls with pre-defined ranges. An example is throttle position in a UAS. The unidirectional alteration of the buffers is suitable for establishing the flight envelope as, for a specific platform, the range of use is predetermined. The technique seeks only to establish the flight envelope and to keep inputs proportional. If the aircraft were operated at its maximum altitude once, then at half altitude for a long time, it is not reasonable to scale the inputs down to half altitude as the aircraft maintains the possibility to fly over its entire flight envelope.

Using anomalously large data to expand the range of values allowed into the buffers has a limitation. Opportunities to identify anomalously small maximum, or anomalously large minimum values, may be missed. These instead get classified simply as data within the existing distribution, rather than being used in set interpolation, via the reclassification of the maximum and minimum set values. This process occurs on each UAS individually.

An example of the proposed method is presented in Figure 3.4. In the example the buffer size is 10 values, all of which are expected to be 100. This might represent the percentage of throttle

requested. It is also assumed that the data can have anomalies. In this case it is large anomalies that are of greatest concern, as the examples focus on the maximum buffer, which disregards anomalies below the buffer values. The aim of the buffer is to attempt to converge on a value which represents the likely maximum data value. The value is permitted to be underestimated, as inputs above the maximum will simply fall into the linguistically largest fuzzy set. It is, however, intuitive that this set should be situated near the likely maximum.

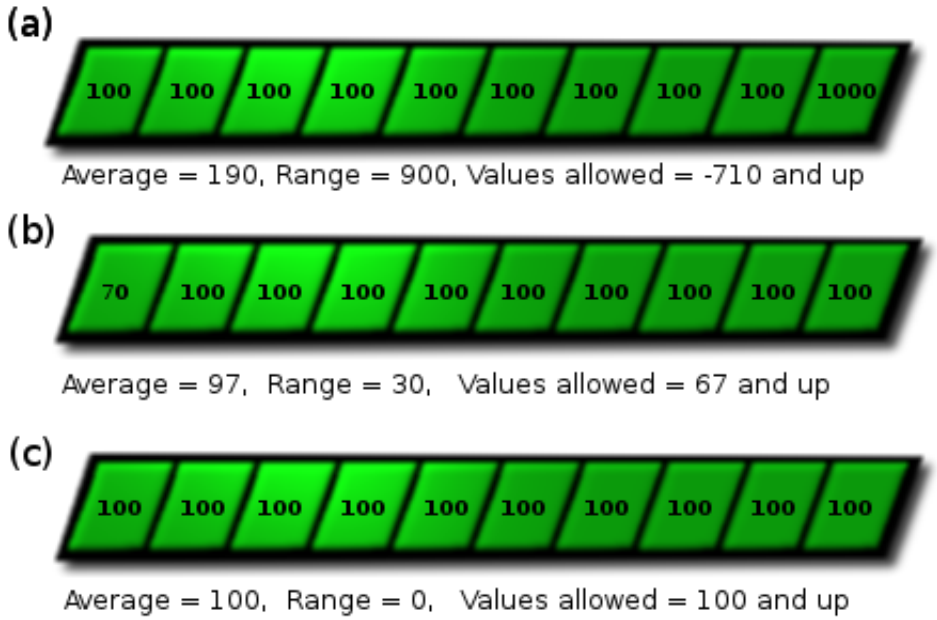


**Figure 3.4: Some example buffer states. Values are added to the left. When full, values are removed from the right.**

As shown in Figure 3.4 (a), the buffer is initially given a large outlier with the value of 1000. After this, it is filled with values of $q = 100$. Note that the large outlier is not immediately discarded, but its influence is relative to its frequency. This makes a change in hardware, to one with different input ranges, affect the interpretation of fuzzy rules by altering the maximum value. The large outlier also affects the range of values that are now allowed into the buffer. As can be seen in Figure 3.4 (a), the buffer will now consider a wider range than the data is likely to have.

The throttle is reduced and a value of $q = 70$ is input, as shown in Figure 3.4 (b). The effect of this is twofold. Firstly, the previous outlier is pushed from the buffer. This drastically reduces the range of values to be considered. Secondly, the value of $q = 70$ is now being considered when calculating the maximum. Ideally this would not be the case, as the value of 70 is not a member of the high input set, but is classified as a regular input. The range drastically reduces and so too do the values allowed into the buffer.

A value of $q = 50$ is encountered next, but discarded as regular input data, not being relevant to the maximum. This is because the range does not permit the interpretation of $q = 50$ forming part of the high input set.

Afterwards, a large number of inputs at $q = 100$ occur. This leaves the buffer containing only this value, see Figure 3.4 (c). Input values at or above $q = 100$ are now the only values that enter the buffer. Input values below this are not considered relevant to the maximum. The buffer now

has established a reasonable maximum and will not alter unless a value larger than $q = 100$ is encountered. After convergence, large values need to occur in order to admit smaller values to the buffer. In this way the buffers become more explorative depending on the deviation of the inputs.

As the fuzzy sets are interpolated and assumed to be the same width, nuances of the system being controlled may be overlooked. In a non-linear environment it may be impossible to converge on optimal behaviour with rules which assume a linear categorisation of inputs and outputs. For example, a smaller change in turn rate might be appropriate at higher speeds. The output linguistic variables, if linear, could result in an inappropriately high increase in turn rate at high speeds. Additionally, the fuzzified inputs are used to generate rules for the fuzzy system. Therefore inputs which would be better considered non-linearly could be inappropriately categorised. The range for a high input could be smaller than the range for a medium input. The detection of a state which should ideally be distinct, needing its own control strategy, could be missed by being assigned to too broad a linguistic value. The fuzzy rules generated for the perceived state may apply better over most of the range of crisp values encompassed, but perhaps not over the entire range.

When hardware changes its input values may also change. In this case, rules may not need to change since the interpretation of the linguistic variables may account for the difference. This assumes a certain superficiality in the differences between the two pieces of hardware, or at least in their management. This kind of generalisation is made possible because the inputs are interpreted using relative orders of magnitude, rather than specific values. The generation of new fuzzy rules may not be required in these cases. The technique may therefore be beneficial when applied to the control of a modular environment.

When inputs are reported to the rule generator input data is fuzzified and the set with the highest membership is reported. Each membership function covers a range of crisp inputs. These inputs change in sequence; altitude rises through each of the fuzzy sets in turn during a climb. Given that inputs are transmitted to the rule generator with sufficient frequency, EBL then encounters each distinct state. These states are considered discretely in EBL as being separate concepts. The specific membership values are not taken into account when generating a new rule in abstract. Just the state and action are considered in isolation. The rules, when executed, are applied with regard to the membership values at a specific time. This is because of the defuzzification process which takes a linguistic value and generates a crisp output. In this way the disparate rules generated by EBL have their effects combined and executed in a fuzzy manner. This maintains the advantages of the fuzzy controller as long as EBL has each distinct state reported to it.

## 3.4 Fuzzy Rule Generation

EBL is used to link the inputs to the output state. The inputs are fuzzified and reported in an appropriate syntax and the goal represents a particular output. A similar assumption to the one EBL uses to generate new rules is employed: the goal of a structure can be implied from a conjunction of its leaf nodes. This is like reversing Modus Ponens. Normally, where $A \rightarrow B \wedge C$ one can replace $B \wedge C$ with $A$. If applied to an explanation structure this would be akin to taking just the goal and its most immediate child nodes as the rule. This would likely be an overly general approach, resulting in rules which require greater effort to continually re-explain. Since the goal of EBL is reformation learning, Modus Ponens is employed in reverse, taking the greater number of antecedents and replacing consequents with them. This process is like taking $A \wedge B$ at each line of

the structure, the end result being that only the endmost nodes remain. This reformation is then stated by forming a new rule where these leaf nodes imply said goal. This process is applicable because the nodes in the tree are connected by a valid form of inference, the resolution principle [24].

The goal represents an output state. If a chosen goal contains the appropriate information then the collapsed explanation structure can be used to form a fuzzy rule. The goal needs to refer to the hardware to be manipulated and the fuzzy set that it should be set to. For example, a goal could be *reduce(throttle,N)*. During explanation EBL can specialise the variable $N$ in order to elicit the new value for the throttle to be set at. The particulars of the domain theory will determine how $N$ gets specialised, as will the particular inputs linked to this rule via the domain theory.

Additional information is needed to generate fuzzy control laws. The other information required to form a new fuzzy rule is the set of inputs, and their values, that describe the state in which an action is applicable. The inputs form the fuzzy rule precedents. When converting the inputs into an appropriate syntax, such as predicate logic, information can be introduced to distinguish inputs from other statements in the domain theory. An example: if the throttle is *large* then this could be marshalled as *reporting(throttle, large)*. In this example, the predicate *reporting* is used to denote any statements in the domain theory that originated as fuzzy inputs. These can be replaced, in the domain theory, as they are updated. Algorithm 1 describes the process of converting an explanation structure into a fuzzy rule.

**Algorithm 1** Generate Fuzzy Rule(struct)

**Require:** struct = an explanation structure
**Require:** the goal of struct adheres to a pre-decided form, in this case: goal(X,Y)
**Require:** the leaf nodes of struct adhere to a pre-decided form, in this case: requires(X,Y)
**Require:** X is the name of a system input
**Require:** Y is a fuzzified input value
  leaves = the leaf nodes of struct
  goal = the goal of struct
  rule = "IF"
  previousXs = empty set (previousXs is a set of the Xs which have already been processed, duplicate Xs in the precedent are to be avoided in a fuzzy rule though the same node may appear multiple times in an explanation structure.)
  **for all** leaf nodes in leaves **do**
    **if** leaf is a fuzzified input **then**
      Xn = X from leaf
      Yn = Y from leaf
      **if** Xn **not** in previousXs **then**
        add "Xn IS Yn" to rule, maintaining the format: IF X IS Y AND X2 IS Y2
      **end if**
      add Xn to previousXs
    **end if**
  **end for**
  Xgoal = X from goal
  Ygoal = Y from goal
  add "THEN Xgoal IS Ygoal" to rule
  **return** rule

An example concerned with reducing the throttle setting in a UAS is presented. The goal chosen is *reduce(throttle, N)*. The predicate *reduce* should feature in the EBL domain theory which, via inference, should link to operational nodes. Fuzzified inputs are considered operational. The domain theory can be defined such that the $N$ variable in the goal will be specialised to the value

that the throttle should be set to, as in Figure 3.5. The specialisation enables the goal predicate to contain the required information to define an output state, whilst remaining general enough to apply to more than one situation.

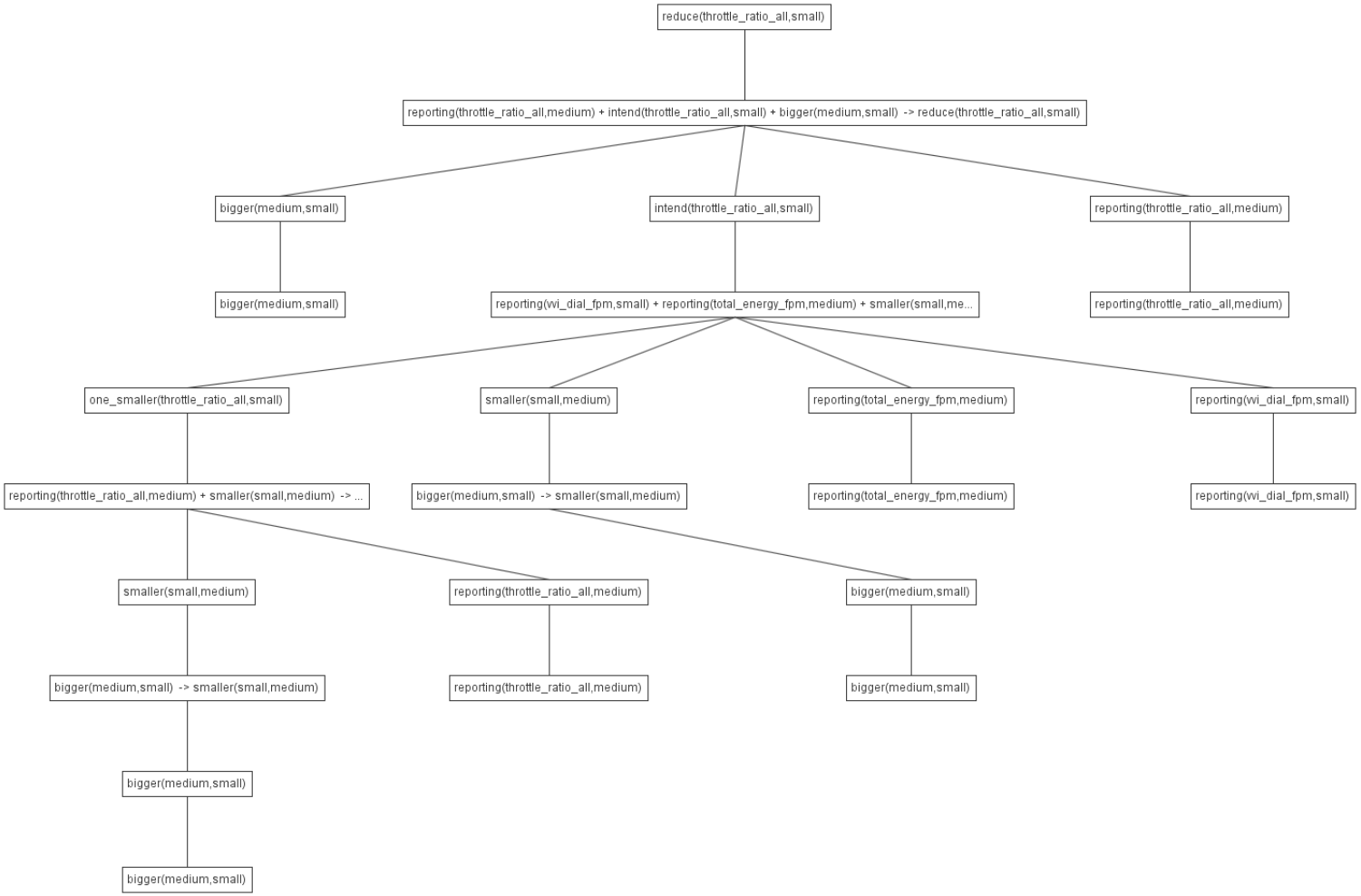


**Figure 3.5: An example explanation structure. For visual clarity some rules have been shortened to fit the figure.**

Selection of the value to specialise $N$ to is achieved by the *one_smaller* predicate from the domain theory. This predicate uses some other rules for reasoning about relative magnitude, given in the list below. The name of a fuzzified input is used to determine which inputs current value should be considered. The leaf nodes underneath it will cause the $N$ variable to be specialised to the linguistic value below the current input value. This requires that the linguistic values can be viewed hierarchically within EBL.

- bigger(X,Y) $\rightarrow$ smaller(Y,X)
- !same(X,Y) $\rightarrow$ different(X,Y)
- !bigger(Y,X) $\rightarrow$ biggest(X)
- !smaller(Y,X) $\rightarrow$ smallest(X)
- !same(X,Y) $\wedge$ bigger(X,Z) $\wedge$ bigger(Z,Y) $\rightarrow$ bigger(X,Y)

These rules are backed by some statements, which can act as terminal nodes in explanation structures. None, tiny, small, medium, large, huge and full correspond to fuzzy sets in an interpolated distribution. These statements are given:

- bigger(full,huge)
- bigger(huge,large)
- bigger(large,medium)
- bigger(medium,small)
- bigger(small,tiny)
- bigger(tiny,none)
- same(X,X)

The explanation structure (Figure 3.5) can now be used in the formation of a fuzzy rule. The explanation is first collapsed to a single specific rule, before the normal generalisation EBL step occurs. The rule generated relates the climb (total_energy_fpm), in feet per minute, of the aircraft to the indicated climb (vvi_dial_fpm), in feet. The indicated climb is set to the height to go to reach the next checkpoint. A rule is generated when the energy of the aircraft, which indicates upward motion, is an order of magnitude above the desired altitude. When this case occurs, the throttle is reduced by an order of magnitude. This strategy reflects the assumption that the two factors are related and the upward motion of the aircraft, which is affected by the throttle setting, should be proportional to the desired climb. In this case: *reporting(throttle,small)* $\wedge$ *bigger(large,medium)* $\wedge$ *reporting(climb,medium)* $\wedge$ *bigger(small,tiny)* $\wedge$ *current(flight)* $\wedge$ *reporting(energy,large)* $\rightarrow$ *reduce(throttle,tiny).*

The convention implemented is that operational leaf nodes, identified by the predicate *reporting*, are the inputs from the fuzzy system needed to identify the fuzzy precedents. This information, along with the specialised goal, can then be converted to: *IF throttle is small and total_energy_fpm is large and indicated climb is medium THEN throttle is tiny.* This is achieved by identifying the

facts which relate to inputs, translating them into fuzzy precedents then translating the EBL rule's antecedent to serve as the fuzzy antecedent. This process is illustrated in Figure 3.6 and extracts information from predicates shared by EBL and the fuzzy controller

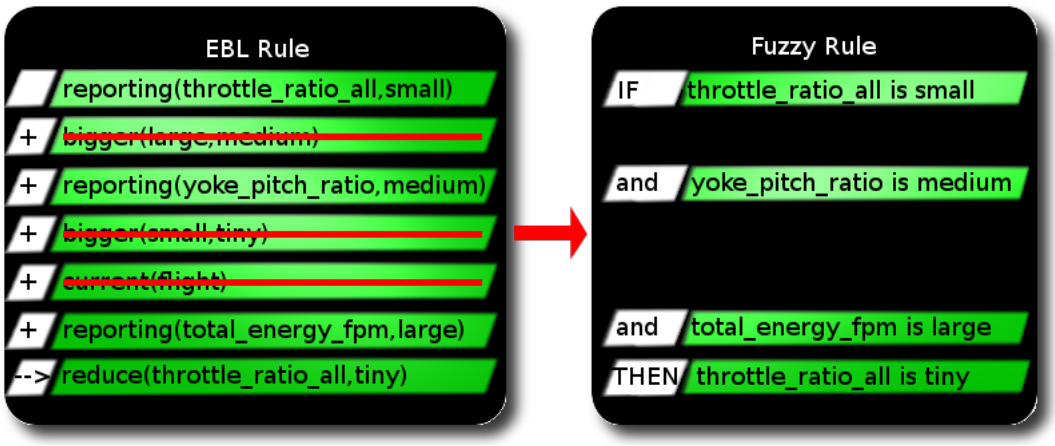


**Figure 3.6: Illustration of an EBL rule being converted to a fuzzy rule.**

## 3.5 Initial Application

This process, as per the example shown in Figure 3.6, was followed: inputs were fuzzified, fed into the EBL domain theory, goals were explained, and structures were converted into fuzzy rules. These rules were fed back into the fuzzy controller. The inputs were obtained from an aircraft simulator (X-plane). These rules were used to control a simulated aircraft. Rules for controlling the throttle were elicited over a short period and are given in Table 3.1.

Tables 3.1 and 3.2 show rules generated during a simulated flight. The rules relate the input, throttle requested, to the output value. In Table 3.1, the fuzzy rules that were generated represent the specialisation of a general strategy to an encountered situation, facilitated by EBL. As there are some rules which share throttle input and output values, but not each combination allowed by the domain theory, it can be inferred that this rule base is conceptually incomplete. Rules are only generated for the situations presented to the system, based on the expert knowledge encoded in domain theory. This is because rules are only generated for the situations encountered. This allows for simulation to be used to train the system in specific ways.

| **Input** | | | **Output** |
|---|---|---|---|
| **Throttle in** | **Energy (f min$^{-1}$)** | **Indicated Climb** | **Throttle out** |
| tiny | tiny | tiny | none |
| none | none | tiny | tiny |
| medium | none | none | small |
| medium | full | full | small |
| large | full | full | medium |
| huge | full | full | large |
| full | none | none | huge |
| full | full | full | huge |

**Table 3.1: Table of the fuzzy rules generated, in simulation, for an all-electric glider (EMG-5).**

| Input | | | Output |
|---|---|---|---|
| **Throttle in** | **Energy (f min$^{-1}$)** | **Indicated Climb** | **Throttle out** |
| full | full | large | huge |
| full | medium | medium | huge |
| huge | medium | large | huge |
| full | full | full | huge |

**Table 3.2: Table of the fuzzy rules generated, in simulation, for a Boeing 747.**

The flight contained periods of over-provisioned thrust whilst climbing and descending. The flights were human-controlled. The coverage of the rules is also shown in Table 3.1, by considering the states which feature in the generated rules. As can be seen in the table, not all possible rules were generated; only the ones driven by the specifics of the flight were elicited.

Using EBL to generate the fuzzy rules means that only rules which are appropriate to the use are elicited, rather than the whole flight envelope. A UAS may well have a large flight envelope but only be used at low altitudes due to mission parameters. This method limits the fuzzy rule base size by only generating rules when a situation is encountered, so reflects the situations which a UAS has been used in, rather than a single monolithic rule base containing all permutations. This gives additional information about what a UAS has previously done and, if examined as part of the certification process, could help to show the coverage of the controller during development. Additionally, the explanation structures contain contextual information, aiding their interpretation. Interpretation of what an adaptive controller has actually learned is an issue in aviation [50].

This same strategy, embodied in the domain theory, could be specialised for other UASs with different configurations. Deviations in, say, the range of numbers reported by the throttle is abstracted during fuzzification, since the fuzzy sets are interpolated.

In Table 3.2, a similar flight was conducted. Rather than simulating a small electric glider, the controller was applied to a Boeing 747. This is a much heavier aircraft with smaller wings, relative to its size. This change of aircraft is akin to an exaggerated example of altering a UAS between missions. The same domain theory produced different rules. This is shown in Table 3.2. All of the rules maintain a high level of throttle. The Boeing 747 may have less of an over-provisioning of thrust for steep climbs. The EMG-5 is aided in terms of climb by having larger wings which produce greater lift. Alternatively, there may be differences in the rules generated for other reasons, based on implementation issues. Also, the characteristics of each flight may have explored slightly different areas of the possible input space. However, this extreme case would appear to support the applicability of the approach to modular platforms as different rules were generated for each platform.

The convention used to express the goal is not the only possible option, but does include a minimal amount of information required to elicit the fuzzy antecedent. Other approaches could be adopted for other situations. Changing the number of variables, how they are specialised, and how the goal interacts with the domain theory, as well as how the inputs interact with the domain theory give

this approach additional flexibility.

However, adopting this approach increases the number of issues to be considered during knowledge engineering - the creation of the domain theory. Issues such as the representation of fuzzy inputs, the rules relating their magnitudes and the linking of goal predicates which can simply translate into a fuzzy antecedent. An additional issue is that information relating the fuzzy linguistic variables needs to be included in the domain theory to facilitate reasoning about their relative magnitudes. The final concern with this architecture is regarding the selection of goals, which may require a domain-centric decision. There are many options, such as agents [66]. Agents are concerned with the selection of an action, given an internal state. They may use techniques such as reinforcement learning, where behaviour which progresses towards an aim is favoured, to modify their behaviour. They also retain an element of random selection in order to remain exploratory enough to escape local minima. The actions an agent considers could be the goal to be explained next. In simple domains it may suffice to have a single predicate, at the top of every conceivable explanation structure, which could always be the goal. One alternative, used here, is for each goal to be continually explained concurrently. This suffices for a simple domain, but is prone to wasteful employment of computational resources. Future work into goal selection for more complex domains could prove fruitful.

The technique has been applied to throttle control in some simulated flights. The generation of different fuzzy rules for both platforms could indicate that the technique is applicable to modular control. However, the rules generated are not considered in terms of whether they are valid. In this case, validity refers to whether the rules generated actually represent the domain theory. Generation of rules do not necessarily mean that those rules are what was intended, much as not every question that can be constructed in English is valid to ask. Additionally, it may be the case that the differences in the generated rule bases are not due to platform differences. Another source of difference could be the route flown, therefore the situations encountered by each platform would be different. The results support the application of this approach in further experimentation.

## 3.6 Conclusion

The gap in the literature concerning the integration of EBL and fuzzy control highlights an area for further research. This area has been partially explored in this chapter. This resulted in the proposition of an approach. The aim of this approach is to reduce the workload which can arise from a change in hardware when managing a modular system. An approach for generating fuzzy rules, using EBL, has been presented. In order to generate the fuzzy rules two things were needed: sufficient information and a mapping between explanation structures and fuzzy rules. Fuzzy rules link an input state to an output state. Therefore, an explanation structure which connects input and output states for a single situation (inputs and outputs are bound to values) contains sufficient information to generate a fuzzy rule specific to that situation.

The communication of information between the fuzzy system and EBL was considered. A shared language is employed, which the fuzzy system and EBL interpret differently. This allows for fuzzified inputs to be used to guide EBL searches. In support of the shared language, the input sets are interpolated. This allows the same linguistic variables to be applied to different inputs, whilst mapping to different crisp values. Interpolation of the fuzzy sets could potentially accommodate for degradations in hardware over time. There may also be times when altering the interpretation

of the fuzzy sets, which determine when a rule triggers and what its effect will be, may avoid rule generation. These topics are potential areas for further research.

The proposed approach was applied to two platforms. Rules were generated which were different for each platform. A more controlled experiment is required to establish that these differences arise from the characteristics between the two platforms, rather than by being subjected to different situations during flight.

This chapter has presented an approach for satisfying the first objective of this project. Initial results confirm that different rule bases are generated for different platforms, given the same domain theory. This effect can be used to reduce the workload required by a software system controlling a changing platform. The reduction in workload comes from the ability to automate rule generation for a new hardware configuration.

Whilst an approach has been presented and considered, dictated by the concerns of controlling a modular aviation platform, it remains to address two issues. Firstly, it has not been ascertained whether the rules generated by EBL reflect the intent of the domain theory. Establishing this would sit in support of the validity of this technique. Secondly, this approach needs to be applied more rigorously to more than one hardware platform in order to consider whether the rules are appropriate for each. In essence, it is necessary to show that the technique is applicable to modular control by achieving the same goal on multiple platforms. The following chapter will present experimental data in response to these two issues.

# Chapter 4

# Explanation-Based Learning and Fuzzy Control for Modular Management

## 4.1 Introduction

The first objective of this work is to investigate the integration of EBL and fuzzy control. In order for this to be fulfilled, the approach proposed in Chapter 3 needs to be investigated. This chapter will use a simulated application of the approach to consider the approach as part of an MPMS.

There is interest in increasing the modularity of UASs. This is not just to enable the inclusion of newer hardware, but to customise a UAS for various roles appropriate to the mission at hand [19],[12]. This project is concerned with the control of a modular platform. The controller(s) may need to be changed to adapt alongside the hardware being managed. In a highly modular setting it would be useful for an energy management system to adapt to new hardware in a manner which reduces additional effort.

Fuzzy controllers are useful for application to a UAS for rule execution. This is because of low computational requirements and the fact that the rules are stored in a human-readable form. EBL is chosen because domain theory can embody general control strategies, which are specialised to form fuzzy rules. This allows common strategies to be applied to multiple, diverse, hardware platforms.

In Chapter 3 an approach for integrating EBL and fuzzy control was proposed. The chapter highlighted the need for further experimentation using this approach. The aim of this chapter is to address two issues which were previously raised. Firstly, to show that the specific rules generated reflect the general strategy. Secondly, to apply the approach to two platforms and establish whether the approach can reduce the workload which comes with managing a changing platform. The application to two platforms is proposed as an extreme example of a changing platform. Each platform contains the hardware to be controlled but the nature of each platform necessitates a different set of control rules.

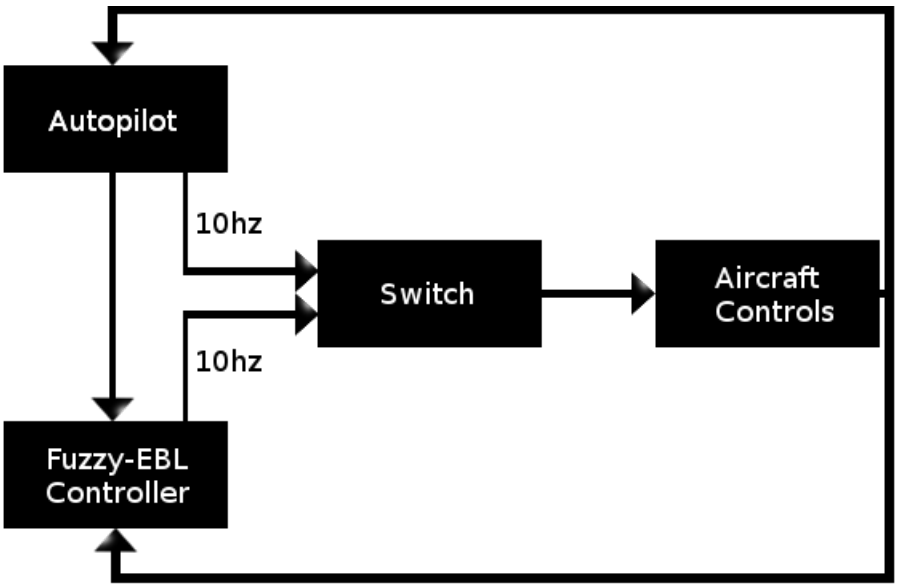


**Figure 4.1: The flow of information between controller, autopilot and aircraft is depicted in concert with the switching mechanism.**

Chapter 3 advocated the separation of rule generation and execution to generate many specific rules, which are platform specific, from a general strategy. There is another reason to separate the control strategies and specific rules. In the aviation domain, where software requires certification as part of a platform, adaptive controls require additional effort to certify [13]. The additional effort makes online learning an unlikely candidate for flight critical software. The separation of rule generation from execution could allow for, essentially, the generation of non-adaptive controllers. Offline learning could derive new rules. But changes to the controller could be examined and updated in batches. The controller could represent a snapshot of learning up to that point.

The system proposed in this chapter is applied to throttle control. As the platform considered in this chapter is all-electric, throttle control is correlated with energy use. The system in question, a fuzzy controller with rules generated by EBL, influences the aircraft and is itself influenced by both the aircraft and the autopilot system. The autopilot output is interleaved with the fuzzy controller output. This interleaving is performed by each system's input being taken at a given rate. The input which is taken at each time interval is communicated to EBL. The fuzzy controller focuses on a reduction strategy for the throttle control. As this strategy is reactive, rather than predictive, it benefits from opposition so as to avoid becoming overzealous. The way in which the autopilot, controller and aircraft interact is shown in Figure 4.1.

In the experiment presented in this chapter the fuzzy controller and autopilot are sampled at equal rates. An strategy to increase the throttle value was originally present in the domain theory. This was removed so that the effect of a single strategy, reduction, could be considered independently. However, the reduction strategy alone is not sufficient to control the throttle output. The autopilot is used to provide increases to the throttle setting which allow the aircraft to follow the route plan. The autopilot is intended to fulfil the role of the increase strategy and not overshadow the reduction strategy, which is why the output of each is sampled at an even rate of 10Hz.

The computational resources of a UAS are limited. Traditionally aviation favours the federated architecture, which is a collection of integrated but separate LRUs. Under a federated architecture, the computational hardware needs to be developed and included in each LRU [5], in order to main-

tain the separation of systems. Due to weight and energy limitations, each LRU will have limited computational resources. Even in an IMA system, as assumed to be the case for the purpose of this work, processing resources likely need to be carefully negotiated between disparate developers and the IMA system integrator [4]. The separation of rule generation, the more computationally expensive task, and rule execution is advocated to address this issue in part. It is reasonable to assume that a UAS has both a line of communication and a ground station with which to communicate, as in works such as [7] and [12]. Rule generation can therefore take place at the ground station, with rule execution on-board the UAS.

The generation of rules from a common domain theory is the main aspect of this approach which facilitates the management of a modular platform. Additionally, the fuzzy controller abstracts away from crisp input values in order to consider more general, linguistic, terms. These terms are reflected in the EBL domain theory such that there is a shared language linking the two techniques. The linguistic terms automatically scale across the range of a particular hardware without configuration. Consider the system being applied to another platform, with a different range of throttle values. Rules could still be generated for the new platform, even though the specific values differ from the original platform, because the general strategy remains the same.

The rest of the chapter proceeds as follows. The design of the experiments are presented in Section 4.2. An experiment is conducted with an all-electric glider and the results are presented in Section 4.3. An alternate platform, a passenger jet, is used in a second experiment and the results are presented in Section 4.4. The chapter concludes with Section 6.5.

## 4.2 Experiment Design

An experiment is proposed to consider the method in light of the objective for this chapter. The objective is to illustrate that the generation of fuzzy rules using EBL can reduce the workload engendered by a change in hardware. It has been posited that this objective is partially fulfilled by the interpolation of inputs interpreting rules differently based on the range of values encountered. In order to demonstrate that the integration of EBL and fuzzy control satisfy the objective two aspects are considered. Firstly, the production of valid fuzzy rules from EBL structures needs to be demonstrated. In order to show this, rules will be generated regarding throttle control. Secondly, if energy is saved by the execution of these rules then the intent of the domain theory will have been translated into fuzzy rules.

