The CUE Logic: Embracing Causality in SAT Based Reasoners

Marcus Björklund
c98mbj@cs.umu.se

November 8, 2004

Abstract

Commonsense reasoning normally involves both causal and evidential rules. While causal rules can be applied at any time, evidential rules may be applied only if there is no other way to explain a certain symptom. To avoid reasoning counter to common sense, systems that admit mixtures of causal and evidential rules must employ mechanisms to inhibit evidential rules for all effects that are established through causal rules.

This thesis explores a way of incorporating causation in any SAT-based reasoner by implementing a logic based on the ideas of the C-E system. The logic resembles a Truth Maintenance System and has been implemented as an extension to the default reasoning system Z-log.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Problems associated with representing causal relationships plague many proposals for default reasoning. Approaches such as circumscription[4], Reiter's default logic[9] and System-Z[8], which are all based on extensions to classical logic, draw counter-intuitive conclusions because they fail to block the chaining of the following default expressions: [6]

r_1: "If the grass is wet, conclude it rains."

r_2: "If the sprinkler is on, the grass will be wet."

Finding the sprinkler on, we do not wish to conclude from these two rules that it rains, nor that it will rain. Hence, we can apply r_2, but we must block r_1.

Almost every default rule can be sorted into one of two categories: *expectation-evoking* or *explanation-evoking*. Rules of the first type describe how causes and effects are related in the outside world (e.g. "Fire typically causes smoke"), while rules of the second type go the other way and describe how humans reason about the world to find explanations for things (e.g. "Smoke suggests fire"). Expectation-evoking rules are commonly referred to as *causal* rules. Explanation-evoking rules are known as *evidential* or *diagnostic* rules.

The usual meaning of a default rule $\alpha \rightarrow \beta$ is that we can assume β whenever α is believed to be true. This is however not true for rules of the type "If observation

α then hypothesis β" (i.e. evidential rules). Such rules should apply only when there is no other explanation available for the observed effect.

The easiest way to avoid problems such as the one described above is to make the reasoning system admit only one type of default rules. The MYCIN system[11], for example, admits only evidential rules and thus can perform simple diagnoses but is unable to combine diagnosis with prediction. The common practice in Bayesian analysis is to use only causal rules: Knowledge about the domain is input in an "If cause then effect" format, while diagnoses are *derived* by explanation-seeking procedures rather than by explicit evidential rules. Such causal-based systems enjoy features of stability, modularity and provide a natural, declarative way of expressing world knowledge.

In practice, however, most default reasoning systems do admit a mixture of causal and evidential rules. The reason for the mixture is that despite the advantages of purely causal systems, it is very tempting for rule-writers to express procedural knowledge leading from familiar situations to previously successful actions or guesses (e.g., that symptoms suggest diseases).

This thesis explores a notion of extending a SAT based inference system with ability to correctly handle mixtures of causal and evidential rules. We have chosen to implement our approach as an extension to Z-log[5], although it can be applied to any system based on propositional logic and satisfiability testing.

1.1 Organization of this Thesis

In Chapter 2 we review System-Z, Z-log and the C-E system. Chapter 3 describes the invented logic that has been used to solve the problem. In Chapter 4 we show some examples and compare the results from our causal reasoning to results from reasoning without causality. We also show how causal modeling and default reasoning work together. In chapter 5 we evaluate our extension to Z-log and talk about problems and limitations and possible future work.

Appendix A focuses on implementation and explains how Z-log has been redesigned, and appendix B provides a brief glossary.

Chapter 2

Background

2.1 System-Z

System-Z[8] is an inference system for default reasoning. It solves some of the problems associated with default logic, such as overrides, irrelevant information and transitivity[5].

Definition 1 *A* world ω *is an assignment of truth-values to a set of propositional variables.*

2.1.1 World Rankings

In System-Z, worlds which violate default rules are accepted with surprise. The degree of surprise associated with a world ω is denoted by a non-negative integer $\kappa(\omega) \geq 0$. The function $\kappa(\omega)$ is known as the *world ranking.*

A default rule $\varphi \rightarrow \phi$ means that when a world satisfies φ, it will typically also satisfy ϕ. This is captured in the admissibility requirement on κ:

$$\varphi \rightarrow \phi \Leftrightarrow \kappa(\varphi \wedge \phi) < \kappa(\varphi \wedge \neg\phi) \tag{2.1}$$