Additionally, the same experiment is conducted on a different platform. This is an extreme example of each item of hardware changing and having a significant impact upon the management of the system. The item of hardware being managed is present on both platforms. The rules generated should be different for this platform as the nature of the system being controlled has changed significantly. The production of different fuzzy rules which are also useful would fulfil the objective of this chapter. In order to be considered useful, the rules generated at this stage should reflect the intent of the domain theory. Demonstration of this would support the integration of EBL and fuzzy control with regards to modular control.

The approach presented in this chapter has been implemented and integrated with the X-Plane simulator. X-Plane has been used in other high-fidelity aerospace research, such as [53], as part of a broader simulation suite. In [53], X-plane provided the visualisation capabilities to consider the

certification of a simulated UAS. Version 9 of X-plane was used for this experiment. The throttle is the only parameter of the aircraft which is controlled by the proposed controller. The remainder of the parameters are determined by X-Plane.

An experiment was conducted in order to evaluate the approach presented in this chapter. A series of replicable flights were devised to test the controller. Each flight begins with a take-off, followed by a series of checkpoints to be hit. The run ends when the final checkpoint is reached. The checkpoints are placed such that climbing, descending and turning all feature. The autopilot flies a set route, input into a Flight Management System (FMS). The route is identical for every flight.

Each flight begins at Dunedin International Airport. The checkpoints are given as three components: latitude, longitude and elevation. The checkpoints are stated below:

- Latitude: -45.9154, Longitude: 170.229378, Elevation: 600ft.
- Latitude: -45.900417, Longitude: 170.2565, Elevation: 1500ft.
- Latitude: -45.914989, Longitude: 170.288429, Elevation: 1000ft.
- Latitude: -45.930991, Longitude: 170.306625, Elevation: 2000ft.
- Latitude: -45.948181, Longitude: 170.333061, Elevation: 2000ft.

The route followed is better visualised in Figure 4.2 below:

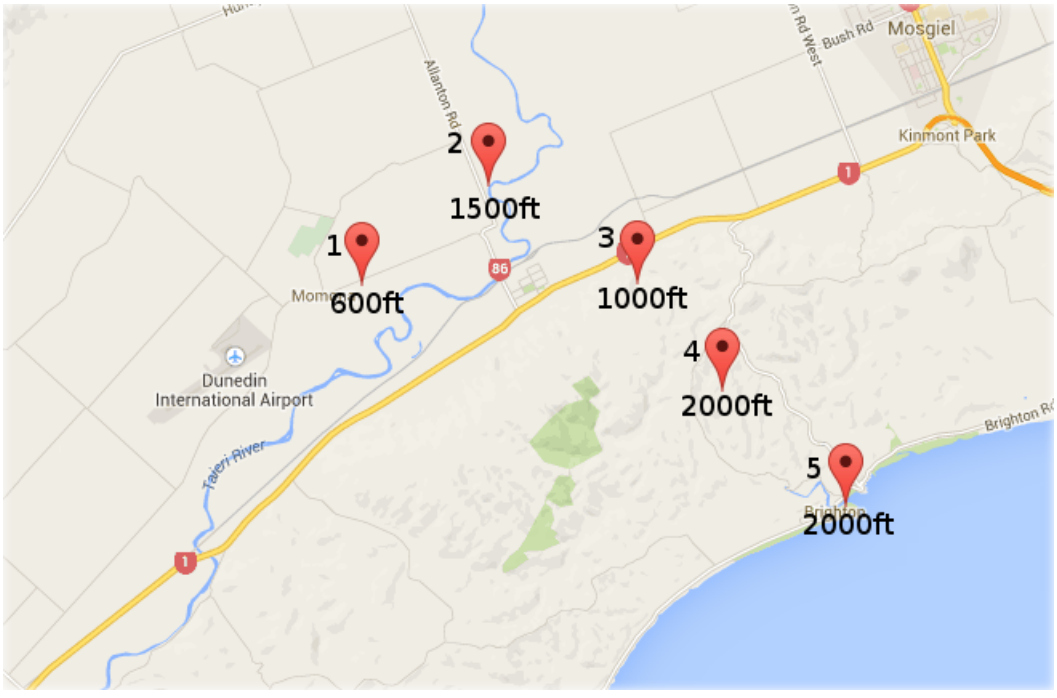


**Figure 4.2: The route input into the Flight Management System, followed throughout the experiment.**

Consideration of the parameters of the experiment is required. There are two sources of variation between the flights within a data set. The first is the throttle control value, which is altered by the proposed algorithm. This is allowed to vary over its entire value range [0-1]. The second source of variation is the simulator. The time of day is not bound which may affect temperature and

therefore air pressure. This can alter the amount of energy perform certain activities, such as take off. The weather was set to be 'clear' which limits the variability of the weather significantly which in turn limits the effect on the simulated flights. Weather includes the winds encountered and other sources of pressure variation. Limiting these effects limits the variation in forces encountered during. This maintains a level of consistency in the energy requirements of each flight. It may be possible to further limit the weather and time variability using X-Plane. The remaining variables controlled by the simulator are bounded by keeping the route fixed. The domain theory persistent between flights. A data set uses a single domain theory which is updated in each flight and passed to the next.

The autopilot controls the horizontal navigation to these coordinates. The throttle controls the elevation. An increase in air speed whilst maintaining an attitude will result in a larger lift force. As it is the auto-throttle which will be influenced by the controller, some deviation from the elevation of each checkpoint is permitted. This can allow for some degradation of performance to increase endurance; less energy is required to reach slightly lower elevations. However, this should not be taken to extremes as the UAS should still be able to perform as intended. The elevation reached may also depend on the width of the membership functions describing the required climb. The narrower the fuzzy set, the more granular the control. For example, the remaining climb in feet will be classified as *none* when reaching a certain boundary number. When this occurs, the aircraft will appear to have reached the intended altitude. However, the actual altitude will just be within a certain range of the exact value.

It is worth noting that, due to limitations with the simulator, elevation is not considered by the autopilot. When using the FMS, with autopilot, elevation simply increases. The vertical navigation (VNAV) system is not available at present:

"VNAV mode is a true vertical planning mode, as found in a G1000 or FMS. However, it is not currently possible to set up and fly VNAV paths with the consumer-level sim. VNAV is only supported in conjunction with the G1000 compatibility option. (In this case a real G1000 provides vertical navigation.)" [67].

This information was given by the community that document the Software Development Kit (SDK) for developing X-Plane plugins. Plugins are external files containing code which may integrate and interact with the X-plane simulator.

Furthermore, there is a system called Flight Level Change (FLCHG), which relies on the auto-throttle. FLCHG is used for vertical navigation in this experiment. The auto-throttle is in direct competition with fuzzy rules derived from the reduction strategy in the domain theory. For this reason, the domain theory supports no increase strategy. The competition of two competing factors prevents throttle reduction from becoming the overriding factor. Competition in this case supports the maintenance of performance of the aircraft in contrast to throttle reduction.

There is an issue with the auto-throttle being employed in this manner. If the auto-throttle built into X-plane were engaged then the controller output would be overwritten and the effect hidden. A separate auto-throttle was therefore implemented using the following information, given by the community that documents the SDK:

"When FLCHG is engaged, if the autothrottle is enabled, power is increased to 100% throttle (climb) or 0% throttle (descent)." [67].

As the implemented auto-throttle is not a predictive system, the altitude during level flight will fluctuate as the system reacts to crossing the level threshold. Additionally, these fluctuations will have uneven peaks and troughs because the lift of the glider will make climbing faster, especially as during climb the throttle is set to 100%. These concerns will magnify the effect of Phugoid motion. Phugoid motion is an oscillation in level flight caused by two competing effects: first, an increase in airspeed results in lift greater than weight and therefore a climb; and second, gaining altitude results in a loss of airspeed and lift, so the aircraft will descend. This effect is explained in detail in [68].

For a few seconds after take-off the controller is disabled. This is to prevent throttle reduction from interfering with, or even preventing, take-off. The enabling of the controller is based on the perceived state of the aircraft. EBL is used to determine the state of the flight. The states considered are: taxi, take-off, and flight. If the state is flight then the fuzzy controller is enabled, otherwise the controller is disabled. The states take the form of rules in the domain theory. These rules form explanations for the goal *state(X)*. The current state resulting from such explanation is included in the domain theory. These rules require some additional domain theory.

It should be noted that the *started* predicate refers to the first value reported, per run, for a fuzzified input. The domain theory for the experiment, as well as the goal which does not relate to state, is given in Table 4.1.

| **Goal** |
|---|
| reduce(throttle,Y) |
| **Domain Theory:** |
| **Rules** |
| ***numbers*** |
| bigger(X,Y) → smaller(Y,X) |
| !same(X,Y) → different(X,Y) |
| !bigger(Y,X) → biggest(X) |
| !smaller(Y,X) → smallest(X) |
| !same(X,Y) ∧ bigger(X,Z) ∧ bigger(Z,Y) → bigger(X,Y) |
| reporting(I,X) ∧ bigger(X,Y) → more_than(I,Y) |
| reporting(I,X) ∧ smaller(X,Y) → less_than(I,Y) |
| reporting(I,X) ∧ same(X,none) → none(I) |
| more_than(I,none) → some(I) |
| reporting(I,X) ∧ same(X,Y) → is(I,Y) |
| reporting(I,X) ∧ started(I,Y) ∧ same(X,Y) → unchanged(I) |
| reporting(I,X) ∧ started(I,Y) ∧ !same(X,Y) → changed(I) |
| reporting(I,X) ∧ smaller(Y,X) → one_smaller(I,Y) |
| reporting(I,X) ∧ bigger(Y,X) → one_bigger(I,Y) |
| ***reduction strategy*** |
| reporting(I,X) ∧ intend(I,N) ∧ bigger(X,N) → reduce(I,N) |
| reporting(height_to_go,P) ∧ reporting(climb,E) ∧ same(P,E) ∧ one_smaller(throttle,N) → intend(throttle,N) |
| reporting(height_to_go,P) ∧ reporting(climb,E) ∧ smaller(P,E) ∧ one_smaller(throttle,N) → intend(throttle,N) |
| ***state*** |

| |
|---|
| some(parking_brake) ∧ less_than(throttle,medium) ∧ none(altitude) → state(idle) |
| none(parking_brake) ∧ less_than(throttle,medium) ∧ none(altitude) → state(taxi) |
| none(parking_brake) ∧ more_than(throttle,small) ∧ none(altitude) → state(takeoff) |
| current(takeoff) ∧ none(parking_brake) ∧ some(altitude) → state(flight) |
| **Statements** |
| ***numbers*** |
| bigger(full,huge) |
| bigger(huge,large) |
| bigger(large,medium) |
| bigger(medium,small) |
| bigger(small,tiny) |
| bigger(tiny,none) |
| same(X,X) |

**Table 4.1: Domain Theory**

Experiments were conducted in the manner laid out, in data sets of size 50. The size of the data sets was chosen during development of the algorithm and domain theory. The following hypothesis is proposed: The size of the data set required to balance the deviation between flights is a product of the variability of the flight conditions. In the case of this experiment 50 flights are considered sufficient to balance the small deviations between flights. Deviations in flight characteristics may result in a new situation being encountered. A new situation in this case refers to a unique binding of input values. A new situation, if relevant, results in a new fuzzy rule. As the coverage of situations increases the generation of new rules slows. If the coverage of situations were complete then all of the variability would be represented in the results. The number of rules generated is therefore used as a guideline for selecting a data set size in this experiment.

If the variability of the system increased then size of the data sets would need to be sufficiently altered. This could be brought about by changes to the simulation settings, the simulator itself, the experimental setup or to the approach. Different rule bases could even require more runs through increased rule interactions.

Data sets were collected in order to test and assess the state of development of the algorithm. These data sets were discarded because they reflected the operation of an incomplete/incorrect system. However, a very low rule generation rate was always reached before 50 runs. A low rate of rule generation indicates either that all, or at least the most common, situations have been encountered. This is borne out in the results given in this chapter, which are now presented and discussed.

## 4.3 Glider Application

The altitude of the aircraft can be used to indicate to what degree the aircraft followed the selected route. This is because altitude is calculated as the height from the ground and so is affected by the terrain as well as how much the aircraft has climbed.

However, each flight may have taken a slightly different route, particularly when the throttle controller is operating. Deviations in weather and timing can affect the distance traversed between two checkpoints. Different speeds or lift characteristics will result in a slightly different path. For this reason, a total of 50 flights were conducted and the outputs averaged.

However, plotting altitude against time can still give an idea of the path taken. An average of the altimeter readings (altitude) without EBL is given in Figure 4.3. This serves to highlight the most common aspects of the flight when compared to a specific example, given in Figure 4.4. As can be seen, the flights begin with a climb, followed by a period of level flight. The level flight is perturbed, but distinct from the steep climbs and descents. The flight then proceeds: climb, level, descent, level, climb, level.

It is worth noting that most of the graphs in this chapter plot the standard deviation about the mean, by means of a grey shaded area.

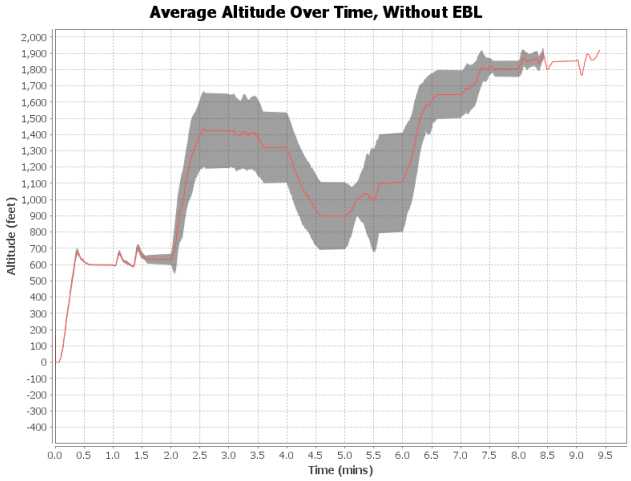


**Figure 4.3: Average altitude of the EMG-5 aircraft without EBL.**

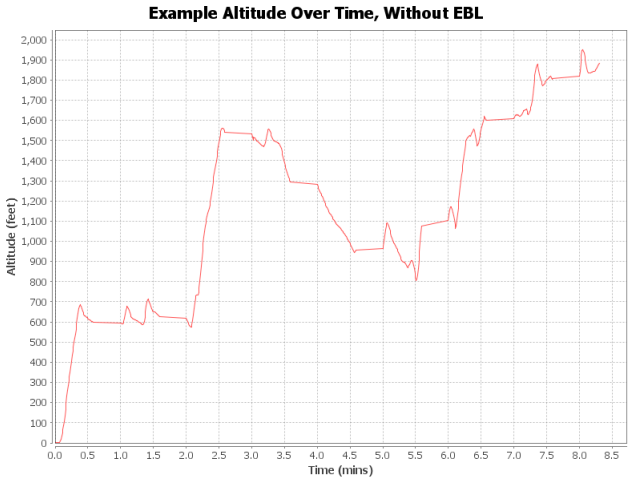


**Figure 4.4: The altitude of the EMG-5 aircraft over time from a single flight without EBL.**

In order to compare the results with and without EBL it should be shown that such a comparison is valid. Figure 4.5 is the counterpart to Figure 4.3 and shows the same information, but from flights with EBL enabled. Given that the same features are present in Figure 4.5 one can infer

that both result sets are comparable as it is likely that a similar route was indeed used.

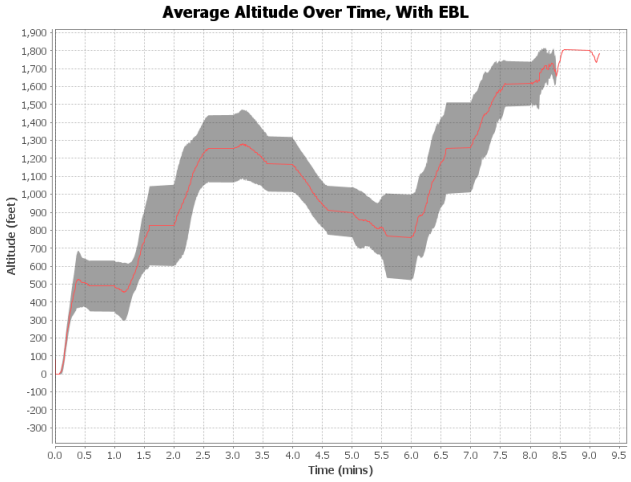


**Figure 4.5: Average altitude of the EMG-5 aircraft with EBL.**

Figure 4.5 also shows the aircraft taking more gradual ascents when the fuzzy controller is used. This is most apparent on the initial climb to around 1500 feet. A less steep climb could contribute to a saving in energy, as could the ascent stopping just short of 1500 feet.

The standard deviation can be useful in comparing runs with and without EBL. As seen, there is a fairly substantial standard deviation in either case. Weather and other non-linear, simulated, effects affect the autopilot. The standard deviation in Figure 4.3 is useful in illustrating this point. Whilst the standard deviations are comparable, both with and without EBL, it remains sensible to compare the results. This could highlight when changes have a large impact. A small point of different behaviour from the autopilot can influence the standard deviation from that point onward.

One explanation for the large standard deviations are that even small changes to aircraft behaviour, that delay the time taken between two given points, will affect all of the flight from that point onward. This is because the route followed is common to each flight in general. The effect of a small change will be temporally transposed. For example, if a steep climb starts earlier in one flight then the difference in values across the flights at this time will be large.

There is a second way in which the standard deviation is useful. The end points of the various runs can be indicated. The flight lengths vary; as each flight ends the standard deviation will be reduced. In this way one can identify, roughly, where the number of concurrent flights is reduced. This may be suppressed by other factors, such as common occurrences in the simulator, where the flights differ less. A length of time without change, occurring in each flight, could cause such suppression. It is somewhat unlikely that this would have a large effect, however. As the flight continues, the variance grows, in general, up to a point. This is because each flight is affected differently and the effect is temporally transposed.

It is worth noting that a degradation in performance (by following a lower altitude) was indeed used to reduce energy consumption. A slower completion time is also notable and implies that less energy was expended to complete the flight. Another reason to suppose that the controller affected the efficiency can be seen by considering the throttle setting on two particular runs. The throttle setting from a flight without EBL is shown in Figure 4.6. The same data for a flight with

EBL is given in Figure 4.7. It can be seen that without EBL the throttle is either at the maximum or minimum controller output, which is a unitless value in the range [0,1]. The exceptions are where the needs change part way through moving the throttle, which can give narrow or shortened peaks. However, there is more movement and a lower average throttle output value when using EBL. Both examples are ones that had a longer flight time compared to their relevant data sets.

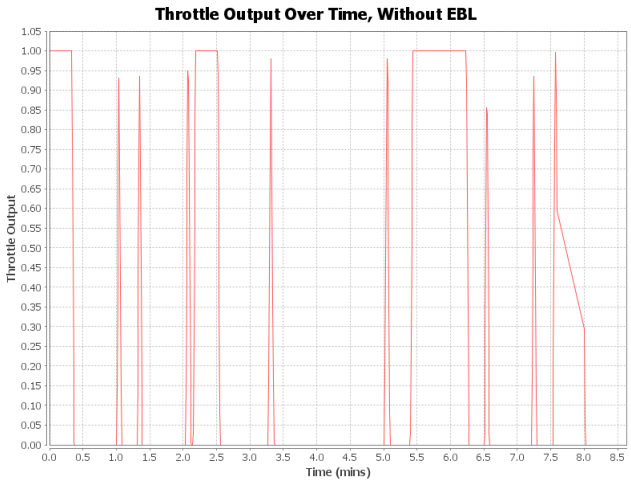


**Figure 4.6: The throttle control value of the EMG-5 aircraft over time from a single flight without EBL**

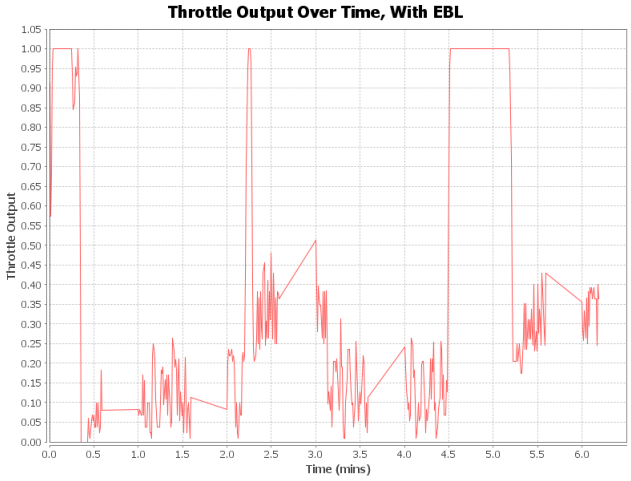


**Figure 4.7: The throttle control value of the EMG-5 aircraft over time from a single flight with EBL.**

In order to compare the energy usage the charge of the main battery was averaged over the 50 runs. Figure 4.8 shows this for flights without EBL, Figure 4.9 for flights with EBL.

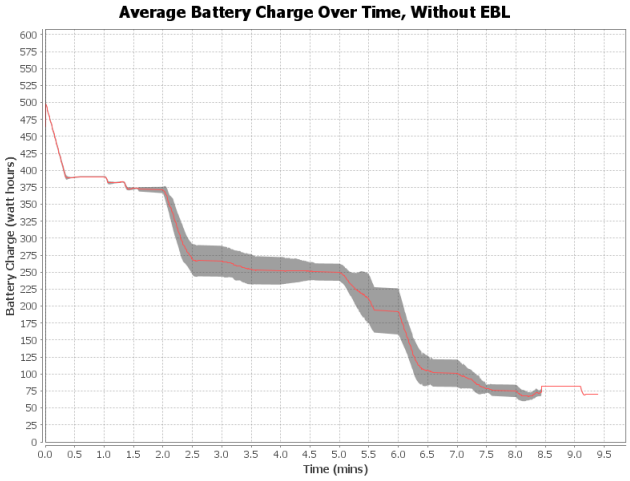


**Figure 4.8: Average battery charge of the EMG-5 aircraft without EBL.**

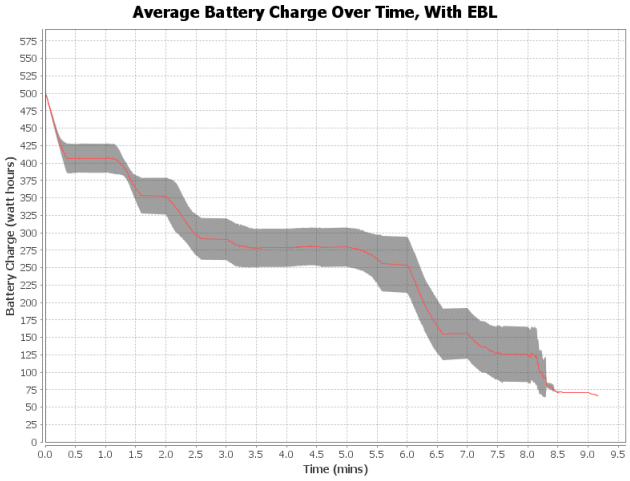


**Figure 4.9: Average battery charge of the EMG-5 aircraft with EBL.**

However, it took 44 runs before the system stopped adding new rules during a flight. The number of rules generated in each run are shown in Figure 4.10. It is worth noting that rules generated in one run are not guaranteed to be executed on the same run. The rules generated are given in Table 4.2. The battery charge for an additional 20 runs after this point was used; meaning a total of 64 runs in the dataset. In this way, only runs where the rulebase is saturated are used. This is more illustrative of the effect of the fuzzy controller. Using the full dataset could obfuscate the effect of the controller. This is because data from runs with an untrained controller would be included.

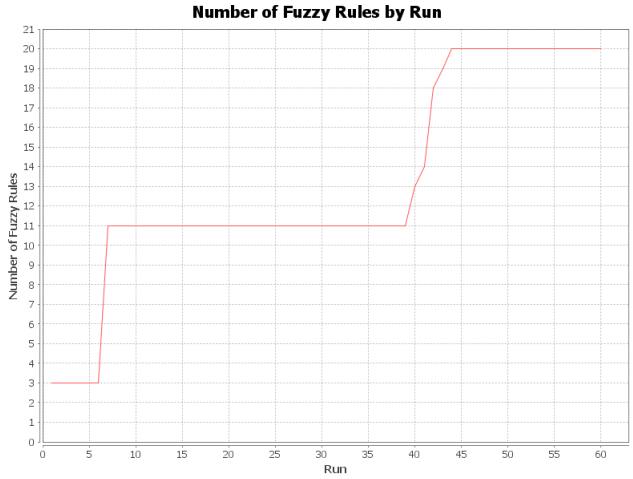


**Figure 4.10: The cumulative number of fuzzy rules generated by the end of each flight.**

| Run | Input | | | Output |
|---|---|---|---|---|
| | Throttle in | Energy (f min$^{-1}$) | Indicated Climb | Throttle out |
| 1 | full | full | full | huge |
| 1 | small | full | full | tiny |
| 1 | medium | full | full | small |
| 1 | large | full | full | medium |
| 1 | full | huge | huge | huge |
| 1 | tiny | none | none | none |
| 1 | small | none | none | tiny |
| 1 | medium | none | none | small |
| 1 | huge | full | full | large |
| 2 | full | large | large | huge |
| 2 | large | tiny | tiny | medium |
| 2 | huge | large | large | large |
| 2 | large | large | large | medium |
| 2 | medium | medium | medium | small |
| 2 | large | medium | medium | medium |
| 2 | medium | small | small | small |
| 2 | large | small | small | medium |
| 2 | small | small | small | tiny |
| 2 | medium | tiny | tiny | small |
| 2 | small | tiny | tiny | tiny |
| 2 | tiny | tiny | tiny | none |
| 3 | huge | medium | medium | large |
| 5 | huge | huge | huge | large |
| 5 | large | huge | huge | medium |
| 5 | tiny | small | small | none |
| 7 | medium | huge | huge | small |
| 7 | medium | large | large | small |
| 7 | small | medium | medium | tiny |
| 8 | tiny | full | full | none |
| 28 | tiny | medium | medium | none |

| | | | | |
|---|---|---|---|---|
| 35 | small | huge | huge | tiny |
| 40 | small | large | large | tiny |

**Table 4.2: Table of the fuzzy rules generated, by run.**

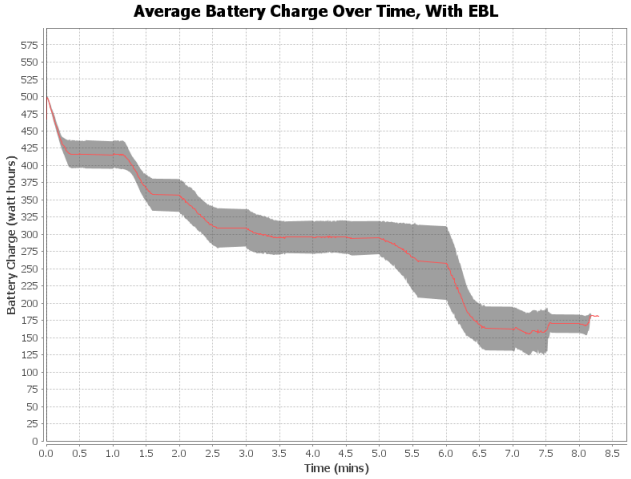


**Figure 4.11: Average of the EMG-5 aircraft's battery charge with EBL and a saturated fuzzy rule base.**

At this point it is worth further considering the effect of flying to lower than specified elevations. Figure 4.12 shows the altitude over time, averaged over 50 flights, where the elevations input into the FMS were lower. The elevations used were:

- 500 ft
- 1200 ft
- 750 ft
- 1600 ft
- 1600 ft

An average of the battery charge, over 50 flights, is shown in Figure 4.13. These flights were conducted without EBL and using the lower altitude checkpoints. By comparing Figure 4.13 to Figure 4.11, it appears that the degradation in altitude alone was not responsible for all of the energy savings.

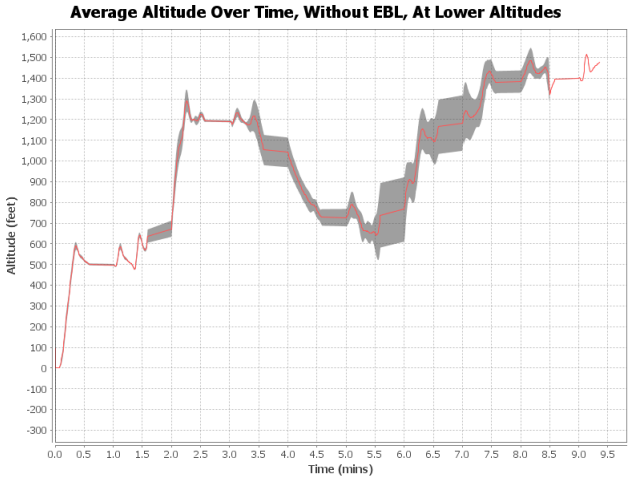


**Figure 4.12: Average altitude of the EMG-5 aircraft without EBL and with lower altitude checkpoints.**

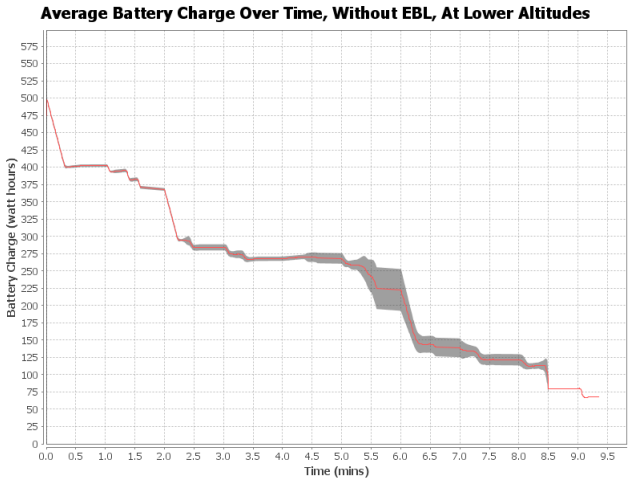


**Figure 4.13: Average battery charge of the EMG-5 aircraft without EBL and with lower altitude checkpoints.**

Figure 4.14 shows the explanation structure for the first rule generated, given in Table 4.2. The nodes circled in red, in the explanation structure, highlight the leaf nodes used to generate the fuzzy rule. Each of these is taken in conjunction to imply the goal. Note that when there are duplicates only one example is taken. The parameters from each atom are extracted and converted into the fuzzy rule format as they contain the information required. The required information is the input/output name and value.

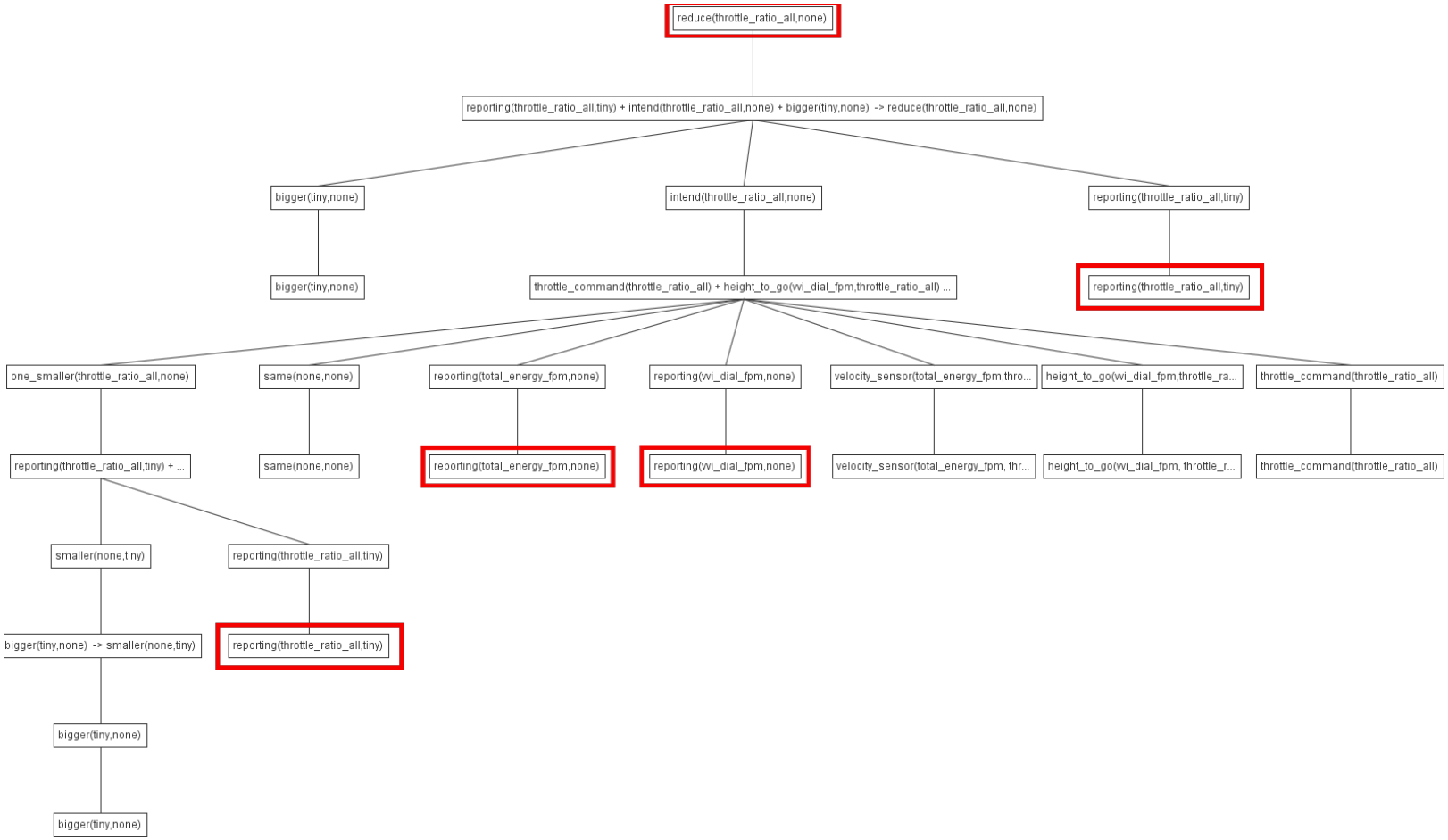


**Figure 4.14: An explanation structure which is used to derive a fuzzy rule.**

It is noteworthy that only certain parts of the diagram differ for rule derivation. Due to the simplicity of the domain theory, not all of the areas of the explanation structure have alternatives. There is an alternative version of the *intend* rule which would replace *same(none,none)* with nodes using the predicate *smaller*. All areas of the structure which deal with relative orders of magnitude could be larger or smaller depending on how far apart the linguistic values are. For example, *smaller(none,full)* will require more explanation than *smaller(small,medium)*. This is because there is a statement, which is considered operational, for *bigger(medium,small)*. This statement along with the rule relating smaller to bigger are all that are required. More rules are needed to explain *smaller(none,full)*. The only part of Figure 4.14 where this does not hold is *bigger(tiny,none)*. This is because the *one_smaller* nodes ensure that the variables considered near the top of the tree will be satisfied with one subnode; the linguistic values will be next to one another in order. It is the generalisation of linguistic values to variables which allows multiple rules to be generated even though much of the explanation structure remains the same for each derivation.

There is evidence to show that the domain theory does not result in an optimal controller. The variance between throttle output in different flights is pronounced; this would imply that there was no convergence to an ideal state. This can be seen by comparing Figure 4.7 with Figures 4.15 and 4.16. It can be seen in Figure 4.16, such as between 5 and 6.2 minutes, that there are times in the flight where no fuzzy rules are being executed. These areas take the form of a plateau of full throttle output. A likely cause of this difference is the domain theory. The rules generated are based on the domain theory, which restricts the possible rules. This means that there will be states for which no rule will be generated. The width of fuzzy sets limits the granularity of recognising a given state. Each of these states are delineated by the fuzzification process. This means that there will be cases where the state is close to, but just outside of, the coverage of the domain theory. The unaffected areas become rarer as the rulebase saturates. A greater number of rules mean that a larger number of situations will have action bindings. Figure 4.15 is taken from after the rulebase reached saturation, but Figure 4.16 is from a flight where the rulebase was still developing and was comprised of 11 rules.

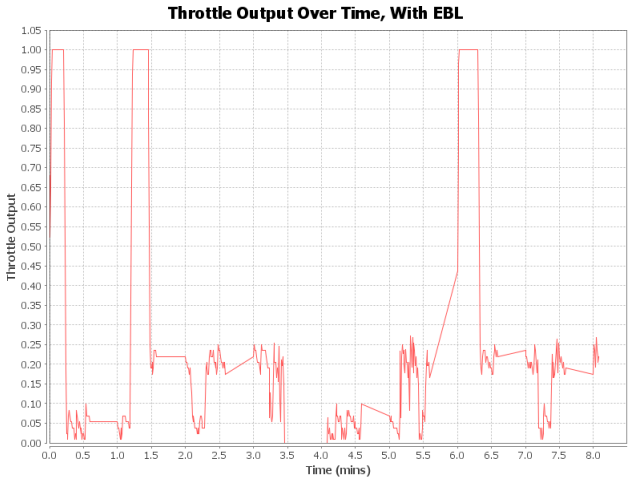


**Figure 4.15: The throttle control value of the EMG-5 aircraft over time from a single flight with EBL and a saturated fuzzy rule base.**

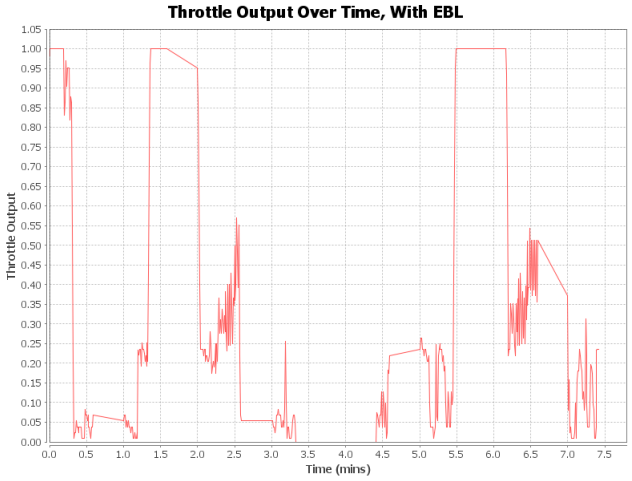


**Figure 4.16: The throttle control value of the EMG-5 aircraft over time from a single flight with EBL.**

Figure 4.17 sums up the work so far. The figure shows the remaining charge in the battery at the end of each run, both with and without EBL. The blue line shows the runs without EBL enabled. The red line shows runs with EBL enabled. The difference between these two graphs comes from the alteration in throttle behaviour. A reduction in throttle over the flight reduces the power consumption and saves energy; this is because the aircraft under test is all electric. Therefore, it can be seen that by employing EBL energy is saved. An interesting point of future work can also be elicited from this graph. It is not clear that the technique is monotonic: further study into this aspect of the system could prove beneficial.

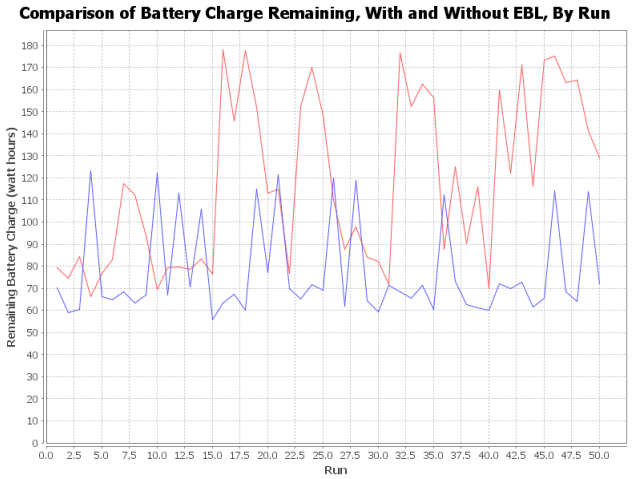


**Figure 4.17: The battery charge remaining at the end of each run with and without EBL.**

The results presented in this chapter show that the approach presented can be used for energy management, with even a simple domain theory. The results fulfil the objective of showing that this technique can generate rules which reduce energy consumption of an aircraft, satisfying the intent of the domain theory. This supports the position that the generation of fuzzy rules via EBL can produce useful rules. It remains to consider whether the technique can produce useful rules for a different platform; this would support the applicability of the technique to the control of a modular system.

## 4.4 Passenger Jet Application

The controller was applied to a different platform: an Airbus A320. The same flight was conducted both with and without EBL. Rules for controlling the throttle were elicited over the same route as the previous experiment. The aim of conducting the same experiment on an A320 is to show that the approach can adapt the same general strategy to a different platform. Application to a different platform is an extreme example of a change in hardware. In this case the hardware being controlled (throttle) remains, but changes to the platform affect the specifics of how it should be controlled. When applied to a new platform the rules generated should still cause a reduction in energy consumption.

The route followed can again be suggested by graphing the altitude. There are two main factors which alter the altitude from the previous experiment: Firstly, the A320 is capable of far greater speeds. Secondly, the A320 appears to operate using a different autopilot. This was evinced by the internal state of the autopilot differing from the EMG-5 when given the same commands. Additionally, the difference in autopilot was shown by the distance required to pass through a checkpoint differing. The A320, essentially, took a slightly truncated route as it counted as passing a checkpoint when further away than the EMG-5. The altitudes for the runs with and without EBL are given in Figures 4.18 and 4.19 respectively.

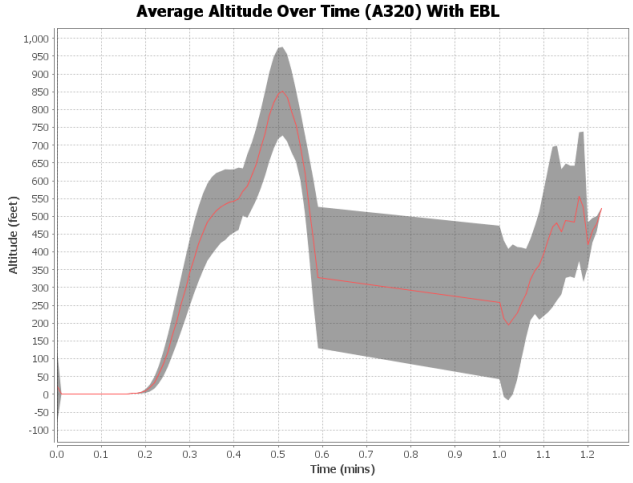


**Figure 4.18: Average altitude of the A320 aircraft with EBL.**

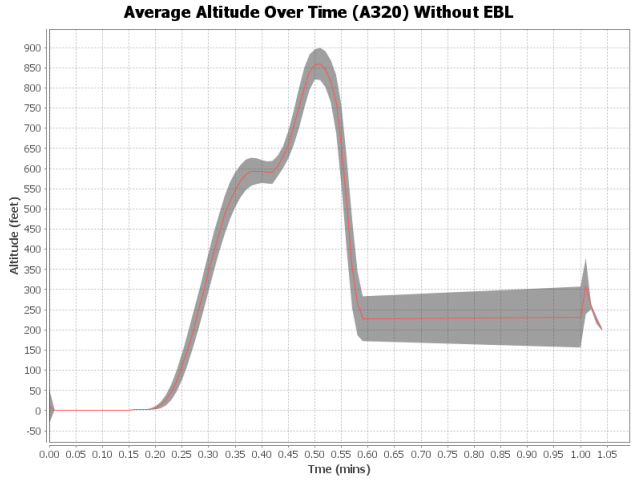


**Figure 4.19: Average altitude of the A320 aircraft without EBL.**

These graphs differ as the alterations in throttle caused the autopilot to fly a slightly longer route. The actual route also deviated between runs resulting in a large standard deviation. The route flown passed over a more significant rise in the ground. This resulted in the drop in perceived altitude after half a minute. The sensor for altitude measures the distance from the ground below. As the ground rises, the perceived altitude diminishes.

When applied to a different platform, the controller generated a different set of rules. The number of rules per flight is given in Figure 4.20. In order to demonstrate that these rules impacted the throttle control Figures 4.21 and 4.22 show the throttle output for example runs both with and without EBL respectively. The example which uses EBL is from a run after the rule base had reached saturation.

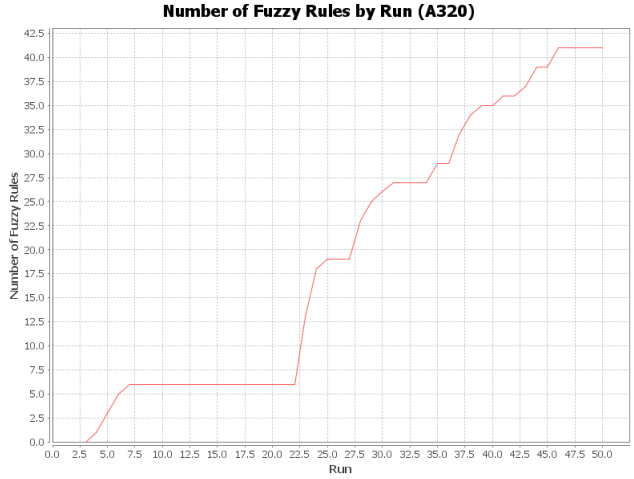


**Figure 4.20: The number of rules generated by the end of each A320 flight.**


**Figure 4.21: Throttle control value of the A320 over time from a single flight with EBL.**

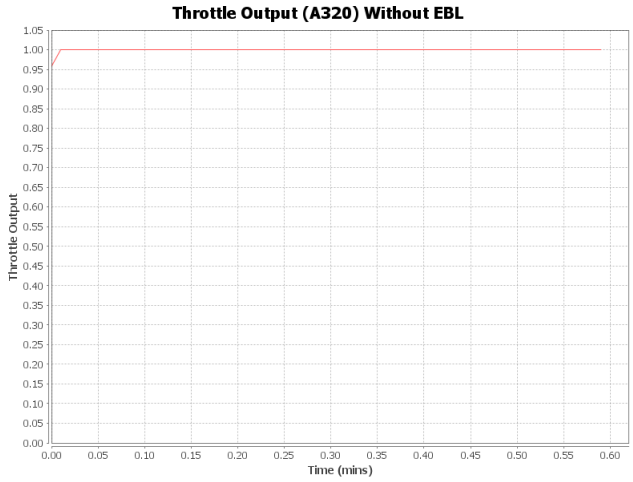


**Figure 4.22: Throttle control value of the A320 over time from a single flight without EBL.**

Table 4.3 shows the rules generated during the simulated flights. The rules relate the input, throttle requested, to the output value. In Table 4.3 the fuzzy rules that were generated represent the specialisation of a general strategy to an encountered situation, facilitated by EBL. As there are some rules which share throttle input and output values, but not each combination allowed by the domain theory, it can be inferred that this rule base is conceptually incomplete. Rules are only generated for the situations presented to the system because rules are generated using these situations as an inductive bias. This allows for simulation to be used to train the system to apply to different platforms.

| **Run** | **Input** | | | **Output** |
|---|---|---|---|---|
| | **Throttle in** | **Energy (f min$^{-1}$)** | **Climb (f)** | **Throttle out** |
| 4 | full | huge | huge | huge |
| 5 | full | none | none | huge |
| 5 | full | medium | medium | huge |
| 6 | full | tiny | tiny | huge |
| 6 | full | small | small | huge |
| 7 | full | large | large | huge |

| | | | | |
|---|---|---|---|---|
| 23 | full | full | huge | huge |
| 23 | large | full | large | medium |
| 23 | huge | full | large | large |
| 23 | full | full | medium | huge |
| 23 | huge | full | medium | large |
| 23 | full | full | large | huge |
| 23 | full | full | none | huge |
| 24 | full | full | small | huge |
| 24 | large | full | medium | medium |
| 24 | huge | full | tiny | large |
| 24 | huge | full | small | large |
| 24 | full | full | tiny | huge |
| 25 | medium | full | medium | small |
| 28 | tiny | full | medium | none |
| 28 | small | full | medium | tiny |
| 28 | small | full | large | tiny |
| 28 | medium | full | large | small |
| 29 | huge | full | none | large |
| 29 | large | full | none | medium |
| 30 | medium | full | none | small |
| 31 | large | full | small | medium |
| 35 | full | full | none | large |
| 35 | small | full | medium | none |
| 37 | medium | full | tiny | small |
| 37 | small | full | tiny | tiny |
| 37 | large | full | tiny | medium |
| 38 | small | full | none | tiny |
| 38 | tiny | full | large | none |
| 39 | tiny | full | none | none |
| 41 | full | full | none | medium |
| 43 | large | full | none | small |
| 44 | medium | full | small | small |
| 44 | tiny | full | small | none |
| 46 | small | full | small | tiny |
| 46 | tiny | full | tiny | none |

**Table 4.3: Table of the fuzzy rules generated for the A320, by run.**

In Table 4.3 the rules generated for the A320 are given. This is a much heavier aircraft with smaller wings, relative to its size. This change of aircraft is akin to an exaggerated example of altering a UAV between missions. The same domain theory produced different rules. The heavier aircraft has a larger over-provisioning of thrust, it is capable of much greater speeds and carries more fuel compared the the battery of the EMG-5. These differences result in a larger number of rules being generated. More rules are generated as there are more states possible where the aircraft climbs using more thrust than necessary.

The fuel remaining at the end of each flight, with and without EBL, is presented in Figure 4.23. The red line shows the runs with EBL enabled. The blue line shows runs without EBL enabled. The introduction of the fuzzy controller led to a reduction in fuel consumption. All of the runs with a complete fuzzy rule base consumed a reduced amount of fuel. Run 23 produced an anomalous result. It is worth noting that the differences in fuel level between any two runs is small. This is because the A320 has a much larger supply of energy, in the form of fuel, compared with the EMG-5. The A320 also operated over a shorter distance and smaller time-frame than the EMG-5 due to the differences in flight characteristics. Namely, that the A320 has a much higher top speed which reduces the time taken to fly the route. Also, the aforementioned autopilot disparity between the two platforms results in shorter flight times. The anomalous result was the product of the autopilot flying a significantly different route than in the other flights.

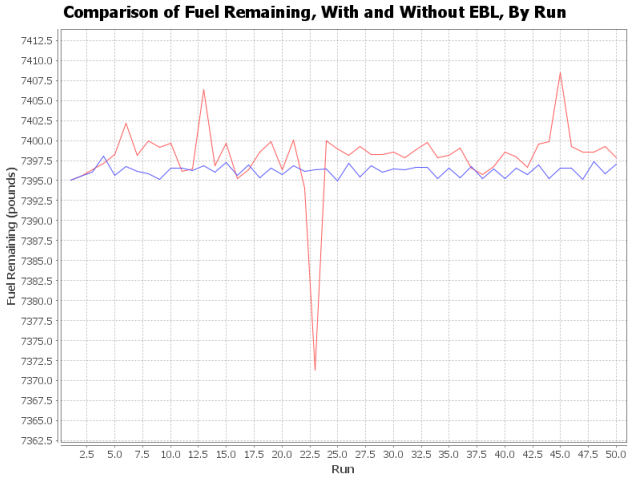


**Figure 4.23: The fuel remaining at the end of each run with and without EBL.**

The results presented in Figure 4.23 support the supposition that the rules generated by this technique were appropriate when applied to a new platform. Use of an A320 is an extreme case and appears to support the applicability of the approach to modular platforms via the generation of appropriately different rules. Furthermore, as energy was conserved in each experiment, the rules generated reflect the intent of the domain theory and can therefore be considered useful. The application to two platforms, each generating useful rules, fulfils the objective of this chapter.

## 4.5 Conclusion

An approach was posited in Chapter 3 to address an issue in the control of a modular system. Namely, control of a modular platform can engender a workload when changes in hardware require changes in control. This chapter has presented an experiment, conducted on two platforms. The aim of this experimental work is twofold. Firstly, to consider whether EBL generates rules which have the intended effect on the system being controlled. Secondly, to compare the results stemming from the experiment when conducted on two different platforms.

The technique was considered in terms of its applicability in general, then specifically with regards to another platform. The generation of fuzzy rules which reflect the intention of the domain theory supports the generation of rules via EBL. The system was applied to a pre-set flight on an all-

electric glider. A marked increase in energy conservation when the controller was operating was shown. Application of the system to another platform is an exaggerated case of modularity. The generation of energy-saving rules on the new platform support the applicability of the technique to modular control. Application of the controller to a new hardware platform caused different rules to be generated. The rules generated were appropriate to the new platform, exhibiting the expected differences. Therefore, the aim of the chapter was met. This is because useful fuzzy rules were generated and the same strategy was successful in controlling platforms with different hardware. Fulfilment of this objective validates the possibility of using EBL to generate fuzzy rules, particularly in support of modular control. The results in this chapter therefore sit in support of the possibility of using EBL to generate fuzzy rules, and the application to modular control

The results of this chapter serve to suggest some areas for further research. A more detailed analysis of this approach as well as further applications would be advantageous. Application of the technique to hardware, rather than in simulation, could also be beneficial. Finally, this method has only been applied monotonically to date. A more robust approach would consider non-monotonic aspects.

Useful rules were generated for two different platforms, an extreme case of modularity, given a single domain theory. This acts as a proof of concept for the approach proposed in 3. This supporting evidence completes this investigation of EBL and fuzzy control in terms of modular management, which is the first objective of this project. The aim of the project is supported by the fulfilment of the first objective because the integration of fuzzy control and EBL can be used to reduce the workload engendered by controlling a modular platform.

It has been shown that the techniques can be applied to the energy management of a modular platform. It remains to extend EBL to include analogical reasoning. This would give the system the benefit of transfer learning, which could be resorted to upon reaching logical impasses. This would allow a domain theory to be extended to include previously unseen hardware. Another aspect of the system that remains to be considered is a way of reducing the recertification effort that comes with a modular system. These issues will be explored in the remaining chapters.

# Chapter 5

# Explanation-Based Learning and Analogy

## 5.1 Introduction

Part of the aim of this project is to reduce the workload required when managing a changing platform. The integration of EBl and fuzzy control can achieve this by utilising generalisation. However, this only covers cases which are within the initial deductive closure of the domain theory. In order to satisfy the aim a second objective is required. This objective is to investigate the extension of EBL to include analogy. Analogy is useful because it can further the generalisation capabilities of EBL to allow strategies to be applied to hardware that was not initially catered for by the domain theory. This second objective is the focus of this chapter.

This chapter proposes a way to incorporate analogy into EBL. The argument is that by linking previously unrelated concepts EBL can use analogical relationships to make inductive leaps. These inductive leaps are limited to situations where no deductive option is available. The analogical links can be used to generalise rules, by abstraction, in order to apply them to situations other than those in which they were derived. Abstraction in this case refers to the replacement of a specific concept which causes an impasse with a new, more general, concept.

A solution or stratagem can be useful in solving problems in contexts other than the one in which it was discovered [38], such as in the apocryphal account of Newton's law of universal gravitation. The law was formed, anecdotally, from observation of an apple but these laws can apply to many objects. Objects have some commonalities, for example mass, which allow them to be considered analogous. Similarly, an energy management system may derive a control law for the efficient use of a particular component. It may be that the domain theory used takes no account of which concepts are analogous in a particular situation. With the addition of a conceptual link showing in which way two components are analogues one can infer: *"what works for one component will work for similar ones"*.

Chapter 3 proposed a method of using EBL to generate fuzzy rules. One of the reasons for using EBL to generate fuzzy rules is to apply a general strategy to multiple specific situations. This can support the control of a modular platform by generating specific rules applicable to it; manual

definition of the same rules would bring a heavier workload. A change in hardware can result in different rules being generated. This approach covers situations where the hardware that changes performs the same function i.e. changing a battery for one of a larger capacity. However, there is a limitation to this approach. There may be cases where the hardware has changed to be more substantially different, with respect to the EBL system, and no deductive solution is therefore possible. The limitation is apparent when a piece of hardware is added for which there is no reference within the domain theory. An example could be the replacement of a battery with a fuel cell. This prevents rule generation.

If the components and architecture of a power system are modular and could change at any point, then the energy management system may also need to change at any point. This is because a new piece of hardware, which replaces another, may require different control rules. This was addressed in Chapter 3. However, the new piece of hardware may simply be outside of the original scope of the controller. Items of hardware like this are referred to as previously unseen components in this work. A modular control system should therefore be able to derive appropriate control laws for previously unseen components.

It is possible that a previously unseen component will share both similarities to other components and an energy management strategy. Rather than constructing a new strategy there is the opportunity for transfer learning, which involves using knowledge out of the context in which it was derived to avoid learning from scratch [9]. A strategy can be used outside of the situation which it was designed for. In this case, a strategy can be applied to previously unseen hardware. This would allow for specific rules to be generated for the previously unseen hardware, as per the approach given in Chapter 3. A modular management system could therefore perform transfer learning in order to achieve more seamless control of modular systems as they change.

Transfer learning can be realised by the application of machine learning techniques, such as CBR [10] and ILP [9]. Analogy is one form of transfer learning. Analogy is a way for people to make inferences and learn new abstractions [69]. Analogy has been used to map different situations to one another, in order to apply prior knowledge to a new situation in [39]. The inferences made based on this mapping are the analogical inferences. This chapter aims to show that by linking previously unrelated concepts EBL can use analogical relationships to make inductive leaps where no deductive option is available.

EBL is constrained by the deductive closure of the terms in its domain theory [26]. This means that the rules within a domain theory limit what can be concluded. Previously disconnected statements in the domain theory can be linked by making inductive leaps. This allows growth of the deductive closure of the domain theory as more rules can be linked to form more conclusions. Therefore, this approach increases the number of situations for which EBL is applicable.

The domain theory can contain extraneous information initially, which is later incorporated into analogies. This may fit better with capturing expert knowledge, where coherent and concise capture of information is difficult. One reason for the difficulty in capturing expert knowledge may be the need to define strict relationships between concepts, hiding real-world complexity. This is akin to a viewpoint expressed on EBL, that the relationships in the domain theory may hide deeper, more complex relational information [70]. The domain theory derived for a system will likely be a subset of the complete knowledge on a subject, so as to remain tractable. In this work, rules in the domain theory which are not necessary during deduction, as part of a minimal but functional rule base, can provide the basis for analogical reasoning.

The chapter presents a method for incorporating analogy into EBL and is structured as follows: Section 5.2 proposes the method, presents some examples and discusses the technique. Section 5.3 presents an experiment where the pitch of an aircraft is controlled by using analogy to modify rules for throttle control. The chapter concludes with Section 5.4.

## 5.2 Analogical Explanation-Based Learning

This chapter proposes that terms with different function symbols be linked. The link is formed by a relation, defined by a new concept which both terms subscribe to. Less than exact analogues are used to derive the relations; these relations may not involve all the terms in each concept definition and the identity claim made by each relation is weaker than equality. If all of the terms in both concept definitions were to match, then each of the concepts would be functionally identical. This is assumed to be an unlikely case. Exact matches are not excluded by the technique proposed, but are not the only permissible identity claim. The complex relationships which might not be captured in the domain theory could be partially present in the form of two concept definitions sharing some term(s).

Similar works establish links between the same terms in different domains. The two linked terms are then used to transfer knowledge from one domain to the other. The combination of both domains forms a more complete model. Here the emphasis is on relating different terms within the same incomplete domain.

The identification of two concepts to use as the basis of an abstraction is similar to the search for similar situations during *Access* [40]. When two concepts share an abstract similarity, it is plausible to suppose that conclusions that apply to one could be mapped to the other.

One way to approach deriving a new general class is to take two concepts and use extracted similarities to form a new concept definition. The motivation for such an approach is illustrated using an example. In this example we consider the definition of a battery and a fuel cell. These two components are different but both are *sources*. Finally, we suppose that a rule has already been derived stating that a *source* can supply a demand where the demand is not greater than the maximum output of a battery. This rule could have been derived by an energy management system for a toy aircraft whose only *source* is a battery. If this system is moved to a small UAV powered primarily by a fuel cell then an analogue may be useful.

Consider some possible energy management terminology. *Sources* and *sinks* may well feature. There will be some stratagems which apply to all *sources*. An example could be that when a demand is within the maximum potential output of the *source* then that *source* could supply that demand. This is a sensible starting point for an analogue at first glance, though it may not hold for all *sources*. We could roughly define a *source* as requiring some fuel and producing electricity and waste energy. If the concept of a *source* were missing and the strategy was specific to batteries then it would not apply to other *sources*, such as fuel cells. There might be no deductive link between the components that could satisfy a given demand. This impasse would not be overcome by the generalisation that is part of EBL. This is the case since generalisation in EBL is often concerned with replacing the terms in a clause with variables sensibly [21]. An addition to this generalisation would be to derive the *source* concept where it is missing from the domain theory. This more abstract concept can be used as the basis for mutating the rules to apply to fuel cells,

an inductive leap.

Inductive leaps are realised by replacing one concept in a rule with another. This gives rise to a new rule which relates previously unrelated concepts, linked by analogy. These analogous concepts are not deductively entailed. The difference in predicates prohibits their mutual unification to explain the same goal. Additions to the domain theory which embody the similarities between the two concepts are used as a basis for replacing one predicate with another. Rather than generalising a rule by abstracting specific variable bindings from terms, as in EBL, two concept definitions are linked by a more abstract class so that the terms themselves can be replaced. This is similar to assuming that the predicates themselves could be bound differently. The abstract class can be formed from commonalities between otherwise disparate concept definitions. This class embodies an analogy and is expressed as a new rule.

An alternative to implicitly linking two terms and replacing one with the other is possible. Both terms could be replaced with the abstraction, which would be a more explicit analogical link. However, this could lead to over-generalisation. By storing a new rule which includes the abstraction, more than the two concepts used to derive it could now be applicable. It could be that analogies only hold in certain cases; the impact of this can be lessened by using each abstraction in only the situation which caused its derivation. Further work could be conducted in studying the over-generalisation issue with regards to this technique.

There is an issue with assuming an abstract similarity exists. Not all analogical inferences are equally likely [41] and where the similarity is based on features, not all will be relevant. This is partially ameliorated by EBL disregarding features not relevant to the example, using an inductive bias. When trying to derive an abstract similarity between two concepts, even using an example as an inductive bias, it is likely that not all features in the example are relevant to the analogue being derived. Using an example as an inductive bias, along with empirical validation, could help to disregard spurious analogies.

### 5.2.1 Abstraction Derivation

The approach proposes the derivation of an analogical link between two deductively separate concepts. This link is formed by the generation of a third concept, which requires commonalities between the two target concepts to be satisfied. In this way an abstraction of the two concepts is formed. Membership of this abstraction implies that concepts are analogous in the manner given by the abstract concept definition. This could be thought of as weakening the preconditions on the target concepts to form an abstraction. The preconditions are weakened as the abstraction definition is formed from a subset of the preconditions of both identified concepts. Alternatively, the same number of preconditions could appear but would be generalised by the introduction of variables. One of these forms of weakening is the case for all instances where both definitions are not identical. Identical concepts are assumed to be unlikely.

The two concepts which are used to form an abstraction are the source and the target. The source is the definition of the predicate which causes an impasse. The target is a concept searched for, using the parameters of the source as a search bias.

*Mapping* establishes the similarities between two identified concepts. An approach to achieve this within EBL is proposed. The approach for forming a new, more abstract, definition from two

target concept definitions is:

1. Collect like terms in the leaf nodes of both concepts and reduce the number of occurrences to one by introducing variables and forming a substitution. Do this for both, or all, concepts being considered.

2. Discard any terms that do not appear in all concept definitions. When considering many concept definitions, one may discard from consideration concepts that lack common terms rather than the terms themselves. As long as a relationship between at least two concepts is established, this is permissible.

3. Use these common terms to form the precedents for a new concept definition. This definition captures some similarity between the concepts considered which is assumed to be absent from the initial domain theory.

This approach has been formalised in Algorithms 2 and 3. Algorithm 2 selects the analogy with the most terms in it, corresponding to matches between any combination of definitions for both source and target terms. This representation calculates all the possible abstractions between each definition of both concepts, sorted in descending order of complexity. In practice, the most complex abstraction is taken and assumed to show a closer connection between two concepts. Algorithm 3 forms the abstraction between two concepts.

**Algorithm 2** Impasse(term source)

**Require:** source is a ground atom
  targets ← rules containing predicate(s) with arguments matching the source.
  SourceDefs ← rules previously derived that imply the source, stored within the rule-base.
  *// Collect analogies between each definition of the source and target. These analogies are stored in "analogies" which is sorted by number of terms (descending) in the abstraction.*
  **for all** target in targets **do**
    targetDefs ← rules previously derived that imply the source, stored within the rule-base.
    **for all** targetDef in targetDefs **do**
      **for all** sourceDef in sourceDefs **do**
        Abstract(source, target)
        store target in analogies along with its abstraction (targetDef).
      **end for**
    **end for**
  **end for**
  add the target, topmost from analogies, under source in the explanation structure.

**Algorithm 3** Abstract(term source, term target)

**Require:** source and target predicates are ordered.
  get the first terms from source and target concepts.
  **for all** predicates in source **do**
    **if** current predicates from source and target concepts differ **then**
      get next source predicate.
    **else**
      **while** the next predicate in the source matches the current one **do**
        note the arguments, by arity, from source term.
        get the next term from source.
      **end while**
      **while** the next predicate in the target matches the current one **do**
        note the arguments, by arity, from target.
        get the next term from target.
      **end while**
      **for all** arguments from source for this predicate. **do**
        **if** there is a corresponding argument in target **then**
          add the current predicate, with the matching argument, to the abstraction.
        **else**
          *// this should be limited to once per predicate*
          add the current predicate, with an introduced variable as the argument, to the abstraction.
        **end if**
      **end for**
    **end if**
  **end for**
  **return** the abstraction.