There are many world rankings κ that satisfy this requirement. The *most compressed* admissible ranking is a unique ranking in which $\kappa(\omega)$ attains the minimal

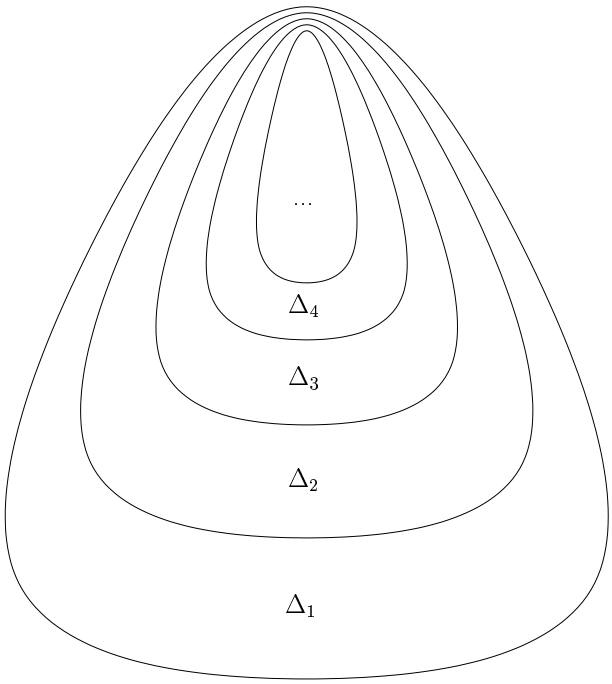

Figure 2.1: Ranked Z partition

value possible under the constraints in Δ. We can use this minimal ranking function to accommodate the violation of defaults. Hence, whenever we find a world in violation of a default rule, we accept it with surprise, but with minimal surprise.

2.1.2 Ranked Partitions

Instead of using a world ranking function directly, in System-Z the knowledge base is first processed in order to build a *ranked partition* $\Delta_1, \Delta_2, \ldots, \Delta_m$, that summarizes the interactions of defaults and sets the level of surprise experienced when a default is violated. This corresponds to the world ranking function in the following way: $\kappa_Z(\omega)$ is the greatest k such that ω falsifies a default in $\bigcup_{i=k}^{m} \Delta_i$. The algorithm used to build the partition is explained in detail in [5].

Figure 2.1 illustrates how the ranked partition can be thought of: A world ω that falsifies at least one default will have $\kappa(\omega) \geq 1$. We can determine $\kappa(\omega)$ by "peeling off" layer by layer until ω no longer falsifies any of the remaining defaults.

System-Z thus sees a world violating a default at an inner lever (higher rank) as more surprising than a world violating a default in an outer layer (lower rank). This gives us the entailment semantics for System-Z:

$$\varphi \vdash_Z \phi \quad \text{iff} \quad \kappa_Z(\varphi \wedge \phi) < \kappa_Z(\varphi \wedge \neg\phi) \tag{2.2}$$

Hard Rules

Hard rules, i.e. rules that no world should be allowed to falsify, can be realized in System-Z by placing them at rank *inifinity* so that $\kappa_Z(\omega) = \infty$ for every world ω that falsifies a hard rule.

2.1.3 System-$\hat{Z}$: Improving System-Z

Definition 2 *A rule $\varphi \rightarrow \phi$ is said to be* verified *when both its antecedent φ and its consequent ϕ are true.*

A default rule that, when verified, does not falsify any other rule in the knowledge base will be placed in the first partition level Δ_1 by the partition building algorithm. A default that overrides another default will be placed in at a higher rank than the one it overrides. As a consequence, System-Z sees worlds that falsify overrides as more surprising than worlds that falsify non-overridden defaults. This runs counter to common sense, because we should be more surprised when a rule that has no exceptions is falsified: The mere presence of an override signals that the material to which the default applies is subject to more non-monotonic behavior[5].

These considerations have lead to a modification of System-Z, called System-$\hat{Z}$, that treats violations of non-overridden defaults with greater doubt, while preserving the necessary ordering properties embedded in the most compressed admissible ranking built by System-Z. This is realized in System-$\hat{Z}$ by further processing the partition built by System-Z to promote "unpinned"[1] defaults to higher ranks. The

[1] A default rule is "pinned" to a partition level if either the default's promotion would cause a higher ranked default to no longer be required to be at its minimal Z determined rank, or it would cause an inconsistency within the higher rank.[5]

procedure to compute whether ϕ follows from φ is the same as in System-Z, but the new, pinned partition is used instead.