Terms are discarded which are not common to both source and target definitions, as these are assumed to be irrelevant to whatever similarity is being captured. This is to facilitate the capture of analogues that have the most in common. More general analogues that make a weaker claim of similarity between concepts could also be useful but could more readily lead to over-generalisation. Also, it may be computationally wasteful to derive very general analogues since these could potentially be applied often to little effect. It seems like a better starting point to only consider analogues that make a stronger claim. A simple heuristic and threshold could be used to prefer more complex analogues. However, an increase in complexity would make the analogues more expensive to match.

Replacing the goal which generated an impasse, with the analogue in place of its counterpart, could be useful; but some account must be made for the fact that it may not apply to all situations. This trial and error replaces some deeper reasoning that is not present in the domain theory. For example, consider the definitions of a sensor (airspeed indicator (ASI) depicted in Figure 5.2) and an engine (internal combustion engine (ICE)) depicted in Figure 5.1). These definitions are similar and one could conclude that a rule, stating that a sensor can be turned off when not in use, may apply to an engine. This is not true. Information about flight mechanics and engine design would need to be included in the domain theory to reason about why an engine often cannot benefit from

being turned off. This mechanism can seek to fill gaps in knowledge without the explicit reasoning, but inductive leaps may not always be appropriate without further information.

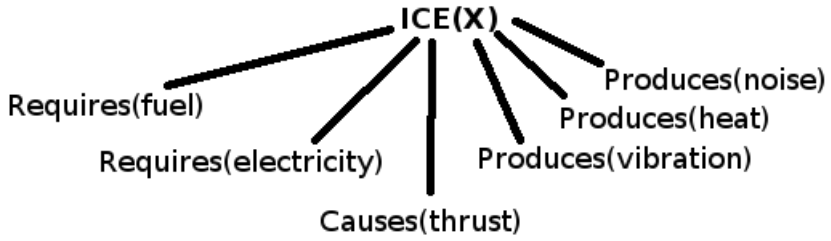


**Figure 5.1: Potential definition of Internal Combustion Engines (ICEs) as a tree.**

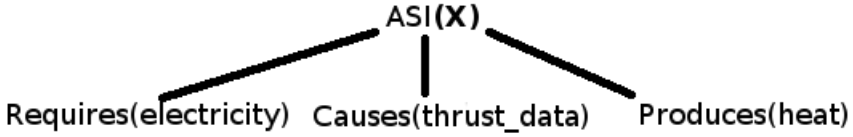


**Figure 5.2: Potential definition of Air Speed Indicators (ASIs) as a tree.**

An example will be presented to illustrate the technique, followed by a discussion.

### 5.2.2 An Example

The analogical EBL algorithm can be applied to an energy management example. Consider the MPMS for a UAS, once powered by only a battery, being altered to be powered by a fuel cell. The domain theory and observations are given in Table 5.1. In this example, the domain theory lacks the definition of a *source*. A definition for this will be derived using information available about batteries and fuel cells.

| **Goal** |
|---|
| can_supply(fuel_cell_1, ASI_demand) |
| **Training Data** |
| max_output(fuel_cell_1, 50) |
| demand(ASI, ASI_demand) |
| **Domain Theory** |
| limited(X, charge) + waste(X, heat) + requires(X, charge) + produces(X, energy) → battery(X) |
| fuel(X, hydrogen) + waste(X, heat) + waste(X, water) + requires(X, hydrogen) + produces(X, energy) → fuel_cell(X) |
| battery(X) + max_output(X,O) > demand(C,Y) → can_supply(X,C) |
| fuel_cell(fuel_cell_1) |
| battery(battery_1) |

**Table 5.1: A sample input for EBL**

A general *source* definition that we will be derived is given as:
*requires(F)* $\wedge$ *waste(W)* $\wedge$ *produces(energy)* $\rightarrow$ *source(X)*

A rule applying to a battery, which should be altered to work with a fuel cell, is: *battery(X)* $\wedge$ *max_output(X,O)* $>$ *demand(C,Y)* $\rightarrow$ *can_supply(X,C)*

During execution of the EBL algorithm an impasse is reached when encountering the goal *battery(fuel_cell_one)*. No rules exist which allow the fuel cell to be selected deductively; there is no statement *battery(fuel_cell_one)*.

The only single argument statement for *fuel_cell_one* is *fuel_cell(fuel_cell_one)*. This correspondence in parameters is used to identify the source and target concepts. In this case, *battery* is the source concept (known solution) as it is this outer predicate which fails to match a known statement. A concept is inferred to take the form *X(fuel_cell_1)*. The definitions compatible with this form are the target concepts. In this case there is one: *fuel_cell(fuel_cell_1)*.

The definitions of the source and target concepts are given in Figures 5.3 and 5.4. These definitions are used to form the *source* definition by conjunction of their similarities.

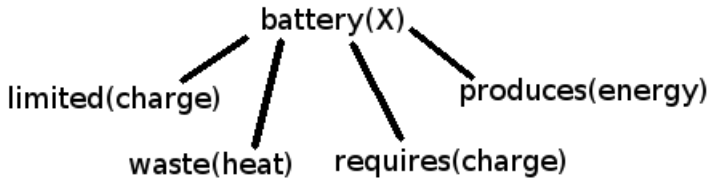


**Figure 5.3: Potential definition of a batteries as a tree.**

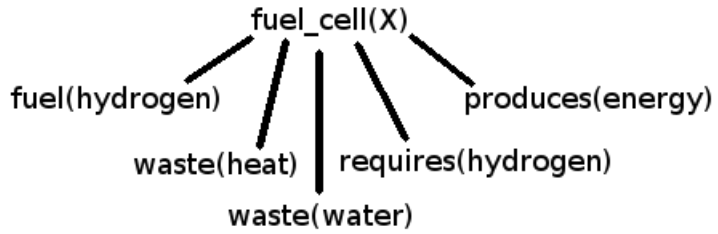


**Figure 5.4: Potential definition of fuel cells as a tree.**

There are three terms in this example (*Requires*, *Waste*, *Produces*) that are common to both concepts. Both the source and target concepts should subscribe to the derived analogy. Differences in parameters need to be resolved. For this reason variables are introduced for most clauses in the new definition. For example, there are different arguments for *waste* so a variable is introduced. This allows the common predicate to be included in the abstraction in a manner which ensures that both source and target can be considered a *source*. Introducing a variable captures a weaker similarity between the two concepts and is akin to weakening the pre-conditions, compared to either target concept. A new class can be formed, resulting in the *source* definition.

However, *produces(energy)* may form part of the analogy. This is because for each occurrence of

the *produces* predicate, in both source and target, *energy* is always the argument. Therefore, no substitution is necessary to resolve the instances of the *produces* predicate. The conjunction of these common terms results in the source definition that was proposed earlier.

After deriving the abstraction of a *source*, rules for the battery can be altered at the time of application. These rules can be modified to apply to hardware that was not present when the rule was first derived, such as a fuel cell. This is shown in Figure 5.5. Single lines depict implications and can be read such that the lower line implies the statement above. A box shows where the proposed technique is being applied with the double line representing an abstract similarity.

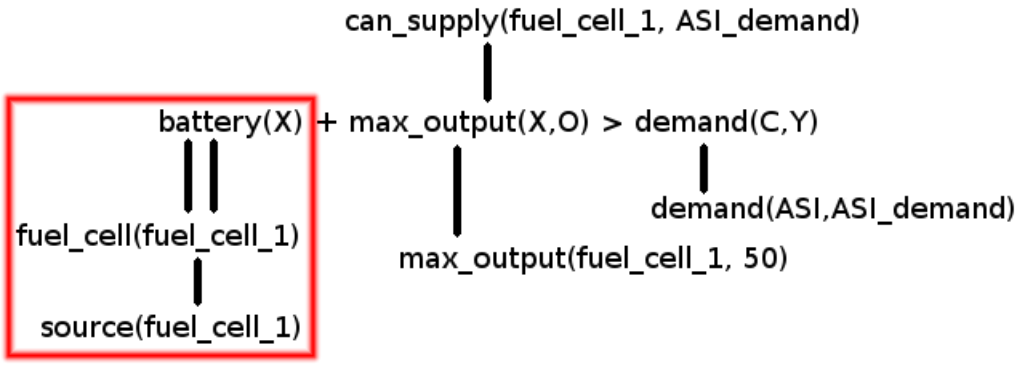


**Figure 5.5: Example explanation structure starting with battery oriented rules and using abstraction to substitute battery for *source*.**

The highlighted area in Figure 5.5 is the point where the explanation for *can_supply* would fail for lack of a sentence *battery(fuel_cell_1)*. The double line is where two analogue cases are selected, one from *battery(X)* in the domain theory. The other is inferred to exist from the binding of *fuel_cell_1* within *battery(X)*. Since there is no exact match for *battery(fuel_cell_1)*, it is inferred that there may be an analogous term for *fuel_cell_1*. Identification of this term is akin to *Access*. The analogy is generated to link the two terms and then the *battery* predicate can be substituted for *fuel_cell*.

The derived concept could then replace the instance of battery in the explanation and a more general rule could be derived: $source(X) + max\_output(X,O) > demand(C,Y) \rightarrow can\_supply(X,C)$. Alternatively, the *source* predicate could be replaced with *fuel_cell* and the analogy sits in support of this decision. The second approach is adopted in this chapter. This produces a less generally applicable rule, as other pre-existing concepts might also subscribe to the new *source* definition. The rule may not be applicable to these other concepts. This is an example of over-generalisation.

The rule for the battery can be expanded to consider a fuel cell. However, whilst this looks like a sensible application of a rule in an abstract way it is likely that over-generalisation can occur. Alternatively, rules could be generated which would ideally be prohibited with a more complete domain theory. It is therefore important to only propose an analogy when the result is a useful rule. The utility of an analogy is not apparent at the time of derivation therefore the individual rules may need to be evaluated post generation.

It is worth noting at this stage that the certification of rules generated by analogy is an open issue. Rules generated by analogy lack the implicit trust engendered by utilising deduction. This controller is not currently certifiable. In order to progress in this direction the issue of how to certify the domain theory itself must be addressed. Analogical rules are then an extent to an

existing domain theory. These rules can be treated in isolation, so far as rule interaction allows. A possible method for certifying an analogical rule is to simulate its application and bound its effects. This representation of the rule can then be considered using a more typical argument. This is akin to the gains scheduling approach.

Rules generated by analogy are best evaluated post generation. This includes evaluation in terms of their certification. Analogically generated rules could be generated which lead to behaviour outside of the flight envelope, which undermines the certification of such a system. The design of the domain theory can limit the rules which can be generated by analogy to avoid this problem. However, this further highlights the issue that certification of the domain theory remains an open problem.

Analogy can be used to incorporate new hardware into an existing domain theory. This comes with the costs of deriving and applying the abstraction. However, the technique gives an alternative to failure which can lead to a more general domain theory. This is particularly advantageous in terms of managing modular systems, where the domain theory may be strategically relaxed to incorporate new hardware without needing to be manually tailored.

### 5.2.3 Discussion

One cost of applying an analogy is the utility problem, which is an issue that comes from any application of EBL. The *Utility problem* is defined as *"The difficulty in ensuring that an acquired concept enhances the performance of the application system"* [11] and is discussed more in [27]. Approaches to mitigating this should be employed, such as including *a priori* knowledge relating to utility in the domain theory. Another approach is the empirical testing and removal of analogies that increase the branching factor significantly without deriving useful rules.

Applying an analogy will have a cost, not just in matching but by increasing the branching factor of the problem. This is a result of augmenting the domain theory; more possibilities result in longer searches. It may be that no useful information is gained by permuting a rule to apply to a new situation via analogy. There is no deductive evidence that the similarities between two concepts mean that a given rule applies to both. As there is a cost and potentially no gain, since the leaps are inductive, this technique should not be overused. There may be reason to think that similarity with nodes higher in an explanation structure are better for judging applicability. This is described in [38]. There may also be reasons to prefer more specific rules [39]. However, when no deductive option is available this technique may give an alternative to backtracking or failure. Choosing to derive an analogy could be likened to strategically weakening preconditions, given that an analogy is a subset of atoms of a concept.

It may be possible, at any point, to derive many potential analogies between concepts. However, since not all of these will be useful a conservative approach is to only adopt an analogy as a last resort to failure. If an analogy is adopted during an explanation to prevent failure, then the validity of the analogy could be assessed by whether the new rule can be successfully applied. The criteria for judging the utility of the rules may well be domain independent. An energy management system might keep only those rules which result in an increase in energy conservation.

Utility for EBL normally involves defining a point where explanation can end, via operational nodes. However, the generation of rules via analogy broadens this issue. Rules generated by analogy which

extend the domain theory are no longer a deductive application of expert knowledge. Additionally, the validity of an analogy cannot currently be judged *a priori*. In this thesis it is advocated that a rule generated by analogy should have its utility judged by empirical means. The experiment conducted in this chapter is intended to validate the claim that useful rules can be generated by analogy.

The current experiment allows one to validate a rule base with regards to the intent of the system. In this chapter the intent of the system is to reduce energy consumption. However, rules could be assessed over a series of flights both individually and in every combination. This would enable the use of a metric to judge the validity of both a rule and a rule base. It is important to test each rule combination in order to isolate the interactions rules have on the utility of other rules.

The assessment of individual rules could be approached by calculating the difference between the rate of energy consumption at a baseline and when a particular rule is being executed. Consideration of this metric over time could be used to judge the utility of a given rule performing a given task. This measurement may also consider the cost of executing a rule or a larger rulebase as there will be a lead time between identifying a rule to execute and its effect being measurable.

The metric could highlight situations where a rule which is sometimes useful does not apply. Such insights enable the augmentation of the rule to increase its utility over the flight envelope, where utility could be an integration of the gradients the metric over a flight. Negative utility rules could be rejected or considered for augmentation to restrict the situations under which they apply.

Measuring energy consumption is appropriate for this particular example but different applications have different goals and would require knowledge of a measured aspect related to the intent of a given rule.

There may be more than one possible analogy which can be derived. Biases could be used to narrow down the list of potential definitions, such as lexical similarity or meta-data labelling of concepts, ideally to one. The current bias employed is to simply take the analogy that makes the claim with the largest number of similarities. Analogues are computed for the potential matches and the strongest is chosen. In the previous example this was the *source definition*.

The source definition is given by backward-chaining from the goal which causes an impasse. The target is inferred to exist and must be searched for. This search is potentially expensive though only a subset of the domain theory could fit the pattern.

Another use for the derived analogy is when forming new rules, the analogy can be considered first and a general rule formed initially. The rule could also be annotated with meta-data showing any specific concepts considered in the rule for which it does not hold, like a caveat. This would follow from the assumption that these analogies are a simple representation of a potentially complex relationship and can be both correct and incorrect based on factors not modelled in the system.

Any analogy proposed by this method may be flawed since it is an attempt to infer some commonality and relationship that is not explicitly reasoned. The more this is applied, the greater the chance of concluding a fallacy. Therefore analogies should initially be proposed and should never be assumed to be correct. The argument is similar to that in [70]; these analogies capture information which can be both correct and incorrect depending on factors that the system is unaware of. To discard them too readily risks missing important relational information but they themselves may only capture an aspect of the real relationship. If an analogy never leads to useful rules that

hold when encountered in reality it should be discarded. Since it may be impossible to know this, a probability threshold, or similar technique, could indicate which rules to discard. Alternatively, the comparison of the impact of a given analogy on the goals of the system could be used.

Commonality in language is important to this method as it requires either shared terms or analogous terms. Note that although the terminal nodes for both the engine and sensor (Figures 5.1 and 5.2) are given in a common language, which is an assumption made for this technique, this need not initially be the case. Multiple applications of analogy could build a common set of terms from other concepts, whist discarding language that isn't commonly employed. The common language could be the result of previous analogical reasoning. The effect of this technique would be most obvious where the domain theory uses disparate terms. In this type of domain, these analogy concepts can be used to transform nodes into a more common language. Commonality in language may not be an unreasonable assumption. This is because the knowledge was originally represented using an altered subset of a shared language between experts. Therefore it may be that lexical similarity can be used heuristically to establish or evaluate an analogue.

There is a danger of applying this technique too readily since over-application would be costly and may bring about no real benefit. It is important to select when to use analogy since each application represents an inductive leap and weakens the otherwise deductive proofs EBL produces. It is important to bear these issues in mind during knowledge engineering.

## 5.3 Experiment

An experiment is proposed in order to show that analogy can allow EBL to generate fuzzy rules for a piece of hardware that is not catered for by the domain theory. This is in order to reduce the workload in adapting to the control of a changing platform, which is part of the aim of this project. The situation considered is that the hardware added is new to the control system, but strategies relating to other hardware may prove to be appropriate. Achieving the adaptation of the domain theory to apply a strategy from one hardware component to another is the objective of this chapter. The generation of fuzzy rules which reflect the intent of the system would satisfy this objective i.e. rules which reduce energy consumption by controlling previously unseen hardware.

The algorithms presented in this chapter have been implemented and integrated with X-Plane. As in the previous chapter, rules are generated for a fuzzy controller to execute. The rules are intended to control pitch, but without including information about controlling pitch in the domain theory. The exception to this is a statement and rule which link the *pitch_command* predicate to a particular X-Plane internal value. These are used to switch the throttle output for the pitch output, during explanation.

These experiments are intended to show that analogy can yield control rules for new hardware where no deductive derivation is possible. This extends the deductive closure of the domain theory. The control of pitch is analogous to the case in modular control where a new piece of hardware is added which has no information in the domain theory with which to generate control rules. The domain theory in this experiment requires analogical reasoning in order to generate control laws for pitch.

### 5.3.1 Design

The experiment consists of batches of 50 flights. It is assumed in this chapter that 50 flights will still be sufficient to elicit all of the fuzzy rules and successfully cope with the variations present in the simulator. Each flight is conducted along the same route as in the previous chapter.

X-Plane version 10 was used in the experiments in this chapter. The X-Plane update, which altered the autopilot behaviour, has lead to the collection of a new set of results, without EBL. The impact can be seen by comparing the average altitudes from the sets of flights before and after changes to the autopilot. Neither set of flights uses EBL. Figure 5.6 shows the average altitude from before the autopilot changes; this is reiterated from the last chapter for clarity. Figure 5.7 shows the same for the dataset collected after the autopilot update.

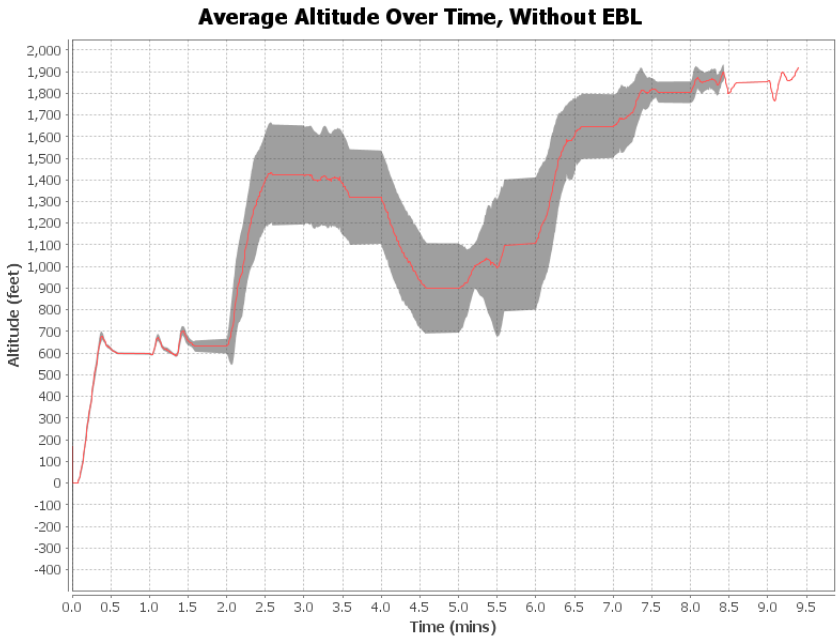


**Figure 5.6: Average altitude of the EMG-5 aircraft without EBL, before autopilot update.**

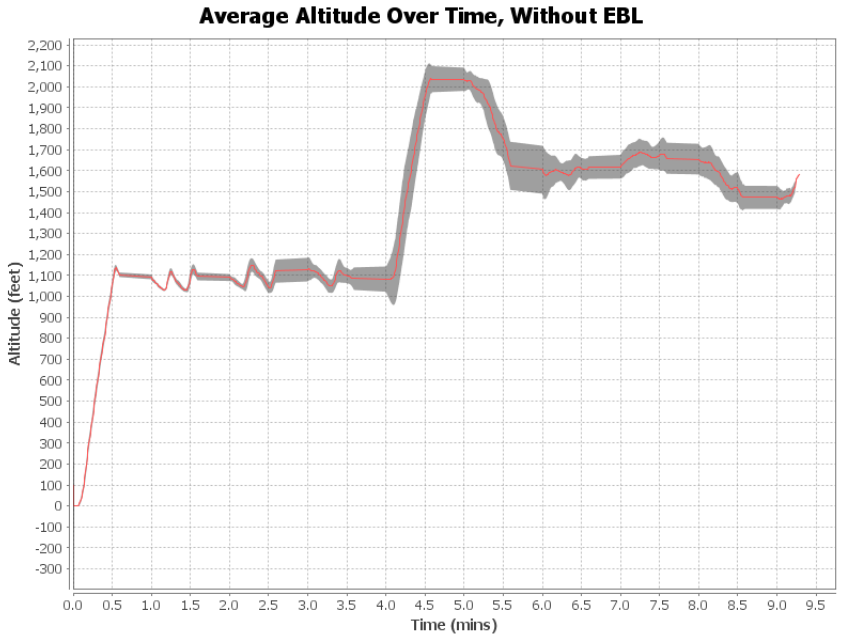


**Figure 5.7: Average altitude of the EMG-5 aircraft without EBL, after autopilot update.**

It is important to note that no changes were made to the plugin code, which controls the autopilot settings and route definition. However, the route flown by the autopilot has altered significantly. The new dataset will be used for comparison with flights which use analogy to derive pitch control rules. This presents an issue with the comparison of findings from the previous chapters with this one. By re-doing the result set without EBL, the results within this chapter remain self-consistent.

Pitch changes are executed by the FLCHG system in X-Plane. This means that the pitch controls themselves are not used. It is the throttle which controls climb and descent. The throttle setting is calculated based on the height to the next checkpoint being positive or negative. Controlling the pitch via the joystick results in a single large alteration which is then compensated for, when possible, by the autopilot. Because of this, it is the height to the next checkpoint which is controlled in this chapter. Alteration of the required climb indirectly affects the throttle and therefore the pitch.

Indirect control of the pitch means that in this chapter the parameters which are varied are the throttle as well as the height to the next checkpoint. The domain theory is again shared persistently between flights. Other parameters are bound by maintaining a set route.

The domain theory only contains rules to determine state, reason about relative orders of magnitude, and reduce the throttle of the aircraft. There are no rules to control pitch. The domain theory is constructed in a way which could support analogical reasoning. The inputs and the output are abstracted such that predicate replacement can occur. Two rules are given below. The first is from an early iteration of the domain theory, while the second is of a form which lends itself more easily to analogy.

reporting(vvi_dial_fpm,P) $\wedge$ reporting(total_energy_fpm,E) $\wedge$ smaller(P,E)
$\wedge$ one_smaller(throttle_ratio_all,N) $\rightarrow$ intend(throttle_ratio_all,N)

throttle_command(I) $\wedge$ height_to_go(H,I) $\wedge$ velocity_sensor(C,I) $\wedge$ reporting(H,P) $\wedge$ reporting(C,E) $\wedge$ smaller(P,E) $\wedge$ one_smaller(I,N) $\rightarrow$ intend(I,N)

By including the *throttle_command* predicate there is the possibility for an impasse here. This impasse can result in an analogical replacement, expanding the rule to control another output.

Additionally, the output is no longer hard coded into a parameter. This could aid the flexibility of the domain theory in a deductive sense. By abstracting the inputs and output, the relationship is made to apply to different sources. For example, if the domain theory included other statements which unify with *throttle_command(I)* then rules for multiple outputs could be derived. This can lead to a different specialisation of *I*. The specialisation of $I$ in this rule can also impact which inputs are selected in a particular explanation as $I$ features in other atoms.

Changes were made to the domain theory to support the formation of specific analogies. Rules which have shared terms have been added to the domain theory to support the generation of analogies. These rules embody the additional contextual information about concepts. This additional information, unnecessary to the intended model, may be easier than eliciting a minimal domain theory.

It is worth noting that extra information may not always need to be added in order to form analogies. Consider if the *intend* predicate, which is used in the reduction strategy, were instead *intend_reduce*. If there was also an increase strategy using the rule below then, by analogy, different inputs could be controlled which have an inverse relationship. This is because the similarities between *intend_reduce* and *intend_increase* could form an abstraction. By inverse relationship it is meant that reduction should occur when $P$ is bigger than $E$. This could expand such a domain theory in a useful way by leveraging the *intend_increase* predicate rather than the *intend_reduce* predicate.

throttle_command(I) $\wedge$ height_to_go(H,I) $\wedge$ velocity_sensor(C,I) $\wedge$ reporting(H,P) $\wedge$ reporting(C,E) $\wedge$ bigger(P,E) $\wedge$ one_bigger(I,N) $\rightarrow$ intend(I,N)

Conversely, it is true that this could lead to too many reduction rules. Some rules would be generated by deduction, some by analogy. Almost all situations could then yield a rule for reduction. Too many rules would be overzealous as there are times when throttle is required. It is therefore important to discriminate between useful rules. In this experiment the possible analogies are limited by the form of the domain theory.

For example, the rule for reduction itself checks that the new value is smaller than the previous value of the output variable. This ensures that, regardless of the analogies employed further down the tree, the output cannot increase when explaining the *reduce* goal. If this premise were to be violated then the *smaller* predicate would lead to an impasse. The consistency of the domain theory as it relates to orders of magnitude should be maintained. The results of *smaller* and *bigger* need to be constant in order to meaningfully relate magnitudes.

Along with the additions to facilitate specific analogies, the domain theory is given in Table 5.2. The table also includes the goal used to derive rules.

| **Goal** |
|---|
| reduce(height_to_go,Y) |
| **Domain Theory:** |
| **Rules** |
| ***numbers*** |

bigger(X,Y) → smaller(Y,X)
!same(X,Y) → different(X,Y)
!bigger(Y,X) → biggest(X)
!smaller(Y,X) → smallest(X)
!same(X,Y) ∧ bigger(X,Z) ∧ bigger(Z,Y) → bigger(X,Y)
reporting(I,X) ∧ bigger(X,Y) → more_than(I,Y)
reporting(I,X) ∧ smaller(X,Y) → less_than(I,Y)
reporting(I,X) ∧ same(X,none) → none(I)
more_than(I,none) → some(I)
reporting(I,X) ∧ same(X,Y) → is(I,Y)
reporting(I,X) ∧ started(I,Y) ∧ same(X,Y) → unchanged(I)
reporting(I,X) ∧ started(I,Y) ∧ !same(X,Y) → changed(I)
reporting(I,X) ∧ smaller(Y,X) → one_smaller(I,Y)
reporting(I,X) ∧ bigger(Y,X) → one_bigger(I,Y)

***reduction strategy***

reporting(I,X) ∧ intend(I,N) ∧ bigger(X,N) → reduce(I,N)
throttle_command(I) ∧ height_to_go(H,I) ∧ velocity_sensor(C,I) ∧ reporting(H,P) ∧ reporting(C,E) ∧ same(P,E) ∧ one_smaller(I,N) → intend(I,N)
throttle_command(I) ∧ height_to_go(H,I) ∧ velocity_sensor(C,I) ∧ reporting(H,P) ∧ reporting(C,E) ∧ smaller(P,E) ∧ one_smaller(I,N) → intend(I,N)

***state***

some(parking_brake) ∧ less_than(throttle,medium) ∧ none(altitude) → state(idle)
none(parking_brake) ∧ less_than(throttle,medium) ∧ none(altitude) → state(taxi)
none(parking_brake) ∧ more_than(throttle,small) ∧ none(altitude) → state(takeoff)
current(takeoff) ∧ none(parking_brake) ∧ some(altitude) → state(flight)

***analogy***

controls(engine_speed) ∧ affects(air_speed) ∧ affects(acceleration) ∧ provides(thrust) ∧ reflects(power) → throttle_command(throttle)
controls(elevators) ∧ affects(altitude) ∧ affects(air_speed) ∧ pitches(aircraft) → pitch_command(height_to_go)
reflects(motion) ∧ affected_by(throttle) → velocity_sensor(velocity, throttle)
reflects(climb) ∧ affected_by(pitch) ∧ affected_by(throttle) → climb_sensor(climb_rate, height_to_go)
change_in(y) ∧ affected_by(throttle) ∧ affected_by(pitch) → height_to_go(height_to_go, throttle)
change_in(x) ∧ affected_by(throttle) ∧ affected_by(pitch) → ground_speed(groundspeed, height_to_go)

**Statements**

***numbers***

bigger(full,huge)
bigger(huge,large)
bigger(large,medium)
bigger(medium,small)
bigger(small,tiny)
bigger(tiny,none)

| same(X,X) |
|---|
| ***analogy*** |
| throttle_command(throttle) |
| pitch_command(height_to_go) |
| height_to_go(height_to_go, throttle) |
| velocity_sensor(velocity, throttle) |

**Table 5.2: Analogical Domain Theory**

The use of a single goal for this experiment also serves to limit the possible analogies. This leaves the domain theory with an inherently hierarchical structure.

It is worth noting that, in the current implementation, analogies are not added to the domain theory as new rules. The (sub)goal causing an impasse is swapped for the target with the larger correspondence i.e. the one which formed the most complex derived class. Additionally, the analogous nodes are not included in the fuzzy rule derivation process. These nodes are omitted because the nature of the link between the source and target terms is assumed to be irrelevant to the fuzzy rule. Instead the analogy is viewed more as evidence, or an operational node in support of the swapped node. This allows the explanation to continue without adding goals which would not have been present in a non-analogical explanation structure, minus the term that has been replaced.

In this experiment the outputs of forming an analogy are any fuzzy rules generated by employing said analogy. As no rules can be formed deductively, all rules will be formed in this experiment by analogy. Any rules generated can, depending on their impact on the flight, sit in support of the technique. The aim of the experiment is to show the possibility that the technique can be used to generate useful rules after knowledge engineering has taken place.

### 5.3.2 Results

#### Initial Results

The goal given to the system is to reduce the pitch, by altering the height to the next checkpoint. This is featured in Table 5.2. An explanation structure can be formed using the domain theory and the given goal. Impasses are reached, and are shown in red in Figure 5.8. Additionally, the internal names from the simulator are used, but human-readable variants are included in the domain theory table. It is worth noting that the implementation denotes $\wedge$ with the symbol "+".