In addition to handling overrides, irrelevant information and transitivity, System-$\hat{Z}$ also sanctions inheritance of stable properties to exceptional subclasses.

2.1.4 Z-log: An Implementation

Z-log represents the most advanced direct System-Z implementation to date [5]. The first version of Z-log was a command-line tool built after a LISP prototype and consisted of 15 classes in approximately 3500 lines of Java code [3]. It was later extended to include a graphical interface and black-box analysis methods that enabled 'what if?', 'are you sure?' and 'why?' queries, which made it more useful. The applet provides an interface in which the user can set or modify the truth-values for a set of input propositions and immediately view the changes in the output conclusions. The applet and the new functions provided by it are covered in detail in [12].

2.2 The C-E System

The C-E logic described in this section was devised to maintain plausibility in a mixed system, where causal and evidential rules reside side by side, each labeled by its type name. [7]

In the C-E system, each default rule is labeled as either *C-def* ("causal") or *E-def* ("evidential"). The former will be distinguished by the symbol $\rightarrow_C$, as in $Fire \rightarrow_C Smoke$ (read fire causes smoke), and the latter by $\rightarrow_E$, as in $Smoke \rightarrow_E Fire$ (read smoke is evidence for fire).

Correspondingly, each believed proposition is labeled by a distinguishing symbol, E or C. A proposition P is *E-believed*, written $E(P)$, if it is a direct consequence of only *E-def* rules. If P can be established as a direct consequence of a *C-def* rule, it is said to be *C-believed*, written $C(P)$. The semantics of the C-E distinction are captured by the three inference rules in figure 2.2.

The rules of the C-E system imply that conclusions can attain *E-believed* status

$$\text{a)}\ \frac{P \rightarrow_C Q \quad C(P)}{C(Q)} \qquad \text{b)}\ \frac{P \rightarrow_C Q \quad E(P)}{C(Q)} \qquad \text{c)}\ \frac{P \rightarrow_E Q \quad E(P)}{E(Q)}$$

Figure 2.2: Inference rules in the C-E system

only by a chain of purely *E-def* rules. *C-believed* conclusions, on the other hand, can be obtained from a mixture of *C-def* and *E-def* rules. The rules can license the use of $A \rightarrow B$ and $B \rightarrow A$ without falling into the circular reasoning trap. Iterative application of these two rules will never cause a *C-believed* proposition to become *E-believed*, because at least one of the rules must be a *C-def* rule.

2.2.1 Compared to Belief Networks

The ideas behind the C-E system can be illustrated by comparing them to belief networks. An introduction to belief networks in given in [10].

Intercausal Inferences

Causes that share a consequence interact when the consequence is observed. Consider the belief network in figure 2.3; as long as the truth-value of $Cough$ is unknown, $Bronchitis$ and $Cold$ are independent, i.e. the status of $Cold$ does not affect the probability of $Bronchitis$, and vice versa:

$$P(Bronchitis|Cold) = P(Bronchitis)$$
$$P(Cold|Bronchitis) = P(Cold)$$

Once $Cough$ is observed, however, $Bronchitis$ and $Cold$ will interact so that evidence for one will cause the probability of the other to decrease. This pattern of reasoning is also known as "explaining away":

$$P(Bronchitis|Cough \wedge Cold) \leq P(Bronchitis|Cough)$$
$$P(Cold|Cough \wedge Bronchitis) \leq P(Cold|Cough)$$

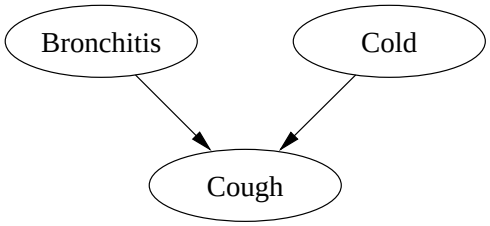

Figure 2.3: *Cough* is a common consequence of *Bronchitis* and *Cold*

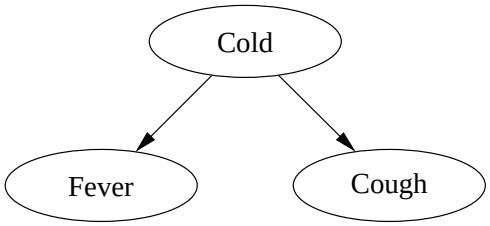

Figure 2.4: *Cold* is a common cause of *Fever* and *Cough*

In the C-E system, a C-believed proposition is one that is known to have been caused by something, i.e. it is already explained. Hence, C-believed conclusions cannot be used as evidence for other explanations.