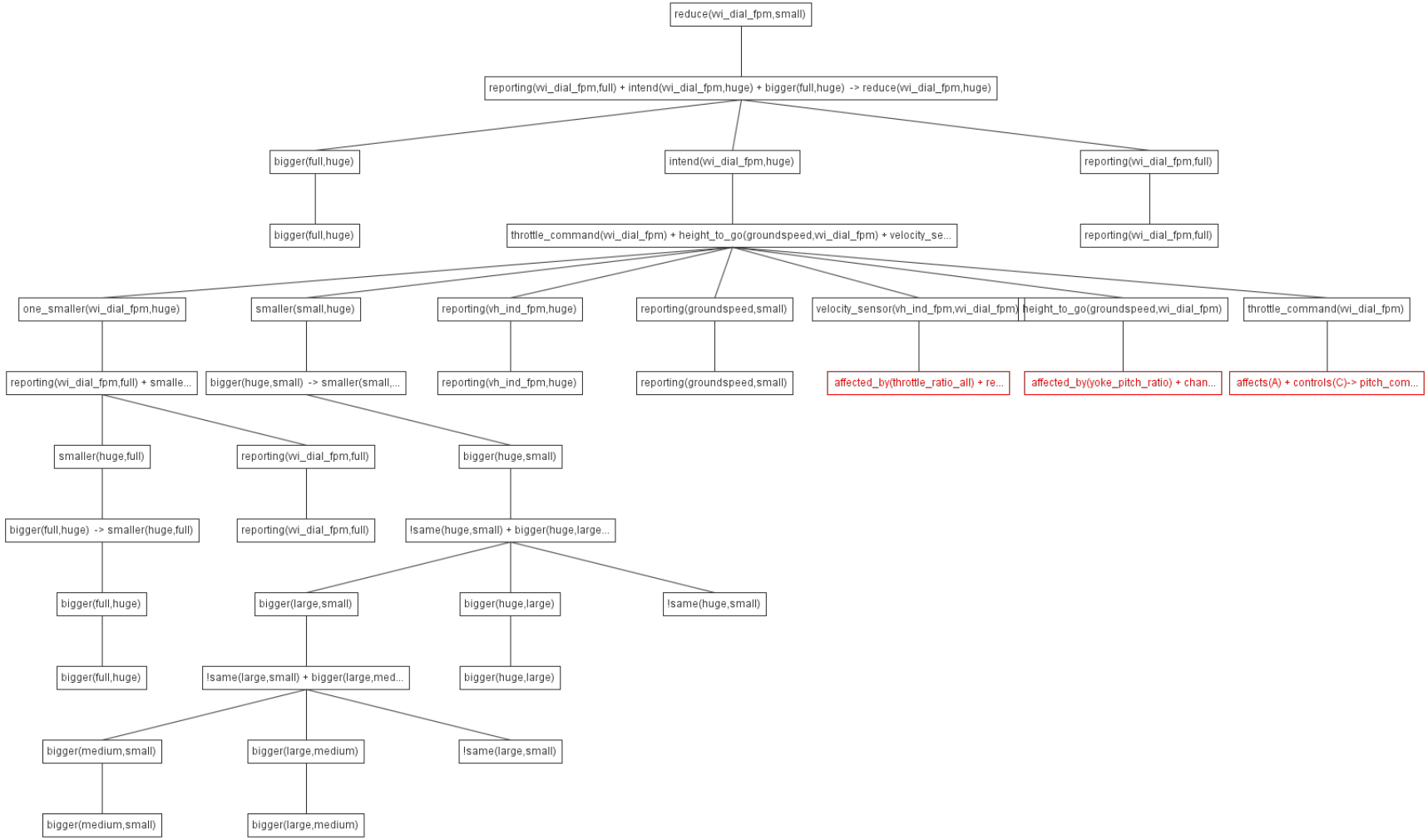


**Figure 5.8: Sample EBL explanation structure for deriving pitch reduction rules upon impasse.**

These impasses can be avoided by forming analogies. The analogies in this case replace the inputs being compared and the output, but maintain the same overall structure as the throttle derivation from the previous chapter. The output is swapped from the throttle to the height to the next checkpoint. The inputs were also altered. The velocity sensor is swapped for the height to go to the next checkpoint. The height to go is swapped for ground speed.

Swapping the inputs follows a simple rationale. When the ground speed is smaller than the height to go to the next checkpoint, reduce the climb rate. The climb rate could be calculated as the gradient: $\frac{\delta y}{\delta x}$. A larger gradient, and steeper climb, would come about when the change in x ($\delta y$) is larger than the change in y ($\delta x$). This is embodied in Figure 5.8 by considering ground speed as a change in x and height to go as the change in y. Therefore reduction occurs when ground speed is smaller than the height to go to the next checkpoint.

Each measurement is fuzzified and interpolated and is therefore relative to itself. This is because the range over which the linguistic values are interpolated is determined by past values. The rules generated are dependent on the number of fuzzy sets and refresh rate of inputs. Both of these factors influence the states which are considered by EBL, as well as the granularity of each rule.

Note that the output is taken as an input. This output is also written to through the switching mechanism. Having one input being written to by two systems means that timing can affect the rules learned, since the input will appear to have one value or another, depending on when it is sampled. This is one motivation for using fairly large data sets and averaging. It is worth noting that in this experiment the autopilot is always operated at 10Hz and the controller is initially operated at the same rate.

The analogies formed, by the proposed algorithm, in order to swap the inputs and output are given below. The derived concept is labelled as the target concept.

- affected_by(throttle_ratio_all) $\wedge$ reflects(R)$\rightarrow$ climb_sensor(climb_rate, height_to_go)
- affected_by(pitch) $\wedge$ change_in(C)$\rightarrow$ ground_speed(groundspeed, height_to_go)
- affects(A) $\wedge$ controls(C)$\rightarrow$ pitch_command(height_to_go)

These analogies allowed fuzzy rules to be generated. A set of 50 flights was conducted. The average altitude, both with and without EBL, can be compared to show that the rules had an effect. These are given in Figures 5.9 and 5.7 respectively.

The standard deviation shows the areas most affected by the derived rules. Large standard deviations can be caused by differences in aircraft behaviour at the same point on the time axis. As can be seen, the effect is concentrated at the start of a climb and in the later section of flight (from about 5.5 minutes to 7 minutes).

The controller's effect being concentrated at the start of a steep climb may be a result of the inputs being measured relative to themselves, rather than a scale that applies to both. The rate of increase in either input may be partially hidden. When both inputs are increasing then a small change, relative to its own scale, might be significant for an input. However, it is possible such an increase is not significant to the relationship being appealed to in the domain theory. Additionally, the baseline throttle control is rather binary. A binary throttle controller leads to a high ground speed when climbing, even when the climb is steep. Investigation of an interpolated fuzzy relationship for the comparison of two inputs could prove an interesting area for future work.

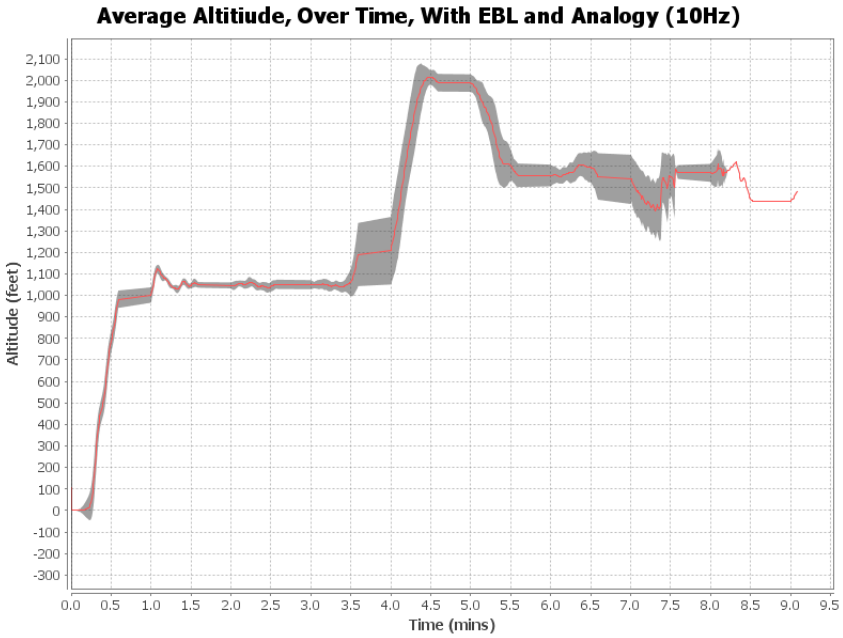


**Figure 5.9: Average altitude of the aircraft. Analogical EBL controller operating at 10Hz.**

The initial section of the steep climb is reduced drastically, then returns to a steep climb. This is a less granular control of throttle than that demonstrated in Chapter 4. The loss of granular control is related to the throttle output only being effected in a binary manner, based on the finer control of another output (height to go). This means that a reduction in pitch may lack the finer control required to approach optimality.

An effect, resulting in a general lowering of throttle output and therefore less dramatic pitch changes, can be illustrated by examining the throttle behaviour. Examples, of the throttle output, both with and without EBL, are given in Figures 5.10 and 5.11 respectively. It is worth noting that Figure 5.10 is taken from a flight after the point where new rules had ceased to be generated.

The effect of the rules, from these two examples, appears to be a general reduction in extreme levels of throttle output. However, more fluctuations are present. Fluctuations could cause an increase in the distance travelled without saving sufficient energy to have a net positive effect. Also, some of the larger plateaus have fluctuations which are small. These fluctuations come from the effect of the fuzzy rules competing with the auto-throttle. This shows that the effect of the fuzzy controller may be truncated by competition with the auto-throttle. The degree of competition with the auto-throttle is affected by the switch timings.

The number of rules generated, by run, is shown in Figure 5.12. A full listing of the rules generated is given in Table 5.3.

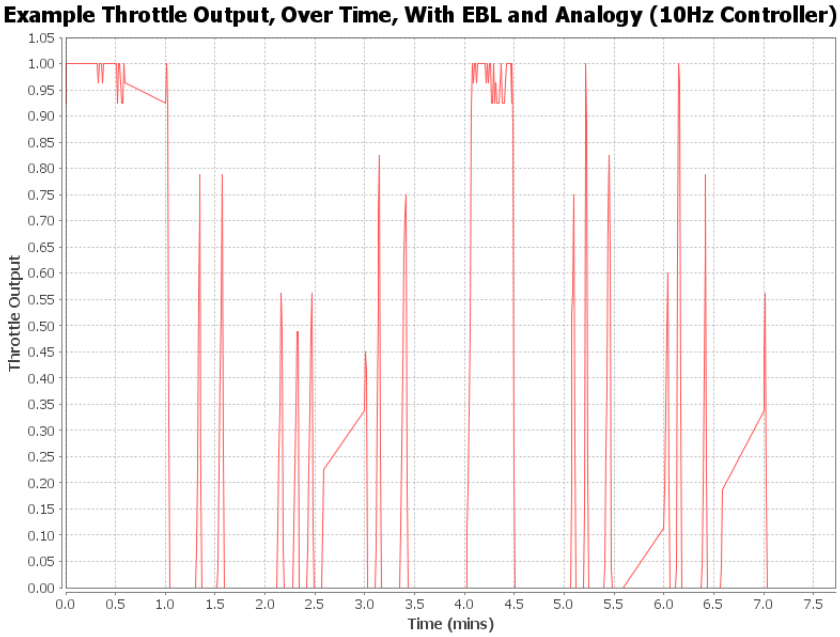


**Figure 5.10: Throttle control value of the EMG-5 aircraft over a single flight. Analogical EBL controller operated at 10Hz.**

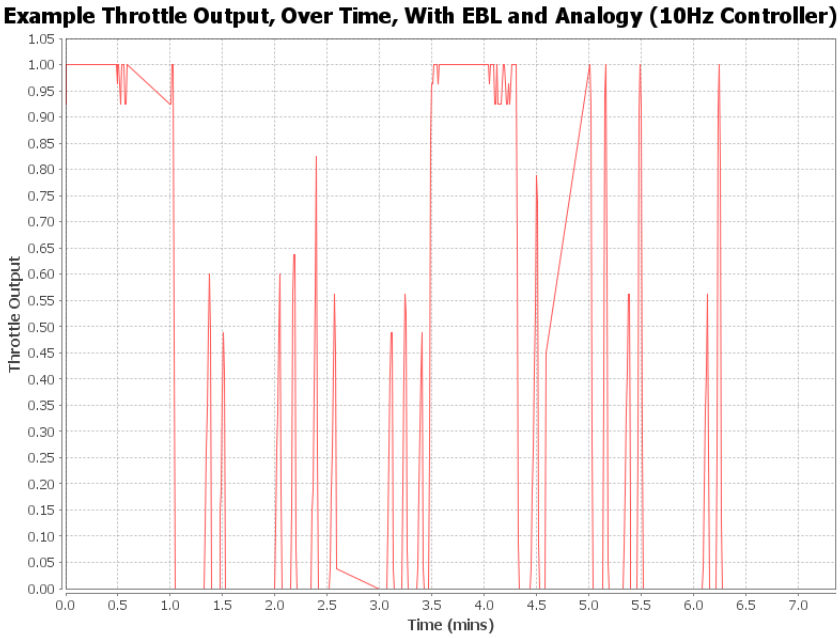


**Figure 5.11: Throttle control value of the EMG-5 aircraft over a single flight without analogical EBL.**

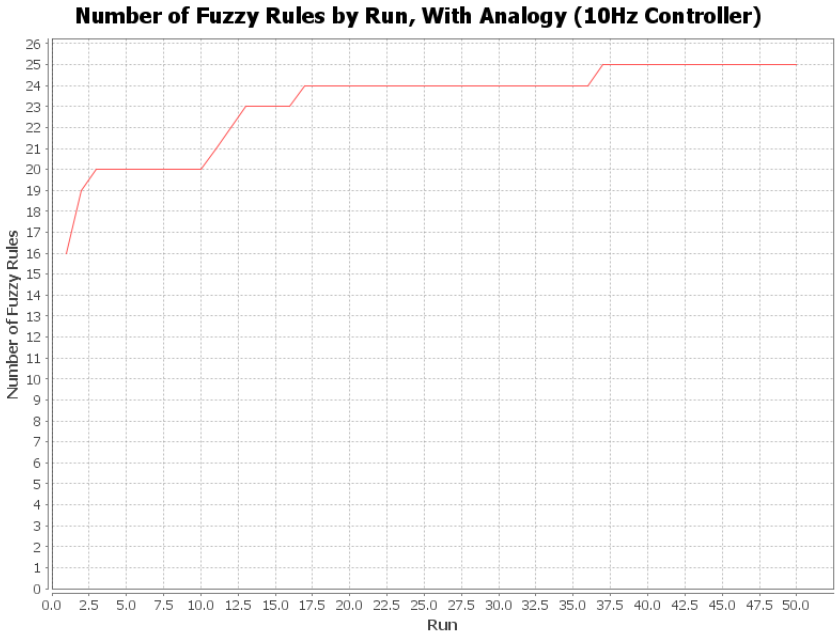


**Figure 5.12: The number of rules generated by the end of each flight is given for the 10Hz controller.**

| Run | Input | | | Output |
|---|---|---|---|---|
| | ground speed | climb rate | Height to go | Height to go |
| 1 | large | large | small | tiny |
| 1 | medium | medium | medium | small |
| 1 | huge | huge | medium | small |
| 1 | huge | huge | huge | large |
| 1 | small | small | medium | small |
| 1 | medium | medium | tiny | none |
| 1 | medium | medium | small | tiny |
| 1 | small | small | tiny | none |
| 1 | huge | huge | large | medium |
| 1 | huge | huge | tiny | none |
| 1 | small | small | huge | large |
| 1 | huge | huge | small | tiny |
| 1 | large | large | medium | small |
| 1 | large | large | large | medium |
| 1 | none | small | medium | small |
| 1 | small | small | small | tiny |
| 2 | medium | medium | huge | large |
| 2 | large | large | tiny | none |
| 2 | medium | medium | large | medium |
| 3 | large | large | huge | large |
| 11 | medium | medium | medium | none |
| 12 | small | small | large | medium |
| 13 | medium | medium | small | none |
| 17 | medium | medium | huge | small |
| 37 | large | large | medium | tiny |

**Table 5.3: Table of the fuzzy rules generated, by run, with analogy. The controller was operated at 10Hz.**

The sharp increase of rules in the final run may indicate that more rules may have been generated given a larger number of runs. The fuzzy rules, generated by analogical EBL, affected the remaining battery charge. The remaining battery charge is plotted by run in Figure 5.13. The blue line shows the runs without EBL enabled. The red line shows the runs with EBL enabled.

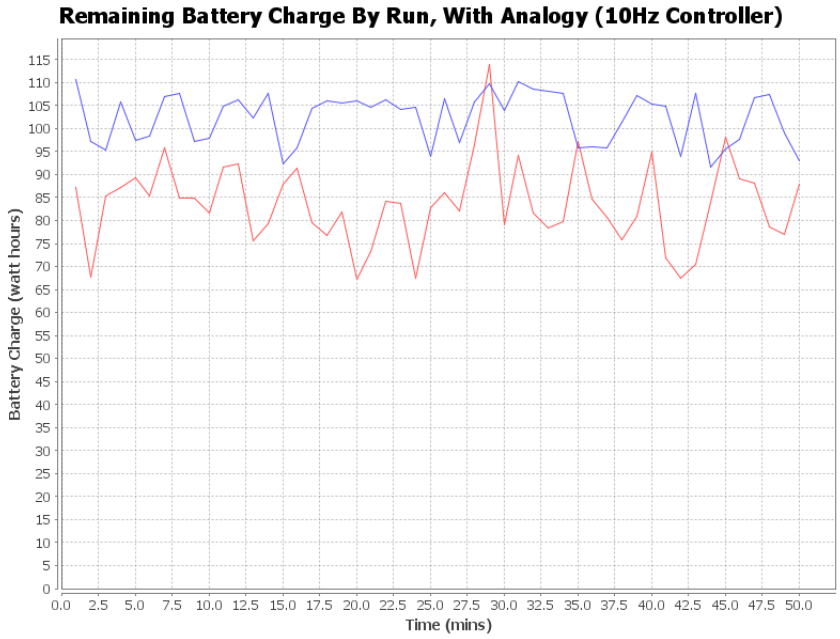


**Figure 5.13: Battery charge remaining at the end of each run. Analogical EBL controller operated at 10Hz.**

Figure 5.13 shows that the rules had a negative effect on efficiency, in general. This is likely due to the large reduction in altitude between about 5.5 and 7 minutes. This results in a longer path taken to reach the final checkpoint.

The fuzzy rules are all part of a reduction strategy, which sits in opposition to the auto-throttle ascending to a new checkpoint. The switch settings alter the balance between these two factors. This was illustrated by small undulations in the plateaus on Figure 5.10, which show the throttle behaviour. The assumption that the autopilot represented an opposing strategy may no longer hold; the rules generated by analogy may not be representative of a strategy designed using expert knowledge. As the assumption no longer holds, the effect of the switch timings is further examined in a series of flights using different sampling rates for the controller.

### Further Results

Sets of 50 flights were again conducted. In each case the switch setting for the fuzzy controller was altered. The remaining charge, compared to operation without EBL enabled is given, as well as

the number of rules generated per flight. The aim is to elicit patterns between the switch settings, the efficacy and the number of rules generated. The following switching rates are presented: 6.6Hz (every 0.15 seconds), 5Hz (every 0.20 seconds), 4Hz (every 0.25 seconds), and 3.3Hz (every 0.30 seconds). The figures for each result set are given below:

- The number of rules generated for a 6.6Hz controller is given in Figure 5.14. The remaining battery charge is shown in Figure 5.15.
- The number of rules generated for a 5Hz controller is given in Figure 5.16. The remaining battery charge is shown in Figure 5.17.
- The number of rules generated for a 4Hz controller is given in Figure 5.18. The remaining battery charge is shown in Figure 5.19.
- The number of rules generated for a 3.3Hz controller is given in Figure 5.20. The remaining battery charge is shown in Figure 5.21.

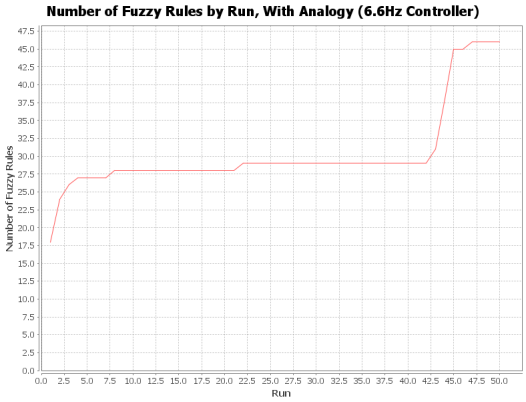


**Figure 5.14: The number of rules generated by the end of each flight is given. The analogical EBL controller was operated at 6.6Hz.**

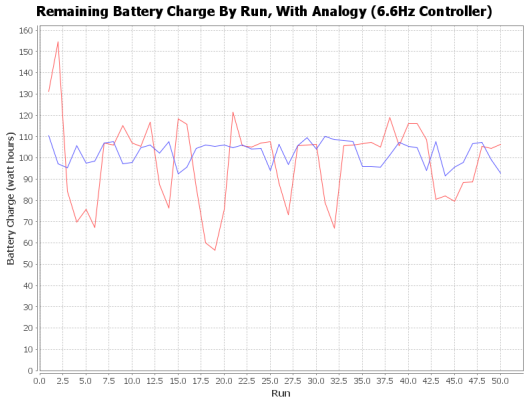


**Figure 5.15: The battery charge remaining at the end of each run. Analogical EBL controller operated at 6.6Hz.**

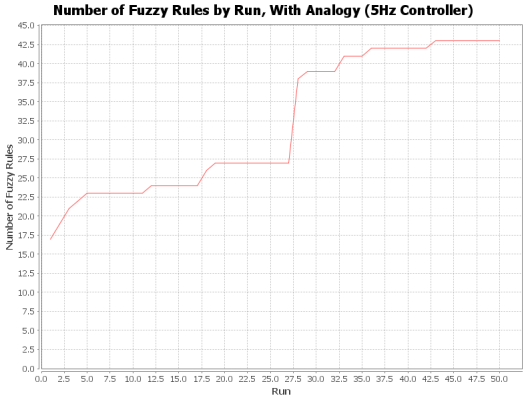


**Figure 5.16: The number of rules generated by the end of each flight is given. The analogical EBL controller operated at 5Hz.**

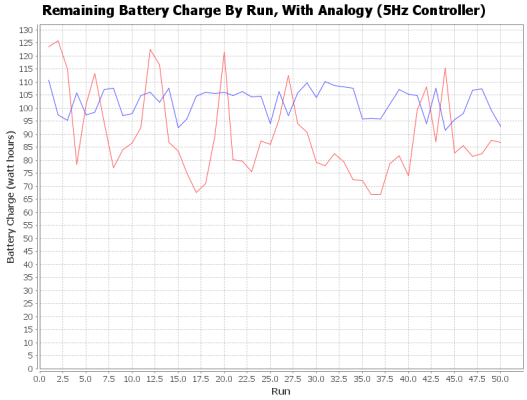


**Figure 5.17: The battery charge remaining at the end of each run. Analogical EBL controller operated at 5Hz.**

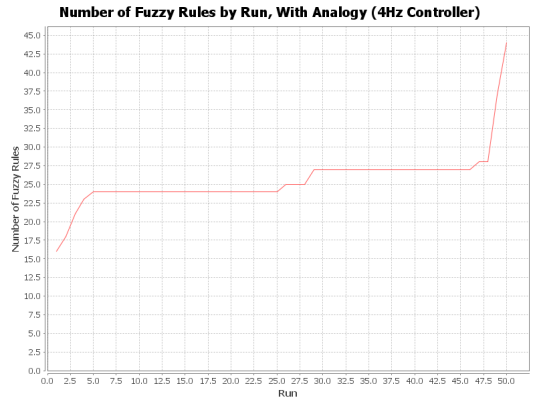


**Figure 5.18: The number of rules generated by the end of each flight is given. Analogical EBL controller operated at 4Hz.**

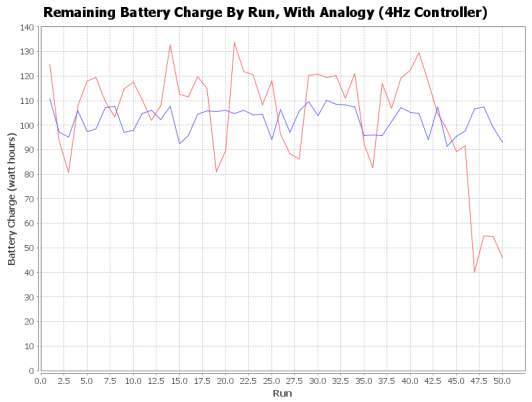


**Figure 5.19: The battery charge remaining at the end of each run. Analogical EBL controller operated at 4Hz.**

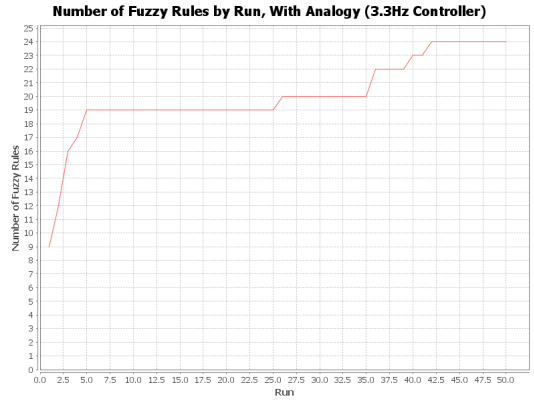


**Figure 5.20: The number of rules generated by the end of each flight is given. Analogical EBL controller operated at 3.3Hz.**

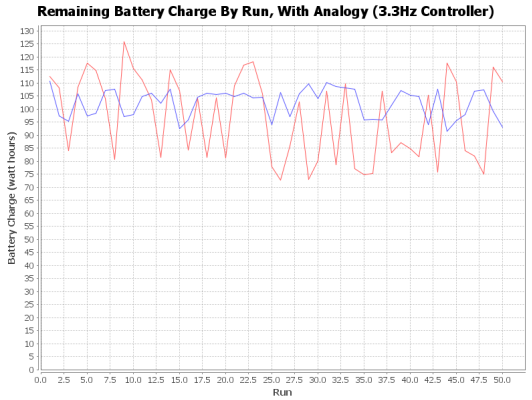


**Figure 5.21: The battery charge remaining at the end of each run. Analogical EBL controller operated at 3.3Hz.**

More rules were generated when the controller operated at the highest and lowest frequencies. Rule generation was most prolific at 6.6Hz. The 'height to go' input, which is also the output, is written to by both the auto throttle and fuzzy controller. Rule generation is guided by the input value. This balance may simply have exposed the EBL algorithm to more situations than other sampling rates.

Some rules may come from seeing an output from a previous controller execution, rather than the value from the autopilot. The input may be insufficiently reduced to cause a change in pitch and the algorithm may produce further rules as a consequence. Not all rules generated are guaranteed to affect the throttle on their own, since the auto throttle is specified as being concerned with the sign of the value. Therefore several rules may need to be generated in turn, and executed in concert, to result in an effect. This could complicate a study of the impact of individual rules in this case.

The remaining charge was generally improved by the 4Hz controller, but a significant loss in efficiency was evident in later runs. This may imply that rules generated earlier in this case were of a greater utility. This implies that even when useful rules are generated from an analogy, invalid rules may also be generated. An algorithm for adding caveats to an analogy in order to prevent detrimental rules from being learned could be a fruitful aspect of further work. This would likely

require clearer data on the individual impact of each rule, both alone and in conjunction with others. Both cases need to be considered, as multiple rules may combine to produce a positive effect where only one would not.

## 5.4 Conclusion

An approach for incorporating analogy into EBL has been presented. The technique could be used to further generalise rules in a system by linking previously separate concepts. These links are formed by deriving an abstraction. Doing this may imply that whatever reasoning held in deriving the original rule analogously holds for the new hardware. Analogy is an example of transfer learning.

Transfer learning can be applied as an alternative to an impasse, such as in [71]. This technique can operate where an impasse is reached due to a mismatch between outer predicates with the same form. The reason for addressing this type of impasse is that EBL, by generalising, already avoids some potential impasses due to differing parameters.

By applying this technique EBL can make inductive leaps when deductive explanation fails. However, this entails all of the drawbacks of making inductive leaps. Given that the potential costs are weighed against the possibility of no gains, the technique should be applied conservatively.

This technique could be useful in a modular power management system. By noting that a similarity between concepts exists, previous reasoning can be assumed to hold for a new component. If these analogues are made explicit they could be used in situations other than the one that gives rise to them.

Whilst the rate at which the controller was operated had an impact, no clear ideal rate was elicited. Each set of results had both positive and negative flights. The potential to derive rules which positively affect efficiency is illustrated by each positive flight.

However, flights which performed significantly below the baseline could imply that the technique requires further work. Each rule should ideally be evaluated for its individual impact, but where the derivations are only possible by analogy it is likely that there may be no information for evaluation. Deviation from known mission parameters or principles could be possible. Further work into eliciting a general evaluation strategy for analogies would be beneficial.

It would be particularly beneficial to glean more from the underlying correlation than just a connecting analogy. Each analogy holds only in certain circumstances. For this reason, formation of analogy may be considered the start of an ongoing learning task.

Some rules were generated which satisfy the objective of this experiment, that being the generation of rules for previously unseen hardware. This validates the potential of the technique in terms of modular management. However, the nature of the results illustrate that rules other than ones with a positive impact on energy management were produced. This means that the intent of the management system was not realised during every rule generation. The discrimination of useful rules should be the focal point of further work.

This chapter has demonstrated that the proposed technique can generate rules, via a simulated application, which are applicable to previously unseen hardware. This is achieved by extending

the closure of the domain theory. These rules support the potential of the technique to apply to modular control, which satisfies the second objective of this work. Satisfying the first and second objectives has resulted in an approach which fulfils part of the aim of this project. Thus far the workload engendered by hardware changes has been reduced. In order to fulfil the aim of this project the corresponding certification workload needs to be investigated.

It remains to consider the Fuzzy-EBL controller being implemented in an industrial context. The main issue which arises when using an adaptive control technique in the aerospace industry is the resulting certification difficulties. Addressing these certification difficulties is the topic of the next chapter.

# Chapter 6

# Explanation-Based Learning and Adaptive Certification

## 6.1 Introduction

The aim of the project will be fulfilled when the certification workload of managing a modular platform can be reduced. The third objective of this work is to extend the MPMS to generate certification artefacts. This chapter is concerned with the fulfilment of the third objective.

The previous chapters proposed an adaptive control system, applied to a UAS. In order to utilise this system in the aviation industry additional work would be required. This is because, in aviation, software requires certification. Certification of software as part of a platform can be gained by any sufficient safety argument. Addressing the guidelines in DO-178 is one way to pursue a safety argument. Adherence to DO-178 relates to the software system and is part of a larger safety argument for the whole platform, whose different aspects have different guideline documents.

This chapter posits that it is rational to have an adaptive safety argument for an adaptive system. The preliminary work for this chapter was published as [58]. The position is illustrated by extending EBL to generate both control laws and safety cases for them, represented using GSN. Each rule derivation generates one GSN safety case. Safety cases link claims to evidence, which mirrors the way that EBL links a goal to operational nodes via a domain theory.

The structures produced by EBL can be augmented to form skeletal safety cases. The structures are first visually expressed using GSN. Then each node in the EBL structure is processed as a new goal using a safety-specific domain theory. The explanations of the sub-goals are added to the structure. The augmented structure can reflect a learning system by adapting alongside the controller.

An augmented explanation structure can be constructed for each rule that has been generated for a platform. The suite of augmented structures, along with the rules they relate to, may be sufficient to form a substantiated safety argument. Generating safety artefacts which mirror an adapting controller can reduce recertification effort. Each time the controller adapts, the safety cases can automatically update to mirror these changes. This reduction of effort addresses part of the aim

of the project.

The augmented structure gives a major benefit. Fuzzy rules normally only offer information about the input and output states. This can limit the assurance of a fuzzy rule. However, rules generated in this thesis can produce more detailed information pertaining to their generation. This can include information about the processes, assumptions or reasons for connecting an input state to an output state. It is posited that the greater transparency of this approach will be beneficial to the certification of fuzzy rules.

The 4+1 model of fundamental principles of software safety assurance was proposed in [72]. The model is comprised of principles which appear in standards and best practices across both domains and projects. These principles are:

- **1:** Software safety requirements shall be defined to address the software contribution to system hazards.
- **2:** The intent of software safety requirements shall be maintained throughout requirements decomposition.
- **3:** Software safety requirements shall be satisfied.
- **4:** Hazardous behaviour of the software shall be identified and mitigated.
- **4+1:** The confidence established in addressing the software safety principles shall be commensurate to the contribution of the software to system risk.

The approach proposed in this chapter concerns principle four. The approach makes the rationale behind the derivation of a rule more explicit. The artefacts produced tie the safety requirements to specific rules and the assumptions that lead to their generation. Unintended software behaviour can be the result of design decisions during development [73]. This link between rules and requirements help to keep the design decisions made by the automatic generation of rules explicit.

Additionally, it is possible that the approach could relate to principle three. There may be cases where demonstrating that some body of rules can never be generated would constitute evidence that certain hazardous behaviours will not be produced. It is worth noting that this approach, concerned with the functionality of generated rules, fits with a performance based approach, as in [55], rather than a process based approach to software assurance.

However, the proposed approach does not impact all areas of software assurance. Firstly, the identification of systems hazards and their appropriate attribution to areas of the software functionality is not addressed. Secondly, it has not been established what level of confidence this approach should confer. However, static checkers could analyse the logical validity of the generated structures and metrics proposed for analysing the utility of individual rules. Thirdly, maintaining the intent of safety requirements during decomposition is outside of the scope of the PhD. Finally, the domain theory used requires its own assurance argument.

A recurrent theme is that the safety argument is important [47], more important than showing that the system meets the specification [60]. This is not surprising given that DO-178, whilst sometimes treated as a standard, is a guideline. The guideline is a suggested method of pursuing a safety argument via good practice. It is therefore the argument that is key, rather than the method of presentation.

A technique that has gained some influence in presenting safety arguments is the safety case. A safety case links claims and evidence into hierarchical arguments, though the link between claims and argument is not always explicit. An argument for certification can be presented using safety cases. However, several areas present additional issues within the domain of certification. Adaptive control is an area highlighted as having additional challenges in an entirely specification-based framework. Adaptive control may benefit from having an adaptive safety argument [60].

Safety cases have been considered in relation to certification in other works. Safety cases as part of certification are discussed in [47]. Whilst there are issues, it may be possible to generate and check safety cases to form part of a solid safety argument [48]. Changes in control rules, by a system which is able to adapt over time, should cause the generation of a set of updated safety cases. The generated cases will, until further learning occurs, illustrate the coverage of the system.

Simulation of the flight envelope can be used to train the system, generating rules and determining the distribution of the fuzzy sets. A trained system, which has ceased to adapt, along with a suite of safety cases could be used to demonstrate determinism. When operating outside of the certified flight envelope, such as in the event of a failure, it may be possible to continue these learning tasks.

Consider a system that is certified and the adaptive elements of which are supported by safety cases. If the rest of a certification argument holds, and the safety cases remain valid, then the certification holds whilst the adaptive safety cases are valid. Of particular import is the tying of evidence to the claims made by the domain theory. Some work has gone into considering both adaptive safety arguments and the verification of them [74],[48].

An adaptive safety argument, when coupled with evidence and analytical tools, could be used to form the adaptive portion of an otherwise standards-based certification argument. If the rest of the argument holds then the argument should hold for any state where the adaptive safety cases remain valid. In this way EBL could be applicable to the aviation domain. Other approaches could be incorporated into adaptive safety cases, such as the methods of applying formal mathematical assessment techniques to autonomous agents, which could be explanation-based agents, developed by QinetiQ [50]. An adaptive system, using EBL to generate GSN safety cases, could reduce certification workload, which is the objective of this chapter.

The remainder of the chapter is presented as follows: Section 6.2 considers how the parallel between EBL and GSN can be leveraged in terms of certification. Section 6.3 describes the process of generating adaptive safety cases with EBL. Example EBL-GSN structures are presented and discussed in Section 6.4. Section 6.5 concludes the chapter.

## 6.2 Adaptive Justification

Safety cases can be used to present an argument that a system is safe for use in aviation. Adaptive systems could require the adaptation of the safety cases to remain valid for each adaptation. The automatic generation of safety cases for control rules would be advantageous. This could alleviate some of the workload associated with the recertification of a system after a change has been made.

Generated safety cases could, in concert, present a safety argument. The safety argument could sit in support of an application for certification. This is provided that the system proposed in previous chapters is trained offline and online learning is disabled within the flight envelope. The

proviso is given due to the fact that there is no sufficiently trustworthy method of automatic validation that could operate alongside changes, justifying them in real-time. Adaptations require re-certification and therefore learning should not take place in flight as re-certification cannot take place in flight. Two principal properties can be demonstrated by such a safety argument: Coverage, and determinism.

EBL is used to generate control rules and is an adaptive technique. EBL is adaptive because explanations form new rules that can change the action of the controller for a given situation. This is analogous to changing the gains of the controller.

EBL has already been applied within the aerospace domain [75]. One reason for the inclusion of EBL in this work is that EBL explanations help to justify situation-action bindings and thereby support the determinism property, which is key to establishing a safety argument. The situation-action bindings are justified by the explanation structure. The structure contains the assumptions and deductions that derive a particular rule. The goal represents an action and the leaf nodes a situation, while additional parts of the structure represent the reasoning behind a rule derivation.

A set of explanation structures for every fuzzy rule would show, when unconstrained by training information, all rules that are used by the system. However, an explanation structure itself would not constitute even a skeletal safety case. A starting point would be to express explanation structures in a safety case notation.

GSN is a graphical notation used to present safety cases in a manner that aims to minimise ambiguity and avoid poor writing. GSN presents arguments as a hierarchy of nodes which represent supporting arguments. There are nodes for representing goals, strategies for achieving goals, solutions (evidence to satisfy arguments), contextual information and unfinished goals. Figure 6.1 shows the components of a GSN diagram. Figure 6.2 is an example GSN safety case.

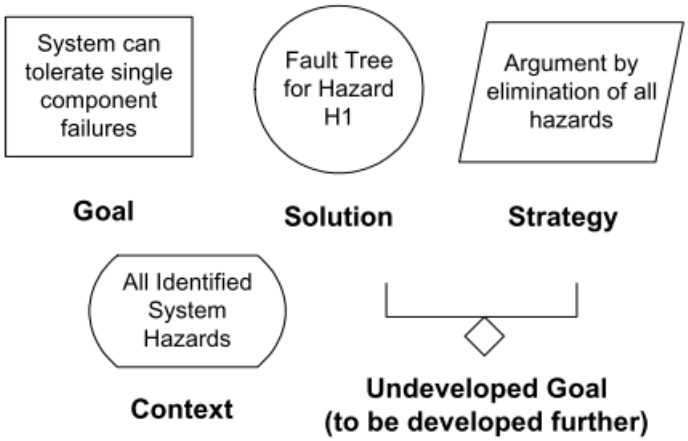


**Figure 6.1: GSN nodes and their purposes. Taken from [1].**

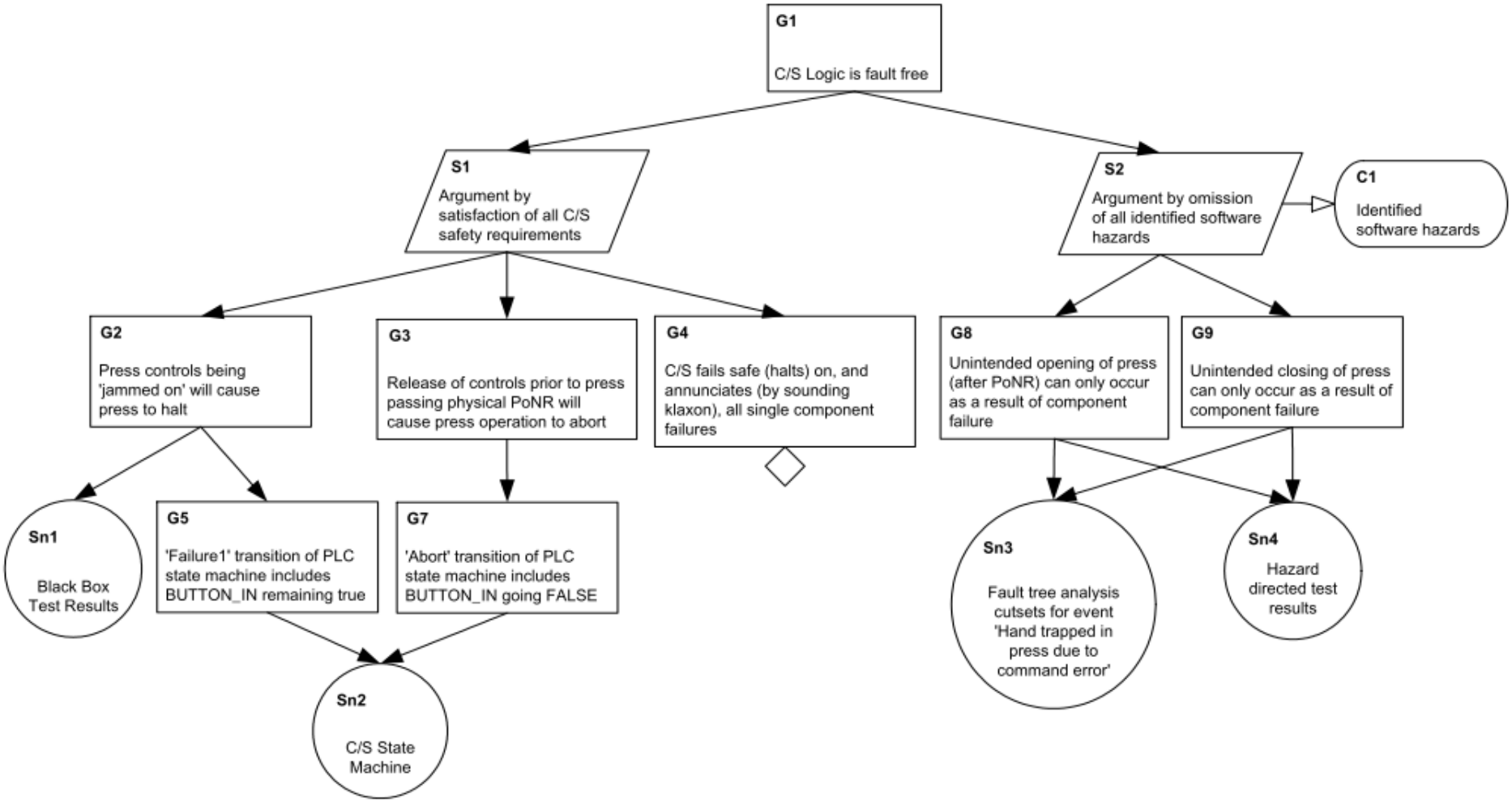


**Figure 6.2: An example safety case using GSN nodes. Taken from [1].**

There is a certain parallel in the representations of EBL and GSN, given in Table 6.1. Both are a hierarchical decomposition headed by a goal. Both structure an argument via a series of interconnecting nodes. Both may eventually terminate in branches where a claim is substantiated. The goal nodes in each representation need to be considered analogous in order for the comparison to hold. Context nodes, which show when a node applies, could be considered similar to preconditions. These preconditions could be rule meta-data in EBL, preconditions on operational actions or even embedded as rules in the domain theory. Solution nodes would be analogous to unifications to operational nodes in EBL. Undeveloped goals need no analogue, but could represent dearths in domain knowledge.

| EBL | GSN |
|---|---|
| Hierarchical structure based on goal decomposition. | Hierarchical structure based on goal decomposition. |
| Goal Node. | Goal Node. |
| Unification with observation. | Solution Node. |
| Unification with domain knowledge. | (Sub)Goal Node / Strategy Node. |
| Preconditions. | Context Node. |
| Conceptual gaps in knowledge | Undeveloped Goal. |

**Table 6.1: Parallels in representation between EBL and GSN.**

Since there is a parallel in representation it is possible to represent an EBL explanation using GSN nodes. Presenting explanations in this form could form a (partial) safety case, depending upon the system. This is advocated as the first step towards generating an adaptive safety case in this work.

The next step is to augment the explanation structures with more typical safety case nodes. GSN nodes, which would have appeared in a hand-crafted GSN structure for the fuzzy rules, comprise the safety-specific domain theory. Unification is the mechanism which connects nodes in the EBL structure. The safety-specific domain theory can take advantage of unification to add safety case nodes to the EBL structure.

A system incorporating EBL could be used as an aid to certification by generating skeletal safety cases. This would be particularly useful when EBL is the basis for control laws that are automatically generated. As EBL uses a human-readable representation, faulty assumptions and rules can be found and corrected by a human examining the structures generated. In addition, being able to generate or update safety cases for an adaptive controller is a good way to illustrate both the determinism and coverage of the system at a given time. The selection of appropriate domain representation and goals directly impacts the value of the generated safety case.

This chapter advocates maintaining an incremental safety argument for systems that need to be change-tolerant, like in [76]. For non-trivial changes the recertification cost is related to the size and complexity of the system. It would be better if the cost of recertification were related to the size and complexity of the change itself [77]. The impact needs to be determined so that the recertification effort can be limited to only relevant parts of the system. The production of a safety case after a change, for comparison to one from before a change, could be used as an advisory tool

for determining the impact of a change on claims and evidence. This can show equivalence between whole, or parts of, safety arguments.

It is worth noting that any change to the fuzzy rulebase should ideally generate a safety case, be validated using tools, and be manually inspected. This is because of the safety-critical nature of aviation. *"The acceptance of a case is (or should be), in the end, a social process."* [57].

Some limitations of specification-oriented certification are summarised in [46]. These limitations apply to adaptive systems in particular. One problem that can occur when certifying adaptive systems lies in interpreting the model that the system has learned [50]. The approach of generating arguments, as well as fuzzy rules, attempts to explain the internal model of the controller in a manner applicable to certification. The fuzzy rules are generated by discarding information which is not related directly to inputs and outputs. Therefore, it is the explanation structure that contains the additional contextual information which could be useful in analysing not just what the rule does, but why it was derivable.

Additionally, using EBL to generate fuzzy rules only from seen situations limits the fuzzy rule base size. The rulebase therefore reflects the situations which a UAV has been used in, rather than having a single monolithic rule base containing all possible permutations. This gives additional information about what a UAV has previously done and alongside the explanation structures, which also contain additional contextual information, can aid the interpretation of a rule.

Adaptive certification couples strongly with verification. Verification can help to expand the domain theory by discussion of the safety argument. For example, the addition of undeveloped goals to an augmented explanation structure, which come about through safety discussions, could identify areas for further domain theory development.

Examination of the system through simulation is useful as it is a non-destructive process, allowing exposure of the system to a variety of scenarios in a safe environment, albeit in a closed-world manner. Simulation is also a useful tool in training and guiding the generation of a fuzzy controller in this work. During verification, greater coverage could lead to a more mature safety argument. There are tools to help with demonstrating coverage [54].

It may be possible to certify rules post generation, through simulation and validation. However, the technique proposed in this chapter attempts to provide additional assurance with regards to what a generated rule's behaviour is. This is achieved via the generation of safety artefacts from explanation structures. However, since the information required to generate these structures is embedded in the domain theory it is important to note that the certification of the domain theory remains an issue.

It is worth considering further the validation of generated safety cases. Past arguments could be stored as baselines. Additionally, a baseline could be developed which constrains the form of future arguments. Before introducing any new rule the argument could be updated and validated, possibly using the approach in [64], to give some measure of confidence that the rule has no compromising impact on system behaviour. This could automate part of the certification process for an adaptive controller. This approach would probably be too costly for online learning, though it could aid in maintaining offline learning systems.

One concern is that the conclusions drawn by EBL are a direct result of the specifics of the domain theory, which are usually hand-coded expert knowledge. The confidence in the fuzzy rules elicited

by EBL is therefore dependent on the confidence engendered by the knowledge engineering phase, which produces the domain theory. Validation of the domain theory is therefore also an important issue.

## 6.3 EBL-GSN Structure Generation

The process for generating automated safety cases is given in this section. This process will be illustrated by considering a simple explanation structure. The structure is shown in Figure 6.3 and considers whether a particular linguistic value is smaller than another.


**Figure 6.3: A simple EBL structure to be converted to an example safety case.**

The structure given in Figure 6.3 does not correspond to a fuzzy rule, as would normally be the case. This example is a simpler case for illustrative reasons. Further examples will be given in Section 6.4 which relate to specific fuzzy rules.

The whole explanation structure is taken as a starting point as it contains the goal and leaf nodes. A subset of the leaf nodes are taken to form a fuzzy rule precedent, while the goal node is used to form the fuzzy antecedent. However, the structure also contains additional information which is useful in explaining the background of the rule.

This information is useful during the development as well as the certification process, since concerns regarding the safety and the nature of the rules to be generated overlap. Discussion of a combined

structure can highlight issues for consideration in both processes.

Two steps are taken to generate the skeletal safety case. The first step is to express the explanation structure in terms of GSN nodes. This is shown in Figure 6.4. The parallel between EBL and GSN is exploited to do this.

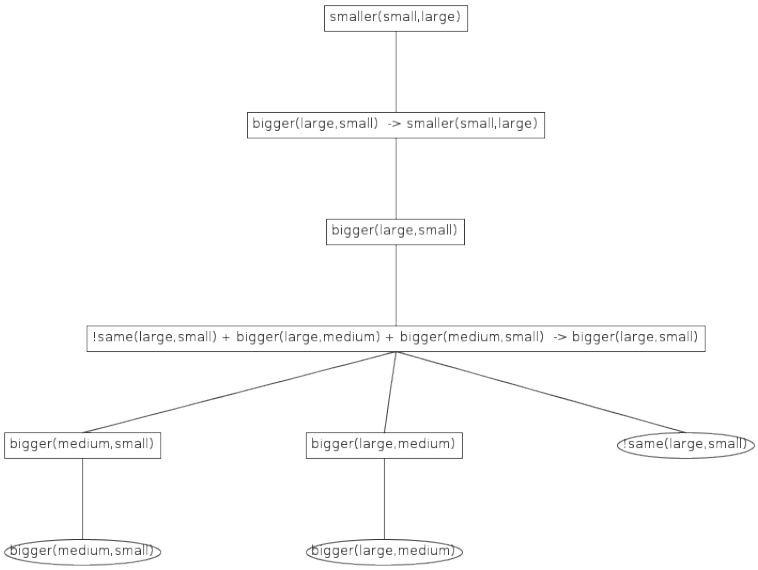


**Figure 6.4: A simple EBL structure expressed using GSN notation.**

The second step is to augment this structure with nodes from a safety-specific domain theory. Each node in the explanation structure is taken as the topmost goal of a new explanation structure. Once this new sub-structure has been generated it is incorporated into the original. The sub-structure is formed from the safety-specific domain theory. Figure 6.5 shows an EBL-GSN strucure. The safety-specific domain theory used is given below in Table 6.2. The domain theory is designed simply to show the process of including each of the nodes from Figure 6.1. The following section applies this process to more complex examples.

| **Domain Theory:** |
|---|
| strategy(this subgoal is specific to the variable binding) $\rightarrow$ bigger(medium,small) |
| goal(The goal compares the variables x=,X, and y=,Y,) $\rightarrow$ smaller(X,Y) |
| undeveloped() $\wedge$ context(Other explanations are possible depending on the variable binding of the topmost goal) $\rightarrow$ smaller(X,Y) |
| strategy(this subgoal applies to all instances of the predicate bigger) $\rightarrow$ bigger(X,Y) |
| solution(each subgoal is supported by a solution in this structure) $\rightarrow$ strategy(T) |

**Table 6.2: Sample safety domain theory**

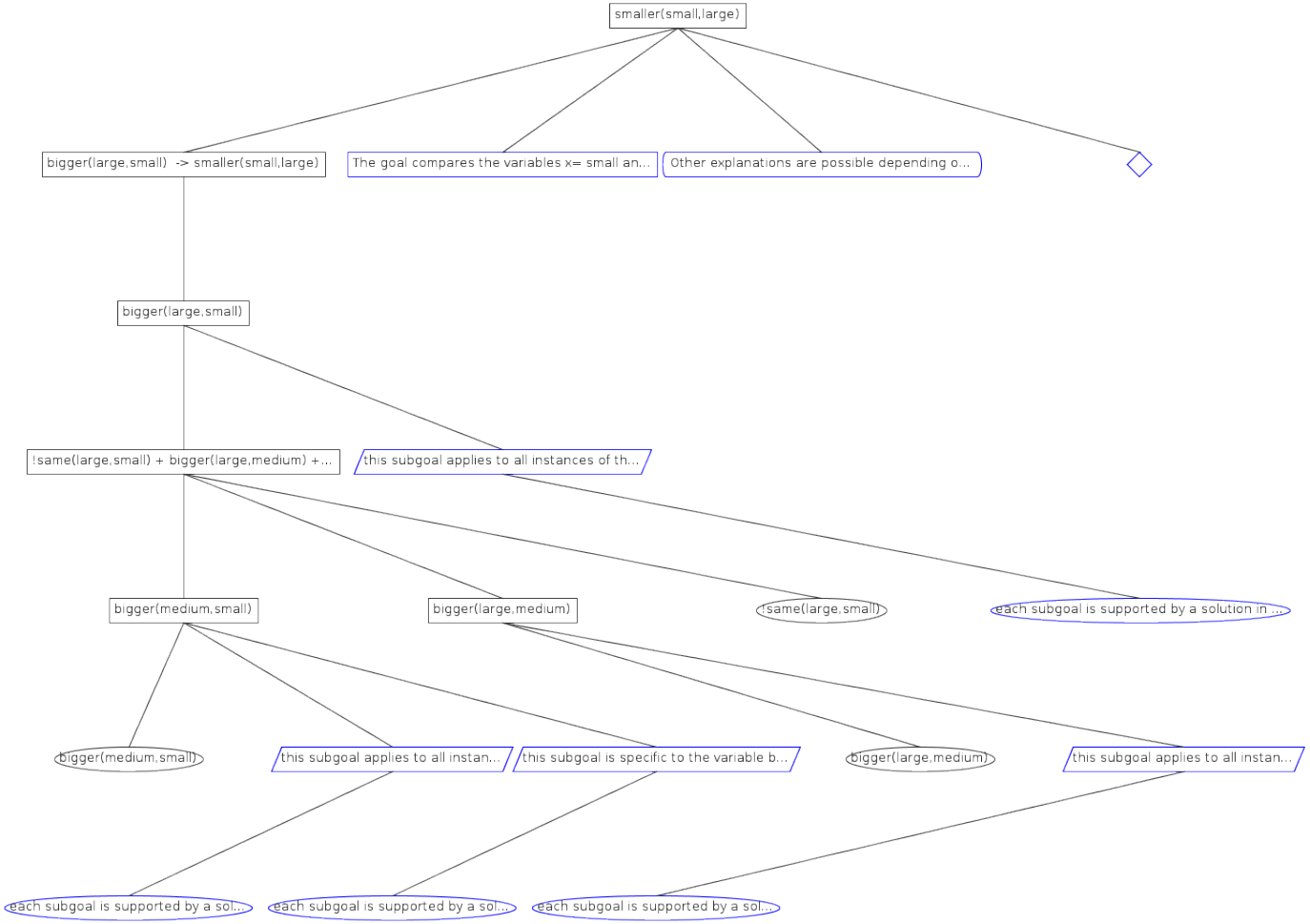


**Figure 6.5: A simple EBL structure annotated to form a sample safety case.**

Figure 6.5 gives an explanation structure which has been augmented with a safety domain theory. It shows that each of the nodes from Figure 6.1 can be incorporated into an EBL explanation structure. In this chapter blue nodes represent those from the safety domain theory. These nodes can also incorporate variables from the explanation structure. Unification can be used to include one node in multiple positions within the structure, where the argument applies in multiple places. If a rule is given in the safety domain theory which unifies with several areas of the explanation structure, then this can be limited by including specific parameters in the head of a rule. A rule in a safety domain theory might have the antecedent *bigger(X,Y)*, which will unify with all examples of nodes using the precedent *bigger*. In addition to this, a rule could have the antecedent *bigger(medium,small)*. This second rule will only be incorporated alongside the first in the event that the variable bindings for the goal from the EBL structure match the substitution: [X=medium, Y=small].

Currently the process shares the same limitations, in terms of forming structures, as the EBL implementation. One such limitation is that of rules having single term antecedents. The technique also shares the expressive power of the EBL implementation. There may still be ideas which cannot be easily expressed using this technique without further implementation. Natural language may not always fit ideally with the explanation structure as it relates to a much larger universe of discourse. However, this work advocates this technique as a starting point to generating safety cases from EBL structures.

The safety domain theory has some differences, in presentation and interpretation, from the domain theory used to construct regular EBL structures. The predicates in this implementation are limited to nodes from Figure 6.1. The visualisation program expresses each of these predicates with the appropriate GSN node. The second difference is that the parameters of each predicate are run together to generate the text for each GSN node. This allows for the inclusion of variables from the original explanation structure. Each of the variables is included as a separate parameter.

There are also some differences between the EBL implementation and the GSN-EBL implementation. The GSN-EBL implementation is responsible for the safety-specific explanations using goals from an EBL structure. No domain theory of statements, inputs, or observations is given as an input to the algorithm. The notion of operationality is dropped when forming the GSN sub-structures. This is because the issues that the notion of operationality was introduced to address are not of concern in this domain. The operationality criterion was introduced to limit the production of rules which had a negative, little, or no gain in overall efficiency. However, these rules still increased the computational overhead of the system. In this case no rules are produced, just structures. It is also paramount that all of the nodes from the safety-specific domain theory be included to express the safety argument.

When a choice point is found a different process is also followed. A choice point is where more than one node can be added to the explanation structure to explain a goal. Normally, nodes after the first are noted as alternative paths which can be tried in the event that the first node leads to a failure of explanation. In terms of a safety domain theory there are no alternative paths, just more aspects to considering a particular goal. For this reason, all unifying nodes are added when they are found. An example of this is shown, where more than one rule is used, when annotating *smaller(small,large)* in Figure 6.5.

The visualisation of the EBL-GSN structures is handled separately from the formation of the structures. There are some differences compared to visualising the EBL structures. The rules,

from the safety-specific domain theory, are themselves omitted from the structure. Precedent nodes are included without the intermediate which links the EBL goal to the GSN node(s). In this work intermediate links are displayed in EBL structures when a rule node has its precedent terms added as separate child nodes. This is not necessarily different from EBL, but it is the case in this implementation that such nodes are included when displaying EBL structures. This would be akin to omitting the node *!same(large,small)* $\wedge$ *bigger(large,medium)* $\wedge$ *bigger(medium,small)* $\rightarrow$ *bigger(large,small)* from Figure 6.5, instead adding *!same(large,small)*, *bigger(large,medium)*, and *bigger(medium,small)* directly underneath *bigger(large,small)*.

The approach has been used to generate some EBL-GSN structures for derivations of fuzzy rules, generated in Chapter 4.

## 6.4 Examples and Discussion

This chapter proposes a method of generating skeletal safety cases in support of EBL generated fuzzy rules. This is to meet one of the objectives of this project; namely, to reduce the recertification workload of the software system engendered by a hardware change.

Rules were generated in the first experiment of Chapter 4. This change in the software system requires an additional certification effort. It would be useful to have skeletal safety cases for the rules that were generated. The generated structures express a safety argument for the software system, focused on the control rules. Some rules from Chapter 4, given in Table 6.3, will be used to form skeletal safety cases.

| **Run** | **Input** | | | **Output** |
|---|---|---|---|---|
| | **Throttle in** | **Energy (f min$^{-1}$)** | **Climb (f)** | **Throttle out** |
| 1 | tiny | none | none | none |
| 6 | full | small | small | huge |
| 23 | full | full | none | huge |

**Table 6.3: Table of the fuzzy rules generated. These rules are to be explained and augmented to form EBL-GSN structures.**

In order to form the EBL-GSN structures a safety-specific domain theory is required. This domain theory is given in Table 6.4.

| **Safety Domain Theory** |
|---|
| context(Does,I, exist as a physical sensor outside of the simulator) $\wedge$ goal(Consider the impact of fuzzification on rule generation) $\wedge$ solution(Link to sensor safety case for,I) $\rightarrow$ reporting(I,X) |
| context(height to go is informed by the flight plan. This is not the standard use for this input) $\rightarrow$ reporting(vvi_dial_fpm,X) |
| strategy(Show that timing does not inhibit a necessary rule being generated) $\wedge$ strategy(Show that the width of fuzzy sets do not prohibit useful input states from being recognised) $\rightarrow$ goal(Consider the impact of fuzzification on rule generation) |

solution(Compare the goals generated over the entire flight envelope with various timing settings) → strategy(Show that timing does not inhibit a necessary rule being generated)
solution(Link to a study on the impact of granularity of fuzzy sets on rule generation) ∧ goal(The system should not change fuzzy set widths after deployment) → strategy(Show that the width of fuzzy sets do not prohibit useful input states from being recognised)
solution(fuzzy sets are chosen through simulation of flight envelope) → goal(The system should not change fuzzy set widths after deployment)
goal(Reduction of throttle does not lead to unsafe performance) → reduce(I,X)
goal(The ability to climb is always maintained) → goal(Reduction of throttle does not lead to unsafe performance)
strategy(Switching mechanism competes with reduction rules) → goal(The ability to climb is always maintained)
solution(Comparison over flight envelope in study on the impact of different switch settings autopilot/controller competition) ∧ undeveloped() ∧ context(Competition is also possible via an increase strategy so as to also be informed by the flight plan) → strategy(Switching mechanism competes with reduction rules)
goal(Selection of inputs for determining reduction is appropriate) → intend(I,X)
solution(Selected inputs mirror approximation of gradient) ∧ undeveloped() → goal(Selection of inputs for determining reduction is appropriate)
goal(Use of next fuzzy set down as an appropriate output) → one_smaller(I,X)
solution(Link to a study on the impact of granularity of fuzzy sets on rule execution) → goal(Use of next fuzzy set down as an appropriate output)

**Table 6.4: The safety-specific domain theory used to produce skeletal safety cases.**

The process was applied to the rules in Table 6.3. The original explanation structures are comprised of the nodes outlined in black, while the blue nodes are those from the safety domain theory. One obvious aspect of the safety domain theory is that it tends towards a greater verbosity. This could make the management and display of such structures problematic. For this reason, the values of the GSN nodes are abstracted into Table 6.5. The EBL-GSN structures that were produced are shown in Figures 6.6, 6.7, and 6.8.

| **EBL-GSN Key** |
|---|
| **Goal:** |
| (1) Reduction of throttle does not lead to unsafe performance |
| (2) The ability to climb is always maintained |
| (4) Selection of inputs for determining reduction is appropriate |
| (6) Consider the impact of fuzzification on rule generation |
| (17) Use of next fuzzy set down as an appropriate output |
| (25) The system should not change fuzzy set widths after deployment |
| **Solution:** |
| (5) Link to sensor safety case for throttle_ratio_all |
| (10) Selected inputs mirror approximation of gradient |
| (14) Comparison over flight envelope in study on the impact of different switch settings autopilot/controller competition |

| |
|---|
| (18) Link to sensor safety case for total_energy_fpm |
| (21) Link to sensor safety case for vvi_dial_fpm |
| (26) Link to a study on the impact of granularity of fuzzy sets on rule generation |
| (27) Compare the goals generated over the entire flight envelope with various timing settings |
| (28) Link to a study on the impact of granularity of fuzzy sets on rule execution |
| (34) Fuzzy sets are chosen through simulation of flight envelope |
| **Context:** |
| (7) Does throttle_ratio_all exist as a physical sensor outside of the simulator |
| (13) Competition is also possible via an increase strategy so as to also be informed by the flight plan |
| (20) Does total_energy_fpm exist as a physical sensor outside of the simulator |
| (23) Does vvi_dial_fpm exist as a physical sensor outside of the simulator |
| (24) This is not the standard use for this input |
| **Strategy:** |
| (8) Switching mechanism competes with reduction rules |
| (11) Show that the width of fuzzy sets do not prohibit useful input states from being recognised |
| (12) Show that timing does not inhibit a necessary rule being generated |