Interevidential Inferences

Two consequences of a common cause interact only when the cause is unconfirmed. Consider the network in figure 2.4: If *Cold* has not been confirmed, observing *Fever* increases the probability of *Cold*, and an increased probability of *Cold* increases the probability of *Cough*:

$$P(Fever|Cough) \geq P(Fever)$$
$$P(Cough|Fever) \geq P(Cough)$$

If the status of $Cold$ is known, $Fever$ and $Cough$ become conditionally independent, and each of their probabilities depend only on the status of $Cold$:

$$P(Fever|Cold \wedge Cough) = P(Fever|Cold)$$
$$P(Cough|Cold \wedge Fever) = P(Cough|Cold)$$

The C-E system allows E-believed propositions to trigger both evidential and causal rules.

Chapter 3

Approach

This chapter describes a logic called *the CUE logic*, that allows mixing causal and evidential rules by capturing the features of the C-E system. While the C-E system defines a small set of inference rules, the CUE approach aims at augmenting the syntax of classic propositional logic and extending the notion of unsatisfiability.

3.1 The CUE Logic

The CUE logic is an extension to classic propositional logic that we have invented in an attempt to capture the useful features of the C-E system proposed by Pearl[7]. We accomplish this by extending syntax and semantics of the logic. The syntactic extensions allow us to keep track of rule and belief types and causes in a way that resembles a truth maintenance system. Apart from providing interpretations to these syntactic extensions, the semantic extensions also define new rules of contradiction.

3.1.1 CUE Beliefs

In classical propositional logic, beliefs about propositions are represented by literals; a positive literal represents the belief that a proposition is true, a negative literal denotes disbelief in a proposition. The only way unsatisfiability can occur is by contradiction, i.e., when a proposition is believed to be both true and false at the

same time.

A belief in the CUE logic is more specific and thus states more than just the believed truth-value of a proposition; it also contains information on what type of inference the belief was attained from. This is known as the *belief type*. A CUE belief may also have a *tag* that contains information on what other beliefs it is related to through the rule that it was attained from.

Belief Types

In the CUE logic there are three types of beliefs, one for each of the letters C, U, and E; *causal*, *undetermined* and *evidential*. Causal and evidential beliefs are also referred to as *C-beliefs* and *E-beliefs*, respectively.

Although the primary reason for the order of the letters in "CUE" was to make it pronounceable, by coincidence this also happens to be the order of strength (from weakest to strongest) of the belief types. E-beliefs are considered more powerful than C-beliefs, because the former can trigger rules of any type while the latter cannot trigger evidential rules.

A belief of undetermined type can be viewed as a potential E-belief and can therefore trigger evidential rules, but may later be demoted to a C-belief as more constraints are applied during reasoning. Once a proposition has become C-believed, it will remain C-believed. Thus, beliefs of undetermined type can be said to be *at least* C-believed.

Tags

Each C-belief and E-belief has a *tag* that can specify other literals that the tagged belief is related to. The tag of a C-belief is used to point out the *known cause* of the belief, whereas the tag of an E-belief points out the *target*, i.e., what the belief is evidence for.

Tags are specified as conjunctions of ordinary literals. Empty tags indicate yet unknown causes. C-beliefs and E-beliefs with empty tags are considered *incomplete*.

3.1.2 CUE Rules

Like the C-E System, the CUE logic keeps track of the type of each rule. Just as with the belief types, there are three types of rules, one for each of the letters C, U and E. This time, however, U stands for *unspecified*. C and E still stand for causal and evidential, respectively.

Causal rules can be triggered by beliefs of any type. When triggered, causal rules produce consequences consisting of C-beliefs only. Evidential rules can be triggered only by E-beliefs. When triggered, evidential rules produce beliefs of undetermined type, i.e. potential E-beliefs that may trigger other evidential rules unless they become demoted to C-beliefs. Rules where causal direction is unspecified act like normal implications; they can be triggered by beliefs of any type and produce beliefs of undetermined type.

3.1.3 Notation

Since the CUE logic extends the syntax of classical propositional logic we also need to extend the notation to be able to express the extra information in a useful way. We propose a notation in which belief types are indicated by adding superscripts to the literals, and tags are added as subscripts.

One thing that has to be kept in mind when reasoning with CUE beliefs is that belief types apply to *literals*, i.e., beliefs or disbeliefs about propositions, not just the propositions themselves, so $\neg p^{\alpha}_{\psi}$ means $(\neg p)^{\alpha}_{\psi}$, *not* $\neg(p^{\alpha}_{\psi})$.

The notation is summarized and explained in table 3.1.

3.1.4 Semantics

In propositional logic, inconsistent expressions are very easy to detect; the only way a contradiction can arise is when the same proposition is believed to be both true and false at the same time. The CUE logic aims not only at disallowing these direct contradictions, but also at preventing implausible conclusions when reasoning with mixtures of causal and evidential rules. Therefore the semantics of the CUE logic is rather complex compared to that of classical propositional logic.

Symbol	Meaning
x^C	x is C-believed, but the cause is unknown
x^C_φ	x is believed to be caused by φ
x^U	x is believed; its belief type is yet undetermined
x^E	x is E-believed, but the target is unknown
x^E_ϕ	x is believed to be evidence for ϕ

Table 3.1: CUE Notation

Implications of CUE Beliefs

CUE beliefs imply that their tags are true, because it would not make sense to believe that e.g. x was caused by y if y is not also believed, or that y is evidence for x unless x is believed aswell. This means that contradictions can occur not only directly between the literals of CUE beliefs, but also between the literal of one CUE belief and the tag of another CUE belief, or between the tags of two different CUE beliefs.

Subsumption

Recognizing redundancies is important for optimization and simplification in any logic based reasoner. Redundancies are be found by searching for subsumptions in conjunctions and disjunctions.

Definition 3 *Let $\Omega(e)$ be the set of worlds for which the expression e is satisfied. Then an expression e_1 is said to* subsume *e_2 (and e_2 is said to be* subsumed *by e_1) iff $\Omega(e_2) \subseteq \Omega(e_1)$. This can be interpreted as e_2 being equal to or more* specific *than e_1.*

Lemma 1 *The set of worlds for which a* conjunction *is satisfied is the* intersection *of the sets of worlds for which each conjunct is satisfied.*

Theorem 1 *If the expression e_1 subsumes another expression e_2, and both e_1 and e_2 are conjuncts in the same conjunction, then e_1 is redundant and can be removed without altering the set of worlds for which the conjunction is satisfied.*

Expression	Result	Condition
$x^{\alpha}_{\varphi} \wedge x^{\alpha}_{\varphi}$	x^{α}_{φ}	
$x^{U} \wedge x^{\alpha}_{\psi}$	x^{α}_{ψ}	
$x^{\alpha} \wedge x^{\alpha}_{\psi}$	x^{α}_{ψ}	
$x^{E}_{\phi} \wedge x^{C}_{\varphi}$	p^{C}_{φ}	ϕ subsumes φ

Table 3.2: Simplifiable conjunctions of CUE beliefs

Lemma 2 *The set of worlds for which a* disjunction *is satisfied is the* union *of the sets of worlds for which each disjunct is satisfied.*

Theorem 2 *If the expression e_1 subsumes another expression e_2, and both e_1 and e_2 are disjuncts in the same disjunction, then e_2 is redundant and can be removed without altering the set of worlds for which the disjunction is satisfied.*

In propositional logic, single belief statements (literals) can subsume each other if and only if they are equal. In the CUE logic, however, one single CUE belief can be subsume another belief even if they are not equal. For instance, x^U (x is believed, but belief type is undetermined) subsumes x^C (x is believed to be caused by something) since x^C is more specific.

Table 3.2 summarizes some of the different kinds of simplifications that can be made for conjunctions of single CUE beliefs.

Unsatisfiability

Apart from the obvious case in which the same proposition is believed to be both true and false at the same time, there are three other types of unsatisfiabiliy that can occur in the CUE logic:

- Since C-beliefs and E-beliefs imply that their tags are true, the literal of one CUE belief might contradict the literals in the tag of another belief, or the tags of two different beliefs might contradict each other.

Expression	Condition
$x^*_* \wedge \neg x^*_*$	always unsatisfiable
$x^\alpha_\psi \wedge y^*_*$	y contradicts ψ
$x^\alpha_\varphi \wedge y^\beta_\psi$	ψ contradicts φ
$x^C_\varphi \wedge x^E_\phi$	ϕ does not subsume φ
$x^C_\varphi \wedge x^C_\psi$	$\varphi \neq \psi$

Table 3.3: Unsatisfiability in the CUE logic

- The basic idea behind the C-E System and the CUE logic is that a belief x cannot be evidence for y if it was caused by something other than y. Therefore, conjunctions containing a C-belief and an E-belief about the same proposition are unsatisfiable unless the tag of the E-belief at least subsumes the tag of the C-belief.