**Table 6.5: The key which abstracts the text out of the GSN nodes in the example EBL-GSN structures.**

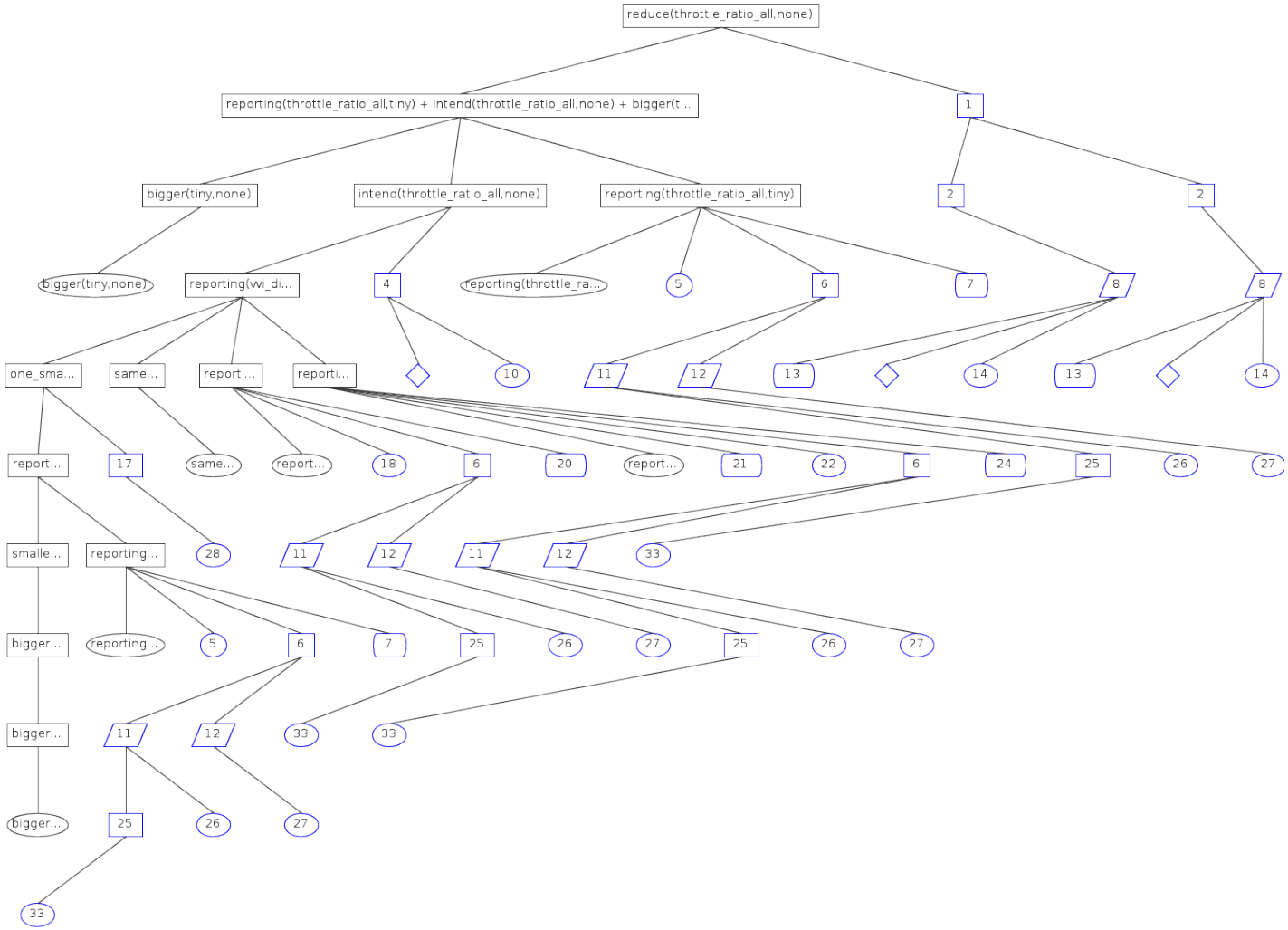


**Figure 6.6: An EBL-GSN structure representing the rule derivation from run 1.**

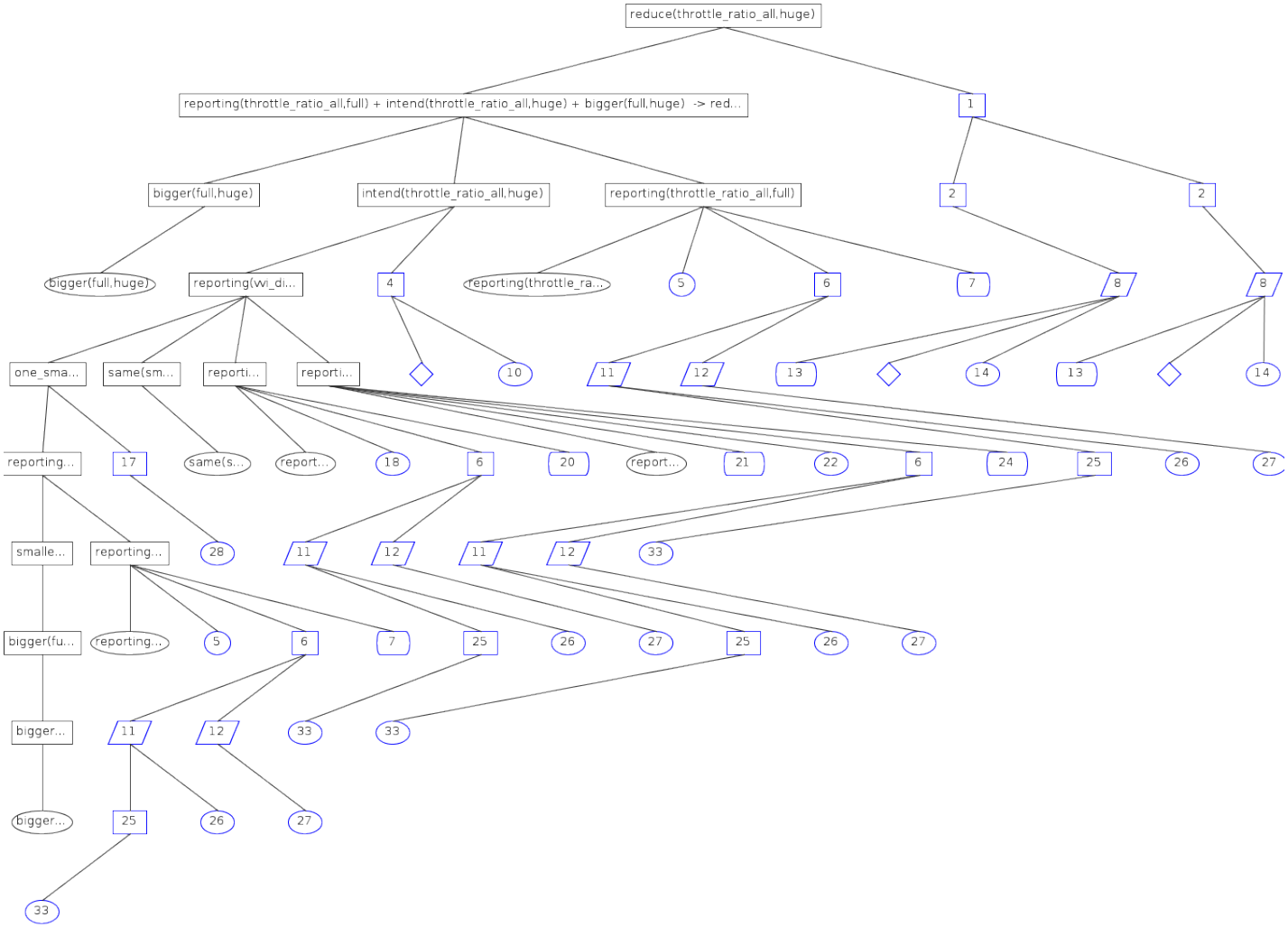


**Figure 6.7: An EBL-GSN structure representing the rule derivation from run 6.**

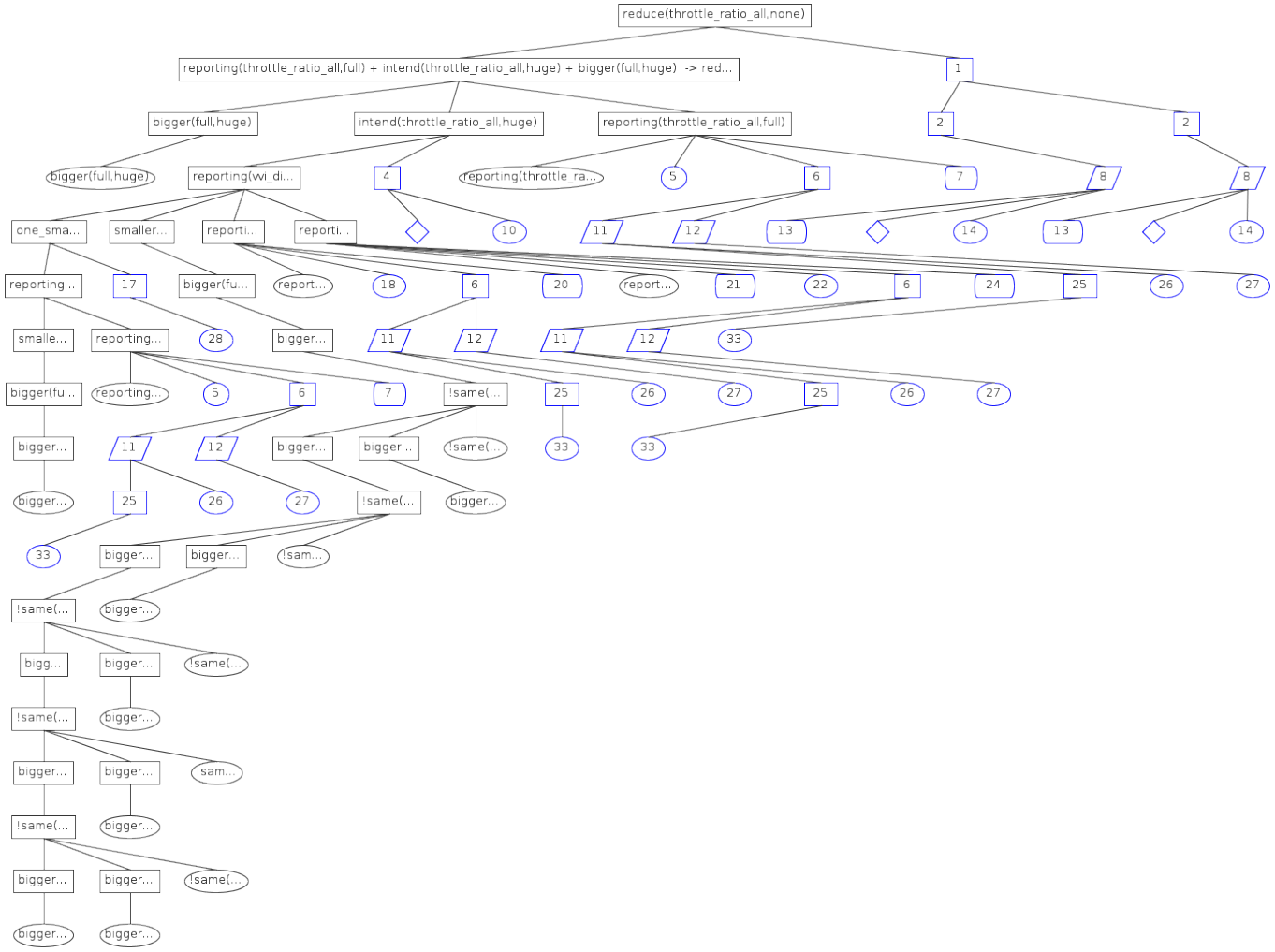


**Figure 6.8: An EBL-GSN structure representing the rule derivation from run 23.**

As can be seen, there is a great deal of similarity between Figures 6.6, 6.7, and 6.8. The structures can differ in several ways. In Figure 6.8, the node *smaller(none,full)* differs from the other two figures which use the *same* predicate in this position. The deviation occurs because the domain theory has two rules which imply the *intend* goal. A more complex domain theory would produce more varied diagrams. This difference could be harnessed in the safety domain theory to elicit a different structure where there are concerns specific to each strategy. Another possibility is to include more instances of nodes with specific variable bindings to only augment some structures. These are both points to consider when contrasting multiple structures during the design process. Both approaches seek to make more specific structures for each rule.

One limitation of the technique is that some nodes may appear more times than is necessary as they will be unified with each possible node. It may be the case that only the topmost instance of *bigger(X, Y)* requires a specific annotation. One possibility is to use an intermediate explanation pass, to annotate the original EBL structure with additional contextual nodes. These additional nodes can then be explained further. An example of a context node might be a node to count the number of occurrences of each particular term in a structure. Alternatively, an additional explanation could occur separately from the process to add statements into one of the domain theories, which would have the same effect. The effect is that additional nodes are added to specific positions to make previously repeated nodes into unique examples. These specific examples can then be referred to in the safety-specific domain theory to prevent repetition of safety-specific nodes. These possibilities apply to situations where variable binding will not give the required uniqueness.

There is one main advantage to this process, apart from in reducing certification effort. This is that, in order to design the safety domain, theory various stakeholders need to be brought together during the design process. This will give both domain theories a more varied scrutiny. This could help to identify development issues early for both processes. Issues which are identified earlier in the software development process require fewer resources to correct.

Further research would be useful in extending this process, via intermediate annotation and the inclusion of more types of GSN node. The extended process should be applied to more realistic case studies in order to ascertain the work that needs to be done in order to have the process accepted into the aviation industry. As presented, the technique has the potential to generate safety artefacts to express part of the safety argument. Changes to the system result in updated structures. Whilst these structures may not form complete safety cases, they reduce the work required by the dynamic interleaving of one domain theory with another, using specific observations as a bias. In this way the objective for this chapter has been addressed.

## 6.5 Conclusion

A process for generating partial safety cases from explanation structures was presented. Several examples were generated for existing fuzzy rules. An argument for adaptive safety cases expressing part or all of a safety argument has also been presented alongside the consideration of adopting EBL within the aviation industry. The method discussed in this chapter is proposed as a way to reduce the certification effort engendered when a change in hardware requires a change in control rules. The automation of safety case generation, allowing for a contrast of the system before and after a change, is how this objective is satisfied. The examples presented are dependent on the

domain theories and represent a larger range of possibility than was demonstrated.

The ability for a system to explain its actions in relation to safety concerns could be useful in the context of aviation certification. EBL could be used as an aid towards automatically maintaining safety arguments, especially where the conclusions or rules that require justification were reached by EBL initially. This representation could also be useful in identifying faulty logic, thus aiding interpretation of the internal model. This advantage is realised by the inclusion of extra information that went into the formation of a given rule. Extra information is information in excess of what forms the rule itself. Additionally, these safety cases can be used to demonstrate coverage and determinism, both key properties of safety arguments. Another advantage to this process is in bringing more stakeholders together early in the design process to consider certification. This is a result of the need to design a safety-specific domain theory which interacts with the EBL domain theory. By facilitating this interaction, issues can be identified earlier, which aids the development process.

This chapter has demonstrated an extension to the MPMS capable of generating safety case fragments. These fragments are centered around the derivation of a given fuzzy rule. Each fragment is intended to support the consideration of the safety aspects of an individual generated rule, early in development. The MPMS can augment rule derivations with the core GSN nodes, utilising the expressive power of unification. This chapter fulfils the third objective of this work. In fulfilling all three objectives, both the software and certification efforts for modular management are reduced. The aim of this project has been satisfied. However, this work has both elicited some interesting angles of future work and may have wider applications which should be considered.

Further work will involve applying this approach to a more realistic case study. The process should also be augmented to allow for a greater range of GSN nodes to be included. Both of these could facilitate a more detailed evaluation of the utility of the approach. It would also be useful to ascertain if there are any aspects of a safety argument that cannot be represented by this approach. Additionally, the process could be expanded to include the consideration of meta-data about the original EBL structure to be included in the safety domain theory. This allows a reduction in repetition of safety-specific nodes. Furthermore, the collection and integration of data collected in simulation could be achieved to facilitate further automation. It is also possible to apply EBL to the examination of control laws embedded or generated by different techniques. This approach could be expanded for use with any tree-structure producing technique. SVMs might be an interesting technique, when integrated with EBL as in other works, to consider alongside the automatic generation of safety cases.

# Chapter 7

# Conclusions and Future Work

## 7.1 Conclusions

There is interest in changing the hardware that comprises a UAS between uses. This is in addition to the changes which occur over the lifetime of any aviation platform, such as upgrades and degradation. When hardware changes it may be the case that the software which manages the platform also needs to change. This is the case when the change alters the nature of the system resulting in a different ideal behaviour. Additionally, hardware platforms are certified in aviation. Changes to a platform require that the platform is recertified, unless that particular combination has already been certified. These two sources of work engendered by a change in hardware are the motivation of this project. Therefore, the aim of the project is to: *reduce the software and certification efforts engendered by the management of a modular platform.* Three objectives were proposed which relate to the techniques proposed to address the aim.

The aim is concerned with reducing the workload of an MPMS in two main areas. Firstly, there is the workload involved with expanding a software system to incorporate new hardware. Secondly, there is the certification effort required to support the software alterations arising in the first area.

In order to reduce the software workload two problems must be addressed. Firstly, the control software must be able to adapt to changes in known hardware. Such changes may be incurred by hardware upgrades, as well as the degradation in hardware performance over time. Hardware upgrades in this sense involve changing one item of hardware for another that performs the same task in the same way. However, a change in hardware may require a different set of control rules. The scope of these changes may exceed a single item of hardware. Secondly, the control system should be able to adapt to the introduction of new hardware. New hardware in this case refers to hardware which cannot be catered for with the existing control system explicitly.

Generalisation is proposed to address the first problem; specifically, the use of general strategies to generate hardware-specific rules. By generating the specific control rules, the effort of effecting a software change can be largely automated. The automation of software changes is intended to partially fulfil the aim of the project. This is the motivation for the first objective.

Transfer learning is proposed to address the second problem. Transfer learning could allow the application of a general strategy to a new hardware item. In this way, the software can use transfer

learning to extend itself to control hardware which was not originally covered by the controller. This is the motivation for the second objective.

In order to reduce the certification workload a problem must be addressed. When the software system adapts, a new safety case must be presented.

This workload can be alleviated by attempting to automate the generation of safety cases. The generation of safety case fragments from rule derivations is proposed. Augmenting rule derivations with additional, safety-specific, information can provide for automation. This is the motivation for the third objective.

The first objective is: *To investigate the integration of EBL and fuzzy controllers.* The two techniques have been integrated. EBL was proposed to generate fuzzy rules which are executed by a fuzzy controller. Inputs are fuzzified, with EBL and the fuzzy controller both interpreting the inputs, as linguistic values, in different ways. This, along with some caveats, facilitated the communication between the techniques. This objective was proposed to investigate an approach to automate the generation of specific rules for a platform given a general strategy. This is one way to reduce the workload of changing a software controller when its hardware platform changes. The kinds of change that are mitigated by this approach are changes where the hardware changed shares a strategy for control. The technique is tolerant to changes in specifics.

The second objective is: *To investigate the extension of EBL to perform transfer learning.* The inclusion of analogy into EBL was proposed. Upon reaching in impasse where no deductive solution is possible it may be useful to employ analogy instead of unification. Analogies are supported by deriving a new concept formed using the similarities between the concept which causes an impasse and one which shares parameters with it. This process may be likened to strategically generalising the predicate rather than the parameters of a term. This allows for the integrated EBL-fuzzy system to be expanded to apply to changes in hardware which have no applicable deductive strategy, by using other strategies to control hardware otherwise outside the deductive closure of the system. These first two objectives are intended to address the workload relating to the software system, referenced in the aim.

The third objective is: *To investigate the extension of EBL to produce certification artefacts.* Explanation structures are used to generate fuzzy rules and each fuzzy rule is generated from one structure. The structures contain information in excess of that which is required to form the fuzzy rule. This additional information, contained within the domain theory, is augmented with further explanation using a safety-specific domain theory. These composite structures are the basis of a safety case. By generating these structures, the certification workload for the fuzzy controller is reduced.

By completing the objectives of this project the aim has been satisfied. The MPMS is comprised of the components developed for each of the three objectives. The MPMS itself was proposed to address a dearth in the literature; In aviation control, the work on IMA has so far concentrated on the IMA architecture and not the impact of modularity on the software operating within it. The MPMS is one such piece of software and reduces the workload caused by a change in the hardware being managed. However, the MPMS need not be concerned with only power management or aviation but can apply more generally to modular control. Additionally, modular systems within domains which require certification could benefit from this work.

Another area which the MPMS could relate to is the management and control of modular reconfig-

urable robots. This is an example of a domain where part of the MPMS could be relevant, since it is a domain where certification may not be required. The selection of EBL and Fuzzy control was motivated by the desire to utilise human-readable techniques, which certification benefits from. Without this constraint, other techniques could be integrated. ANNs are a good example, as they are a black-box system and would present additional barriers to certification, but could potentially bring *tabula rasa* learning to EBL. Extending EBL in this manner is another interesting area of literature that the MPMS could impact after further work.

Without the constraint of controlling a platform with very limited resources, a UAS, a tighter coupling between techniques could be considered. This could further the impact of the MPMS by including more domains into a system following the same concept. The shared concept is the aim of this project.

An existing limitation of the MPMS is the need to train controllers. This requires time and high quality simulation. The simulations need to be high quality as the system becomes specialised by its inputs and therefore its fidelity is reliant on the input data provided.

The MPMS as a system requires further work. The most significant limitation is that it is not clear how to pursue the certification of a rule based on analogy. Further work is required to extend the system to be adept at including evidence into safety cases in a manner which would support both deductive and analogical rule derivations. Furthermore, work would be required to adopt this system into industry. One matter that would need to be addressed before industrial application is the certification of the MPMS itself. Adaptive safety cases could help to form a safety argument for the generated rules, but the controller and rule generator would require their own justification. Currently, the MPMS is limited to domains where certification can be obtained via safety cases.

The work for each objective is considered further. Each area may have a wider impact than was discussed within the scope of this project.

#### EBL and Fuzzy Control

An approach for integrating EBL and fuzzy control was presented in Chapter 3. By integrating these techniques EBL can reason about inputs by employing a set of linguistic values. The fuzzy controller can have specialised rules generated from a general strategy by EBL. These rules can benefit from the comparatively rapid execution inherent to fuzzy controllers.

Communication from EBL to the fuzzy controller was considered. Explanation structures are taken and converted into fuzzy rules. The goal of the structure needs to represent an output and the leaf nodes of the structure need to describe a state. Any nodes which are not relevant to these two aspects are discarded. The remaining nodes are converted into a fuzzy rule. The underlying implication is that the leaf nodes can be taken to imply the goal. This implication is already employed in EBL as the term chunking. Chunking is part of the generalisation process, along with undoing variable bindings.

The domain theory EBL uses contains general control strategies. These strategies are specialised to form fuzzy rules. Specialisation is driven by the inputs, which are fuzzified and communicated to EBL. The fuzzy rules are executed by the fuzzy controller. By employing general strategies, rules can be generated for a variety of situations and hardware configurations without additional work. This approach is successful in preventing additional work where hardware changes require

a different specialisation of a particular strategy. An example of this is that, when controlling a particular item of hardware, the system changes. The changes are made to other hardware items, but the change in platform behaviour requires a different set of control rules for the given item.

Using a general strategy to generate specific rules has some implications. Firstly, the permutations are driven by a bias. Training data presented to the controller determines what rules will be generated. This process can be automated to reduce the workload required, but there is still a training overhead. Secondly, whilst it may be useful to analyse each rule when evaluating a controller, much can be discovered by considering the strategy which leads to a group of rules. This allows the consideration of rules alongside additional information relating to their conception, which can aid in understanding what has been generated.

Communication from the fuzzy controller to EBL was considered. This is realised by converting inputs into a form which can be interpreted differently by each controller. This takes the form of fuzzifying inputs and marshalling them into a form which includes a predicate, an identifier for the input, and the linguistic value. This marshalled input is interpreted using predicate logic by EBL. Membership is abstracted out of this form and the state of each input is considered in abstract. When these values are included in the fuzzy rules generated by EBL, membership is considered during the defuzzification process. In this way the power of fuzzy control is not diluted, despite being abstracted from EBL.

Another aspect of the EBL-fuzzy integration proposed is that fuzzy sets are interpolated. This allows the same linguistic variable to be applied to every input, yet have a different meaning. Each distribution of linguistic values is scaled across the values which have been previously encountered for that input. This allows for certain hardware changes to be automatically accounted for as the sets scale. Scaling the fuzzy sets avoids the need for additional work when certain changes in hardware occur. Examples of this include the degradation of hardware over time, or upgrading a particular item of hardware such that the input values change but the control strategy remains the same.

The interpolation of a single linguistic variable over multiple inputs may impact the wider literature. In conception this allows each linguistic term to be contextual. A bucket being *full* and a lake being *full* are described using the same language but refer to very different volumes of water. Having only a few terms, contextualised to each input, allows EBL to reason about orders of magnitude with only a few terms. Additionally, when rules referencing these values are generated they will be interpreted appropriately to the input.

Chapter 4 presents an experiment for validating the integration of EBL with a fuzzy controller. The energy requirements of a given flight are recorded. When the EBL-fuzzy controller is employed an energy saving is noted; this corresponds to the assertion that the approach can produce rules which reflect the intent of the domain theory. Additionally, this experiment is conducted on a second platform to consider whether the approach is applicable to modular control. Control of a different platform is considered as an extreme case of modularity. The throttle is controlled on both platforms, to reduce energy consumption. A different set of rules were generated for each platform, both resulting in a reduction in energy expenditure over the flight.

The experiments are concerned with providing a proof of concept. More detailed experiments could be conducted to establish more specific conclusions. However, the behaviour of the controller is dependent upon the knowledge engineering, in the form of the domain theory. A different do-

main theory can produce very different results. This merits consideration when designing further experiments. Additionally, the fidelity of the simulator in relation to energy management is limited. Further experiments would benefit from hardware test-beds or matlab-integrated X-Plane solutions.

The experimental results provide a proof of concept for the integration of EBL and fuzzy controllers. This approach could apply to many other areas. An example is the games industry, where an entity might have a single strategy which can be specialised to many situations, rather than developing a monolithic rule base or decision tree. The commonality which would make this an interesting additional domain is the need for fast reactions while desiring a simple development process. The entities in games could be allowed to learn online and develop faster reactions to seeing the same situation more than once.

The experiment has some limitations. Firstly, the goal selection process is competitive in nature which could lead to a bias in the rules developed as well as variation regarding when they are first derived. This had a limited impact on the experiment as only the *reduce* and *state* goals were in competition and these were threaded. Further experimentation may need to consider this issue. There was also variation in the data received for each flight; a sample size of 50 was used in order to limit the variations given by the simulator. These can be identified by comparison to the control group. The control group are the experiments conducted without EBL control. Due to the variance only general trends are elicited from this experiment. However, the effects of including the controller are pronounced. A final limitation of the experimental work is the lack of contrast with other techniques. The existing industry implementation is absent from the simulator and AI techniques are limited to non-flight critical applications currently. A more speculative study could contrast this technique with others. However, in flight fuzzy controllers are used and these are a mature area of study. Additionally, since the specific rules generated are expert knowledge it is this which would be tested if the technique were contrasted with ANNs.

The results show that the integration of EBL and fuzzy control can be used to automate software changes. These software changes are a reaction to an update or degradation of existing hardware. Investigation of EBL and fuzzy control in these terms satisfies the first objective and goes some way to fulfilling the first part of the project aim.

This research could be of wider interest, in domains other than aviation control. It remains to propose some further areas for study.

### EBL and Analogy

Chapter 5 presented a technique for incorporating analogical reasoning into EBL. When reaching an impasse it may be possible to replace the predicate by forming an analogy. The predicate causing an impasse is selected as the source. A target predicate is searched for, using parameter values as a bias. The definitions of both source and target are used to form an abstraction which both subscribe to. This is achieved by forming the abstraction from commonalities between the source and target definitions. Abstractions act as supporting evidence for swapping the source predicate for the target predicate. This allows explanation to continue.

An experiment was conducted to consider whether useful rules can be generated by analogy. These experiments were the same as in Chapter 4 except that no deductive rules could be generated.

The item of hardware being controlled was not given a strategy and instead permutes the throttle reduction strategy. Controlling an item of hardware which lacks rules within the domain theory is akin to encountering a new item of hardware for which the domain theory takes no account. The intent of including analogy into EBL was to further EBL's ability to generate fuzzy rules for other examples of modularity than those considered in Chapter 3.

Rules were generated which positively impacted the energy consumption of the hardware platform. This supports the possibility of extending EBL to include analogy. However, rules were also generated which had a negative impact on energy consumption.

Rules generated by deduction are as suitable as the domain theory which is used to generate them. However, analogy does not share this property. Therefore, rules generated by analogy require justification and may not be applicable. This is the reason why analogy is only used at an impasse where no deductive option is available. Additionally, the validity of a rule generated by analogy may not be decidable *a priori*. It seems likely that rules require evidence to support their adoption in addition to an analogy. This evidence may need to be gathered by using such a rule.

The autopilot attempts to employ an over-provision of thrust and climb more steeply than was necessary. The competition between the rules generated and the autopilot was also considered by altering the switch sampling rates. Different rates had different impacts on the energy consumption observed. This may imply that the rules generated may be applicable, or not, based on their interactions with other opposing influences.

EBL has previously been considered alongside analogy in order to form a more complete domain theory from two incomplete domain theories. This work proposes a different outlook, as well as a focus on supporting the generalisation of predicates. The outlook is to assume that the domain theory is incomplete but may hint at deeper relations than are explicitly stated. The extension of EBL to continue explanation when reaching an impasse is part of a broader literature. This work may highlight some directions for impasse resolution in general. Perhaps impasses can be used to highlight areas where a technique should become more explorative as they signal a gap in the domain theory.

Another area where this work could find value is in considering the value of mixing modes of learning. By demonstrating the ability of analogy to generate useful rules, the applicability of EBL can be extended. However, emergent behaviours as a result of this have not featured in this work.

The ability to generate some useful rules using analogy has been shown. These rules are generated for hardware which was previously outside of the scope of the system. This has been achieved using transfer learning. By extending the rule generation mechanism in this way the technique has been shown to be applicable to modular control. Investigation of this issue satisfies the second objective. By satisfying the first two objectives the project aim is fulfilled in terms of reducing the relevant software alteration workload.

Some areas for further study were elicited in this work and are discussed in more detail in Section 7.2.

**Adaptive Safety Arguments**

A method for augmenting explanation structures to generate skeletal safety cases was presented in Chapter 6. Explanation structures are used as a basis as they relate to a particular rule derivation but include additional information. This additional information can be useful in understanding the rule and the constraints present during rule derivation. Automation of this process could help in maintaining an adaptive safety argument for the adaptive elements of a controller. This adaptive argument could fit within a more traditional safety argument. This was motivated by the desire to reduce the workload in re-certifying a system after a hardware change causes a change in control software.

The comparison of an argument generated before and after a hardware change could help in identifying the impact of such a change, which is an issue with re-certification. This technique could apply to other domains, but is limited to techniques which use a tree structure. Additionally, effort is required to generate a safety-specific domain theory which is used to augment the tree structures. However, this effort may encourage bringing together safety and knowledge engineers early in the development process, which could prove beneficial.

This approach could be used, by the strategic selection of goals, to examine the domain theory entire. This would be useful as deductive rules are a result of a given domain theory. An argument for the domain theory, separate from the specific rules derived, may prove useful. In this case, any technique which increases the deductive closure of the domain theory could be more clearly examined.

The approach considers the functionality of the software by expanding each rule and including information relating to its generation. This makes design decisions made by the automatic generator explicit. Linking rules and the assumptions that lead to their generation explicitly to software safety requirements can aid in validation.

The approach could be applied to other industries as well as techniques. The techniques need to be tree-based, such as decision trees or theorem provers. Currently this approach is only applicable to domains where GSN is an accepted method of presenting a safety case.

Three rule derivations from Chapter 4 were used as the basis of safety cases. These were used to demonstrate the ability of the approach to incorporate various GSN nodes. Additionally, the ability to leverage unification to replicate portions of arguments was shown. Some ways for reducing replication, when desired, were also discussed.

Two limitations are apparent with the approach to date. Firstly, the structures generated require manual review. This is likely to remain true for safety argumentation. Secondly, there are no specific tools or precautions for deriving a safety case for analogically derived rules. These rules are likely to be under greater scrutiny as they are not a result of deduction.

Safety case fragments were generated for several fuzzy rule derivations. The expressive power of this technique has been considered. The possibility of the extended MPMS to generate certification artefacts has been demonstrated. By supporting the automation of safety case fragments the third objective has been fulfilled. The project aim has been satisfied by fulfilling all three objectives.

Section 7.2 further explores the areas of future work that resulted from the augmentation of EBL structures to form safety cases.

## 7.2 Suggestions for Future Work

Further work is called for in several areas relating to this work. Firstly, the project as a whole is considered, followed by the consideration of the individual works proposed to fulfil the project objectives.

The MPMS is concerned with behaviour which impacts power usage. However, the technique is general and could apply to the management of any area for which a domain theory can be developed. Adequate inputs and outputs also need to be present. Due to limitations with the simulator the control strategies border on flight control. An autopilot could be developed using the MPMS architecture. One concern with this is that if online learning is disabled then the autopilot would require training for every situation within the flight envelope. This suggests a second line of enquiry which should be part of such a project: to explore the possibility of certifying the possible rules which can be generated, by consideration of the domain theory strategies without specialisation. Flight control is not the only possible alternate aviation application using the MPMS as a basis, but does border the existing work closely.

In addition to utilising this work to control different systems it may be useful to consider further the interaction between an MPMS-based system and a more integrated level of control. An example would be to integrate the MPMS with the FMS and mission planner more tightly so that the specifics of a flight can be used to tailor behaviour. Value could also be found in examining any of these examples on a hardware platform, rather than in simulation.

Different industries might also benefit from an MPMS-based system. Given the possibility of reducing recertification effort, this work might successfully be applied to other industries which require certification. Some interesting possibilities are the automotive, rail, and nuclear industries. Similar benefits could be found in application to these domains, though modularity may be less of a concern.

Alternate application domains can be found by considering areas which consider modular control. An interesting example is the control of modular reconfigurable robots. In this work there were a minimum of two platforms, one with limited computational resources, another with laxer restrictions. The more expensive task of rule derivation was therefore separated from rule execution. With distributed robots it may be more useful to fragment the domain theory and rule derivation tasks as well as allowing each unit to execute rules. Consider a group of robots forming into a structure to enable locomotion. Different areas would require different rules, need different knowledge and have different constraints. It would be interesting to consider the splitting of both knowledge and derivation tasks. One possible line of enquiry is the sensible delegation of explaining sub-goals within a rule derivation to other modular units. This could obscure the issue of identifying gaps in the domain theory by having other possible explanations for an impasse. These include communications issues between units and the delegation of explanation to modules which currently do not require the knowledge to explain a given goal. Some areas of the overall structure of robots may not be required to execute rules often, which could also help in distributing explanation tasks.

The project as a whole deals with modular control. One aspect of this is in adapting existing strategies to work with new platforms. But it may be that a strategy is required but is absent from the domain theory. It would be a useful line of enquiry to consider integrating EBL with a technique which could allow for *tabula rasa* learning from observed data. ANNs might be one such

technique. One concern is that new ideas would need to be put in a form which EBL can relate to. If this is to remain human readable then either the internal representation of predicates/concepts could be separated from what is displayed to people. A possibility here is the manual annotation or renaming of a predicate to a human recognisable form, which would make the system specific to a particular human language. Rather than separate the internal and external languages of the system, the predicates could employ language synthesis to name new ideas as they are categorised, in a manner which fits with human language. Either approach would be an interesting area of further work.

The inclusion of a technique which deals with the categorisation of raw data into concepts and relations, proposed above, fits into another complimentary area for further research. It is worth pursuing an analysis of such further work as an example of mixing modes of learning and the overall impacts of such designs.

Further to these considerations, other areas of interesting further work come from each of the approaches proposed to fulfil the project objectives. These are now explored further.

### EBL and Fuzzy Control

Several areas for further research became apparent when conducting the experiments relating to the integration of EBL and fuzzy control.

Goal selection was achieved using the selection of a single goal and exploiting a hierarchically designed domain theory. However, in more complex applications this may not suffice. Further work into goal selection would be advantageous. One possibility is to have agents select goals for explanation periodically. Each agent could have a pool of goals which relate to its particular area of concern, for example throttle control. Another possibility could be to have the outputs of a neural network giving a different goal for explanation and have the system inputs as ANN inputs. This would be like learning the situations when each given goal is an appropriate concern.

The experiments conducted in Chapter 4 do not provide clear results to show whether the technique is monotonic. That is, whether learning a new rule always has a positive impact. It would be useful to establish this aspect of the system. Analysis of the individual impact of rules generated could prove useful, if the technique is non-monotonic, in order to discriminate between rules to keep. However, the utility of a rule may be affected by factors other than the rule itself, such as the timing of the execution of the rule or the combined effect of the rule with another.

The fuzzy set distributions are set and then interpolated across the range of input values encountered. The distributions, both the number and shape of the individual linguistic values, could be made into a learning task so as to more optimally apply to each given domain. This would be another interesting area for future work. One aspect to consider is whether some form of information about the relationship of an input to 0 should be included in fuzzification. This can be useful information but is absent from this implementation. The current fuzzy distribution is linear. A non-linear distribution may be more applicable to aviation control. However, this would make the system less transparent and may require extra consideration in terms of certification, which could also be a fruitful area of future study.

When comparing two inputs, it may be useful to consider different linguistic values. Currently, the fuzzification process accounts for previous values which relate to one input. When comparing

two inputs it may be useful to re-fuzzify the crisp values using a different process. It may also prove useful to consider each crisp value in relation to the historical values of both inputs. Re-fuzzificaiton may not be necessary; an alternative might be to include meta-data into fuzzified values or to include such meta-data into another linguistic variable.

Interpolation of the fuzzy sets potentially accommodates for degradations in hardware over time. This particular aspect is worthy of further experimentation. There may also be times when the interpolation of the fuzzy sets, which determine when a rule triggers and what its effect will be, may avoid rule generation. This is another area of research which could be performed with the existing implementation.

Finally, this approach could be applied to different domains and analysed in more detail. Application of the technique to hardware, rather than in simulation, could also be beneficial.

### EBL and Analogy

The main area of future work that became apparent in Chapter 5 was the need for a way to evaluate the utility of rules. Rules generated by analogy are not always beneficial. It appears that it may not be possible to judge whether a rule will be useful at the time of generation; more information is needed. It would be useful to devise a universal way of judging the utility of rules and incorporating this into the rule generation process.

One possibility is to use data from the application of a generated rule to annotate the supporting analogy. This could prevent the generation of other rules with low or negative utility. Additionally, this could add some *tabula rasa* learning into EBL. It would be particularly beneficial to elicit more from the underlying correlation than just a derived, more general, concept. Analogy holds only in certain circumstances, and the over-general application of any particular analogy will not always prove fruitful. For this reason, formation of analogy may be considered the start of an ongoing learning task.

Analogically generated rules need to be considered with regards to the certification process. Given that they are derived by analogy, rather than deduction, it may be that less confidence can be held in each derived rule. More scrutiny may be required to allow such rules to be used industrially. Eliciting a set of guidelines required for non-deductively derived rules would be a fruitful goal for further research.

The current implication could be further examined in a few ways. Firstly, the domain theory was constructed to allow only certain analogies to be generated. It would be useful to consider the impacts to knowledge engineering when starting with domain theories not designed for the purpose of analogical reasoning. Application to a more complex domain theory could also help to highlight the dangers of the overuse, or possibly underuse, of analogy as proposed in this work.

Finally, when there are multiple possible analogies to choose to employ it would be beneficial to have a better method of discrimination. Currently, the most complex analogy is favoured. However, it may be that there is an opportunity to learn more about the non-explicit relationships which underpin the domain theory by further investigation.

### Adaptive Safety Arguments

The process of generating adaptive safety cases should be considered further. Currently, the process has not been evaluated in relation to a larger argument, which may be standards-based. The integration between the two approaches should be further considered.

The approach should be considered with a more complex example. This could help investigate whether all the required aspects of GSN can be generated, or if more work has to be done to preserve the expressive power of the GSN representation.

When changes occur it would be useful to generate an analysis of the impact of a change. This can use the state of the controller before and after a rule change as well as safety cases generated before and after. Change analysis would prove to be a beneficial further work.

The current system should be extended to include and expanded range of GSN nodes. Additionally, patterns in argumentation could be considered as part of safety case generation. This work would fit alongside the inclusion of safety case analysis and validity checking during the generation process.

Meta-data could be included into the domain theory by an additional explanation pass of the tree prior to augmentation. This could allow for additional information about the tree to be included in the safety case generation process. Furthermore, the collection and integration of data collected in simulation could be implemented in order to generate richer safety cases.

Currently the work has considered how a given technique, which is able to produce its own safety artefacts, fits into a specification-based certification process. Further work could consider how the adaptive safety cases produced here could fit into a larger adaptive argument, as proposed in [74]. The approach developed in this work could be viewed as an argument fragment generation activity. The safety fragments can then be assembled into a larger argument, as shown in [62].

The existing implementation could be applied to generate GSN structures from any tree structure. This could apply to decision trees or theorem provers as well as other techniques. This could expand the utility of the technique to other application domains.

# Bibliography


[1] T. Kelly and R. Weaver, "The Goal Structuring Notation – A Safety Argument Notation," *Elements*, 2004.

[2] T. A. Faculty and D. Kahn, "The Design and Development of a Modular Avionics System," 2001.

[3] S. Madan and K. E. Bollinger, "Applications of artificial intelligence in power systems," *Electric Power Systems Research*, vol. 41, no. 2, pp. 117–131, may 1997.

[4] A. Wilson and T. Preyssler, "Incremental certification and Integrated Modular Avionics," *Aerospace and Electronic Systems Magazine, IEEE*, vol. 24, no. 11, pp. 10–15, 2009.

[5] C. B. Watkins and R. Walter, "Transitioning from federated avionics architectures to Integrated Modular Avionics," in *Digital Avionics Systems Conference, 2007. DASC '07. IEEE/AIAA 26th*, 2007, pp. 2.A.1–1–2.A.1–10.

[6] I. Moir and A. Seabridge, *Aircraft Systems : Mechanical, Electrical and Avionics Subsystems Integration (3rd Edition).* Hoboken, NJ, USA: Wiley, 2008.

[7] J. S. Jang and C. J. Tomlin, "Design and Implementation of a Low Cost, Hierarchical and Modular Avionics Architecture for the DragonFly UAVs," in *Proc. AIAA Guidance, Navigation, and Control Conference*, 2002, pp. 4465–4477.

[8] T. W. Gould, S. R. Stovner, R. J. Reuter, S. W. Engdahl, J. M. Parker, and B. P. Kesterson, "Remote aircraft manufacturing, monitoring, maintenance and management system," 2009. [Online]. Available: http://www.google.com/patents/US7636568

[9] S. J. Pan and Q. Yang, "A survey on transfer learning," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 22, no. 10, pp. 1345–1359, 2010.

[10] M. Sharma, M. Holmes, J. Santamaria, A. Irani, C. Isbell, and A. Ram, "Transfer learning in real-time strategy games using hybrid CBR/RL," in *Proceedings of the 20th international joint conference on Artifical intelligence*, ser. IJCAI'07. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007, pp. 1041–1046. [Online]. Available: http://dl.acm.org/citation.cfm?id=1625275.1625444

[11] G. DeJong, "Explanation-Based Learning," in *Computer Science Handbook*, 2nd ed., A. Tucker, Ed., 2004.

[12] J. Lopez, P. Royo, C. Barrado, and E. Pastor, "Modular avionics for seamless reconfigurable UAS missions," pp. 1.A.3–1–1.A.3–10, 2008.

[13] S. A. Jacklin, "Closing the Certification Gaps in Adaptive Flight Control Software," in *AIAA Guidance Navigation and Control Conference*. Honolulu: AIAA, 2008.

[14] L. Karunarathne, J. T. Economou, and K. Knowles, "Model based power and energy management system for PEM fuel cell/ Li-Ion battery driven propulsion system," in *Power Electronics, Machines and Drives (PEMD 2010), 5th IET International Conference on*, 2010, pp. 1–6.

[15] J. Moreno, M. E. Ortuzar, and J. W. Dixon, "Energy-management system for a hybrid electric vehicle, using ultracapacitors and neural networks," *Industrial Electronics, IEEE Transactions on*, vol. 53, no. 2, pp. 614–623, 2006.

[16] A. Stranjak, "Agent-based Control of Autonomous Power Management on Unmanned Platforms," in *4th SEAS DTC Technical Conference*, no. 2, Edinburgh, 2009.

[17] K.-S. Jeong, W.-Y. Lee, and C.-S. Kim, "Energy management strategies of a fuel cell/battery hybrid system using fuzzy logics," *Journal of Power Sources*, vol. 145, no. 2, pp. 319–326, aug 2005.

[18] D. Choi and S. Ohlsson, "Interoperating Learning Mechanisms in a Cognitive Architecture," in *AAAI Fall Symposium Series*, 2011.

[19] N. R. C. Committee on Materials, Structures, and Aeronautics for Advanced Uninhabited Air Vehicles, Commission on Engineering and Technical Systems, *Uninhabited Air Vehicles : Enabling Science for Military Systems*. Washington, DC, USA: National Academies Press, 2000.

[20] K. VanLehn and R. M. Jones, "Learning Physics Via Explanation-Based Learning of Correctness and Analogical Search Control,," oct 1998.

[21] T. Ellman, "Explanation-based learning: a survey of programs and perspectives," *ACM Comput. Surv.*, vol. 21, no. 2, pp. 163–221, jun 1989.

[22] N. S. Flann and T. G. Dietterich, "A Study of Explanation-Based Methods for Inductive Learning," *Machine Learning*, vol. 4, no. 2, pp. 187–226, nov 1989.

[23] T. M. Mitchell, R. M. Keller, and S. T. Kedar-Cabelli, "Explanation-based generalization: A unifying view," *Machine Learning*, vol. 1, no. 1, pp. 47–80, mar 1986.

[24] J. A. Robinson, "A Machine-Oriented Logic Based on the Resolution Principle," *J. ACM*, vol. 12, no. 1, pp. 23–41, jan 1965.

[25] K. Hoder and A. Voronkov, "Comparing unification algorithms in first-order theorem proving," in *Proceedings of the 32nd annual German conference on Advances in artificial intelligence*, ser. KI'09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 435–443.

[26] E. Hirowatari and S. Arikawa, "Incorporating Explanation-Based Generalization with Analogical Reasoning," *Bulletin of Informatics and Cybernetics*, vol. 26, pp. 13–33, 1994.

[27] M. Steven, "Quantitative results concerning the utility of explanation-based learning," *Artificial Intelligence*, vol. 42, no. 2–3, pp. 363–391, mar 1990.

[28] S. G. Hans Bandemer, *Fuzzy Sets Fuzzy Logic Fuzzy Methods with Applications*. John Wiley & Sons, Inc., 1995.

[29] L. A. Zadeh, "Fuzzy sets," *Information and Control*, vol. 8, no. 3, pp. 338–353, jun 1965.

[30] Z. L.A., "The concept of a linguistic variable and its application to approximate reasoning," *Information Sciences*, vol. 8, no. 3, pp. 199–249, 1975.

[31] R. Krishnapuram and F. Chung-Hoon Rhee, "Compact fuzzy rule base generation methods for computer vision," pp. 809–814 vol.2, 1993.

[32] T. Cazenave, "Self Fuzzy Learning," *Strategy*, pp. 1–8, 1996. [Online]. Available: www.lamsade.dauphine.fr/ cazenave/papers/fuzzy.pdf

[33] V. Novák, "First-Order Fuzzy Logic," *Studia Logica: An International Journal for Symbolic Logic*, vol. 46, no. 1, pp. 87–109, 1987.

[34] G. Feng, "A Survey on Analysis and Design of Model-Based Fuzzy Control Systems," *Fuzzy Systems, IEEE Transactions on*, vol. 14, no. 5, pp. 676–697, 2006.

[35] 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, jan 1975.

[36] A. Riid, "Transparent Fuzzy Systems: Modeling and Control," 2002.

[37] A. A. Kaur and A. A. Kaur, "Comparison of Mamdani-Type and Sugeno-Type Fuzzy Inference Systems for Air Conditioning System," *International Journal of Soft Computing & Engineering*, vol. 2, no. 2, pp. 323–325, 2012.

[38] J. G. Carbonell, "Derivational Analogy and Its Role in Problem Solving," in *AAAI-83*, 1983.

[39] M. N. Huhns and R. D. Acosta, "Argo: a system for design by analogy," pp. 53–68, 1988.

[40] B. Falkenhainer, "An examination of the third stage in the analogy process: verification-based analogical learning," in *Proceedings of the 10th international joint conference on Artificial intelligence - Volume 1*, ser. IJCAI'87. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1987, pp. 260–263.

[41] T. R. Davies, "A logical approach to reasoning by analogy," in *In IJCAI-87*. Morgan Kaufmann, 1987, pp. 264–270.

[42] M. Klenk and K. Forbus, "Domain transfer via cross-domain analogy," *Cognitive Systems Research*, vol. 10, no. 3, pp. 240–250, sep 2009.

[43] R. L. de Mántaras and E. Plaza, "Case-Based Reasoning: an overview," *AI Commun.*, vol. 10, no. 1, pp. 21–29, jan 1997.

[44] T. Ishikawa and T. Terano, "Analogy by abstraction: Case retrieval and adaptation for inventive design expert systems," *Expert Systems with Applications*, vol. 10, no. 3–4, pp. 351–356, 1996.

[45] T. Könik, P. O'Rorke, D. Shapiro, D. Choi, N. Nejati, and P. Langley, "Skill transfer through goal-driven representation mapping," *Cognitive Systems Research*, vol. 10, no. 3, pp. 270–285, sep 2009.

[46] J. Rushby, "New challenges in certification for aircraft software," in *Proceedings of the ninth ACM international conference on Embedded software*, ser. EMSOFT '11. New York, NY, USA: ACM, 2011, pp. 211–218.

[47] S. Linling, Z. Wenjin, and T. Kelly, "Do safety cases have a role in aircraft certification?" *Procedia Engineering*, vol. 17, no. 0, pp. 358–368, 2011.

[48] J. Rushby, "A safety-case approach for certifying adaptive systems," in *In AIAA Infotech@Aerospace Conference, American Institute of Aeronautics and Astronautics, John Rushby*, 2009.

[49] G. Romanski and V. Inc, "The Challenges of Software Certification," *CrossTalk, Sep*, vol. 2001, 2001.

[50] R. Alexander, M. Hall-May, and T. Kelly, "Certification of autonomous systems," in *Proceedings of the 2nd Systems Engineering for Autonomous Systems (SEAS) Defence Technology Centre (DTC) Annual Technical Conference*, 2007.

[51] R.-R. AOS, BAE Systems, Cassidian, Cobham, QinetiQ and Thales., "ASTRAEA Project." [Online]. Available: http://www.astraea.aero/

[52] A. Jones, "UAS Virtual Certification," in *2011 ASTRAEA Conference – Opening Airspace to UAS*, London, 2011.

[53] N. Cameron, M. Webster, M. Jump, and M. Fisher, "Certification of a Civil UAS: A Virtual Engineering Approach." in *2011 AIAA Modelling Simulation and Technologies Conference and Exhibit*. Portland, Oregon: AAAI Press, 2011, pp. 1–15.

[54] S. A. Jacklin, J. M. Schumann, P. P. Gupta, M. Richard, K. Guenther, and F. Soares, "Development of Advanced Verification and Validation Procedures and Tools for the Certification of Learning Systems in Aerospace Applications," feb 2005.

[55] Z. Kurd, T. Kelly, and J. Austin, "Developing artificial neural networks for safety critical systems," *Neural Computing and Applications*, vol. 16, no. 1, pp. 11–19, oct 2006. [Online]. Available: http://link.springer.com/10.1007/s00521-006-0039-9

[56] B. Siddhartha, C. Darren, M. David, M. Joseph, and E. Eric, "Certification Considerations for Adaptive Systems," 2015.

[57] R. Bloomfield and P. Bishop, "Safety and Assurance Cases: Past, Present and Possible Future – an Adelard Perspective," C. Dale and T. Anderson, Eds. Springer London, 2010, pp. 51–67.

[58] M. Timperley, M. Mokhtar, and J. Howe, "Adaptive Safety Arguments and Explanation-Based Learning," in *Proceedings of Workshop SASSUR (Next Generation of System Assurance Approaches for Safety-Critical Systems) of the 32nd International Conference on Computer Safety, Reliability and Security*, M. ROY, Ed., University of Central Lancashire. Toulouse, France: HAL, 2013. [Online]. Available: http://hal.archives-ouvertes.fr/hal-00848501

[59] J. Rushby, "Just-in-Time Certification," in *Engineering Complex Computer Systems, 2007. 12th IEEE International Conference on*, 2007, pp. 15–24.

[60] J. Rushby, "Runtime Certification," in *Runtime Verification*, ser. Lecture Notes in Computer Science, M. Leucker, Ed. Berlin, Heidelberg: Springer-Verlag, 2008, vol. 5289, ch. Runtime Ce, pp. 21–35.

[61] E. Denney, G. Pai, and J. Pohl, "AdvoCATE: An assurance case automation toolset," in *Computer Safety, Reliability, and Security*. Springer, 2012, pp. 8–21.

[62] E. Denney and G. Pai, "Automating the Assembly of Aviation Safety Cases," *Reliability, IEEE Transactions on*, vol. 63, no. 4, pp. 830–849, dec 2014.

[63] R. M. French, "The computational modeling of analogy-making," *Trends in Cognitive Sciences*, vol. 6, no. 5, pp. 200–205, may 2002.

[64] E. Denney, G. Pai, and I. Habli, "Perspectives on software safety case development for unmanned aircraft," pp. 1–8, 2012.

[65] J. V. de Oliveira, "Semantic constraints for membership function optimization," *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, vol. 29, no. 1, pp. 128–138, 1999.

[66] E. Alonso, M. D'Inverno, D. Kudenko, M. Luck, and J. Noble, "Learning in multi-agent systems," *The Knowledge Engineering Review*, vol. 16, no. 03, pp. 277–284, 2001.

[67] "X-Plane SDK: Autopilot State," 2009. [Online]. Available: http://www.xsquawkbox.net/xpsdk/mediawiki/Sim/cockpit/autopilot/autopilot_state

[68] M. Cook, *Flight Dynamics Principles*, 2nd ed. Butterworth Heinemann, 2007.

[69] D. Gentner and K. J. Holyoak, "Reasoning and learning by analogy." *The American psychologist*, vol. 52, no. 1, pp. 32–34, jan 1997.

[70] G. Dejong, "Toward Robust Real-World Inference: A New Perspective on Explanation-Based Learning," 2006.

[71] M. H. Burstein, "A Model of Learning by Incremental Analogical Reasoning and Debugging," *Machine Learning: An Artificial Intelligence Approach (Vol. 2)*, 1986.

[72] R. D. Hawkins, I. Habli, and T. Kelly, "The Principals of Software Safety Assurance," in *International System Safety Conferene (ISSC)*, Boston, Massachusetts USA, 2013. [Online]. Available: http://www-users.cs.york.ac.uk/ ihabli/Papers/2013Habli_ISSC.pdf

[73] R. Hawkins, I. Habli, and T. Kelly, "Principled construction of software safety cases," in *SAFECOMP 2013-Workshop SASSUR (Next Generation of System Assurance Approaches for Safety-Critical Systems) of the 32nd International Conference on Computer Safety, Reliability and Security*, M. ROY, Ed., Toulouse, France, 2013, p. NA.

[74] E. Denney, I. Habli, G. Pai, I. Habli, G. Pai, I. Habli, G. Pai, and I. Habli, "Dynamic Safety Cases for Through-life Safety Assurance," in *37th International Conference on Software Engineering - New Ideas and Emerging Results (NIER)*, no. 2, Florence, Italy, may 2015.

[75] K. R. Levi, D. L. Perschbacher, M. A. Hoffman, C. A. Miller, B. B. Druhan, and V. L. Shalin, "An explanation-based-learning approach to knowledge compilation: a Pilot's Associate application," *IEEE Expert*, vol. 7, no. 3, pp. 44–51, jun 1992.

[76] M. Nicholson, P. Conmy, I. Bate, and J. Mcdermid, "Generating and Maintaining a Safety Argument for Integrated Modular Systems," in *5th Australian Workshop on Industrial Experience with Safety Critical Systems and Software*. Melbourne: Australian Computer Society, 2000, pp. 31–41.

[77] J. L. Fenn, R. D. Hawkins, P. J. Williams, T. P. Kelly, M. G. Banner, and Y. Oakshott, "The Who, Where, How, Why And When of Modular and Incremental Certification," in *System Safety, 2007 2nd Institution of Engineering and Technology International Conference on*, oct 2007, pp. 135–140.

# Appendices

# Appendix A

# Publications

# Adaptive Safety Arguments and Explanation-Based Learning

Matt Timperley, Maizura Mokhtar, and Joe Howe

Centre for Energy and Power Management
University of Central Lancashire
Preston, Lancashire, United Kingdom, PR1 2HE
{mtimperley,mmokhtar,jmhowe}@uclan.ac.uk

**Abstract.** Software for use in aviation requires certification. This certification is based on a safety argument. These arguments are formed of claims that are linked to evidence about the system. Adaptive systems are a grey area within the current certification guidelines (DO-178 document). Safety cases (sometimes called safety arguments) link claims and evidence in support of an overall safety argument. This paper argues that it is rational to have an adaptive safety argument for an adaptive system. This is illustrated by considering an adaptive controller that uses Explanation-Based Learning (EBL) to generate both control laws and a safety argument, represented using Goal-Structuring Notation. An adaptive safety argument, when coupled with analytical tools, could be used to form the adaptive portion of an otherwise standards-based certification argument. If the rest of the argument holds then the argument should hold for any state where the adaptive safety cases remain valid.

## 1 Introduction

Certification for aviation software, as part of a platform, can be gained by any sufficient safety argument. A safety argument makes claims about the system that are substantiated by evidence. DO-178 (currently at revision C) is a guideline on how to develop a system that is likely to be certifiable. Addressing the guidelines in DO-178, satisfying criteria for levels of integrity, is a prescriptive approach to obtaining certification. This relates to the software system and is part of a larger safety argument for the whole platform, whose different aspects have different guideline documents.

A technique that has gained some influence in making safety arguments is the safety case. A safety case links claims and evidence into overall arguments. An argument for certification could be presented using safety cases. Adaptive control is an area highlighted as having additional challenges in an entirely specification-based framework. Adaptive control may benefit from having an adaptive safety argument [1].

This argument will be illustrated assuming a controller on board an Unmanned Autonomous System (UAS), which elicits control laws using Explanation-Based Learning (EBL). The structures produced by EBL could be used to form

safety cases that reflect a learning system by adapting alongside the controller. Consider an adaptive controller capable of generating safety cases, in addition to control laws, sufficient to form a safety argument for its own action-response mappings. This could demonstrate both coverage and determinism at the point where the generated safety cases hold i.e. until further changes to the system are made. An explanation of each control law would show, if unconstrained by training information, all rules that are used by the system until further learning occurs. Past arguments could be stored as baselines. Or a separate baseline, which constrains the form of future arguments, could be developed. Before introducing any control law the argument could be updated and validated, possibly using the approach in [2], to give some measure of confidence that the new rule would have no compromising impact on system behaviour. This could automate part of the certification process for an adaptive controller.

This paper proposes the adaptive certification of EBL-based controllers. The paper is divided into 3 sections. Section 2 describes certification and introduces the argument notation being considered alongside the adaptive control method proposed, introduced in Section 3. Section 4 concludes the paper.

## 2 Aviation Certification

The reason for certification is to have a level of confidence that the software being produced will safely serve its purpose. Certification is awarded by a regulatory body, such as the FAA in the US [3] or CAA in the UK. Once certified, the software can be included in airborne avionics. DO-178 (US), or ED12 in Europe, is a guideline on how to develop a software-based system that is likely to be certifiable. Since these documents are guidelines they are not absolute, arguments may be presented in other ways, though frequently these guidelines are followed [3]. Flight critical software must demonstrate compliance to airworthiness standards. DO-178 is identified as the method of choice for this. Alternatives are permitted but not defined. There are limitations to specification-oriented certification [4], which particularly apply to adaptive systems. One problem that can occur, when certifying adaptive systems, is in interpreting the model that the system has learned [5].

It is worth noting that not all cases are covered by the document, for example UAS specific software and adaptive control. Adaptive flight control software changes its gains, which could be conceived as its situation-action bindings, after deployment. This creates difficulties in adhering to the certification guidelines. The main areas that adaptive systems may require additional work to conform, from [6], based on DO-178B, are: (i) defining software performance requirements, (ii) providing a software verification plan, (iii) defining software requirements and derived requirements, (iv) providing software verification test cases and procedures, (v) providing a Plan for Software Aspects of Certification (PSAC), and providing software life cycle data.

Non-adaptive controllers cannot alter their gains to match evolving situations, such as decay in system responsiveness. Adaptive controllers can alter

gains after deployment automatically, but this comes with the issue that certification guidelines are lacking. Adaptive systems would require adaptation of the component safety cases, to keep the safety argument valid for each new adaptation.

## 3 Certifying Adaptive Systems using EBL

Goal-Structuring Notation (GSN) has been adopted in the aviation industry for presenting safety cases [7]. GSN is a graphical notation used to present safety arguments in a manner that aims to minimise ambiguity and avoid poorly written safety cases. GSN presents arguments as a hierarchy of nodes representing supporting claims. There are nodes for representing the goals, strategies for achieving goals, solutions (evidence to satisfy arguments), context information and unfinished goals.

Adaptive generation of safety cases would be advantageous, especially in line with adaptations to the controller. A technique that can potentially derive control laws and generate a safety case using GSN is Explanation-Based Learning (EBL). Explanation Based Learning (EBL) was developed as an alternative analytic learning technique that, rather than using many examples to draw a conclusion, draws a conclusion and generalises it from one example [8]. This was posited on the basis that the inclusion of domain knowledge would allow the algorithm to explain an example. This explanation can then be generalised so as to be useful in similar situations. EBL typically links rules using deduction (e.g. Unification) to form an explanation structure. By employing deduction, EBL is potentially of a high enough fidelity for application to safety-critical software. However, the conclusions drawn by EBL are a direct result of the specifics of the domain theory, which are usually hand-coded expert knowledge. The confidence in the control laws elicited by EBL is therefore dependant on the confidence engendered by the knowledge engineering phase, which produces the domain theory.

Adaptations have been developed that concern the generalisations between examples, though each can potentially gain useful knowledge from a single example [9]. EBL can be considered as a search through the space of all possible explanations. In order to make this search tractable a bias is used. The domain knowledge encoded provides a bias by restricting the explanations that fit within the model defined by the domain knowledge. This search is further restricted by a training example to give a specific explanation. This explanation can be generalised and chunked into a new rule. Chunking is often done by computing the most general preconditions i.e. flattening the explanation structure and taking the lowest nodes which are domain knowledge then using these in concert to infer the goal. This, if all the nodes used are operational, gives the new rule. Operationality is expressed with an operationality criterion. Operational, in terms of EBL, has has several posited definitions and in many cases relates in some way to computational efficiency [8], [10]. This can be considered as a filter for rules whose inclusion improves the performance of the controller. The final com-

ponent in EBL is the goal. The algorithm attempts to deductively explain why the training instance is a positive example of the goal concept.

EBL has been applied within the aerospace domain [11]. Using EBL to generate control laws is an adaptive technique as explanations form new rules that can change the action of the controller for a given situation. This is analogue to changing the gains of the controller. EBL explanations help to justify any situation-action bindings and thereby support the determinism property. In order to show coverage a series of safety cases could be produced using the GSN notation as there is some parallel between GSN and a typical way of representing explanation structures (Table 1).

**Table 1.** Parallels in representation between EBL and GSN.

| EBL | GSN |
|---|---|
| Hierarchical structure based on goal decomposition. | Hierarchical structure based on goal decomposition. |
| Goal Node. | Goal Node. |
| Unification with observation. | Solution Node. |
| Unification with domain knowledge. | (Sub)Goal Node / Strategy Node. |
| Preconditions. | Context Node. |
| Conceptual gaps in knowledge | Undeveloped Goal. |

### 3.1 Parallel Representations

EBL and GSN are both hierarchical decompositions headed by a goal. Both structure (EBL) and argument (GNS), via a series of interconnecting nodes, and may eventually terminate in branches where a claim is substantiated. The goals and claims need to be considered analogous in order for the comparison to hold. Context nodes, which show when a node applies, could be considered similar to preconditions. These preconditions could be rule meta-data, preconditions on operational actions or even be embedded in the domain knowledge. Solution nodes would be the analogue of unifications to training data terms in EBL. Undeveloped goals needs no analogue though could represent dearths in domain knowledge. If expanded upon, the undeveloped goals can be used to drive further knowledge engineering efforts and improve the controller.

Since there is a parallel in representation it is possible to use GSN to represent an EBL explanation. Presenting explanations in this form could form a (partial) safety case, depending upon the system. A system incorporating EBL could be used as an aid to safety argumentation, possibly by generating skeleton arguments. This would be particularly useful when EBL is the basis for control laws that are automatically generated. This clear and human-readable

representation aid interpretation of the internal model to find and correct faulty assumptions or rules. In addition, being able to generate safety cases, or update them, for an adaptive technique is a good way to illustrate both the determinism and coverage of the system at any time. Choosing appropriate goals and domain representation directly impacts the value of the generated safety case.

## 4 Conclusions

An argument for adaptive safety cases forming part or all of a safety argument has been presented alongside consideration of EBL within the aviation industry. The ability for a system to explain its actions in relation to safety concerns could be useful in the context of aviation certification. EBL could be used as an aid towards generating safety arguments, especially where the conclusions that require justification were reached by EBL initially. This representation could also be useful in identifying faulty logic, thus aiding interpretation of the internal model. Additionally, these safety arguments can be used to demonstrate coverage and determinism, both key properties of safety arguments. Further work will involve applying this approach to a case study.

**Acknowledgments** This work is funded by EPSRC CASE award EP/I501312/1, and is supported by the BAE Systems, UK.

## References

1. Rushby, J.: Runtime Certification. Volume 5289 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (2008) 21–35
2. Denney, E., Pai, G., Habli, I.: Perspectives on software safety case development for unmanned aircraft (2012)
3. Romanski, G., Inc, V.: The Challenges of Software Certification (2001)
4. Rushby, J.: New challenges in certification for aircraft software. In: Proceedings of the ninth ACM international conference on Embedded software. EMSOFT '11, New York, NY, USA, ACM (2011) 211–218
5. Alexander, R., Hall-May, M., Kelly, T.: Certification of autonomous systems. In: Proceedings of the 2nd Systems Engineering for Autonomous Systems (SEAS) Defence Technology Centre (DTC) Annual Technical Conference. (2007)
6. Jacklin, S.A.: Closing the Certification Gaps in Adaptive Flight Control Software (2008)
7. Kelly, T., Weaver, R.: The Goal Structuring Notation A Safety Argument Notation. Elements (2004)
8. Ellman, T.: Explanation-based learning: a survey of programs and perspectives. ACM Comput. Surv. **21**(2) (June 1989) 163–221
9. Flann, N.S., Dietterich, T.G.: A Study of Explanation-Based Methods for Inductive Learning. Machine Learning **4**(2) (November 1989) 187–226
10. Steven, M.: Quantitative results concerning the utility of explanation-based learning. Artificial Intelligence **42**(23) (March 1990) 363–391
11. Levi, K.R., Perschbacher, D.L., Hoffman, M.A., Miller, C.A., Druhan, B.B., Shalin, V.L.: An explanation-based-learning approach to knowledge compilation: a Pilot's Associate application. IEEE Expert **7**(3) (June 1992) 44–51