- Another goal is to accept only the simplest explanations for things. Since it is unlikely for something to be caused by two independent causes at the same time, conjunctions containing two C-beliefs about the same literal are considered unsatisfiable unless their tags (sets of causes) are equal.

All types of contradictions explained above are summarized in table 3.3.

Negating CUE Beliefs

Converting rules to clauses requires the beliefs in the antecedent to be negated. For undetermined CUE beliefs, it is sufficient to simply invert the sign. Negating causal and evidential beliefs is not that simple. In fact, negating one E-belief would result in a disjunction of two new CUE beliefs. Intuitively, something that is "not E-believed" can be either

a) completely disbelieved, or

b) believed to be caused by some unknown cause.

Hence, the negation of an E-belief is

$$\neg(x_{\phi}^{E}) \quad \equiv \quad \neg x^{U} \vee x^{C} \tag{3.1}$$

The negation of a C-belief is not as intuitive, but could probably be done in a similar manner. Thus far it has never been necessary, however.

3.1.5 Rules in the CUE Logic

In classic propositional logic, rules are expressed using the implication connective, and can easily be converted into clauses through a trivial algorithm. In the CUE logic there are three types of rules; in addition to normal implication rules one can express causal and evidential rules.

After carefully analyzing the semantics behind each rule type, we can easily figure out how to properly convert CUE rules into clauses:

Definition 4 *A normal implication $p \rightarrow q$ states that if p is believed, then q is believed. Hence, p is disbelieved or q is believed.*

Definition 5 *A causal rule $p \rightarrow_C q$ states that if p is believed in any way, then q is believed to be caused by p. Hence, p is disbelieved or q is believed to be caused by p.*

Definition 6 *An evidential rule $q \rightarrow_E p$ states that if q can be evidence for p, then p is believed. Hence, q cannot be evidence for p or p is believed, i.e., q is either disbelieved or caused by something (other than p), or p is believed but undetermined.*

Table 3.4 shows a summary of how the different types of CUE rules are expressed and converted into clauses.

3.1.6 SAT Solving in the CUE Logic

Testing a set of expressions for satisfiability can be carried out by building their conjunciton as a large DNF expression. The advantage of this method is that once the DNF expression has been built and simplified, one can immediately see what is

Type of rule	**CUE notation**	**CUE clause**
Normal implication	$x \rightarrow y$	$\neg x^U \vee y^U$
Causal rule	$x \rightarrow_C y$	$\neg x^U \vee y_p^C$
Evidential rule	$x \rightarrow_E y$	$\neg x^U \vee x^C \vee y^U$

Table 3.4: Rules in the CUE logic

required for a world to satisfy all the expressions. That is, each disjunct in the DNF will define a set of constraints, and each world that satisfies any such set of constraints will satisfy all the original expressions.

This method can be used with the CUE logic aswell, but it requires postprocessing of the DNF expression to purge it from incomplete disjunts, i.e. disjuncts containing incomplete CUE beliefs. The removal of incomplete constraint sets ensures that only causes within the system are allowed. If incomplete constraints were allowed, evidential rules would lose their significance, because every observation could then be explained by some unknown cause that was never mentioned within the rules of the system.

Chapter 4

Experiments

By using the graphical querying interface of the Z-log applet we can experiment with the CUE logic and analyze what effects it has when it is used together with System-Z. We start out by showing a couple of small examples that are tailored to demonstrate certain features, and then we will conclude the chapter with a larger example.

4.1 Wet Grass Example

In this simple example we will begin by demonstrating the main differences between classic propositional logic and the CUE logic. Suppose we have the following propositions:

$Sprinkler$ = "The sprinkler was on last night"

$WetGrass$ = "The grass is wet"

$Rain$ = "It rained last night"

and the following rules:

$r_1 : Sprinkler \rightarrow_C WetGrass$

$r_2 : WetGrass \rightarrow_E Rain$

4.1.1 Blocking of Evidential Rules

If we use propositional logic and set the input *Sprinkler* to true, leaving the other two propositions undecided, Z-log will draw the correct conclusion that the grass is wet, but it will also be quite happy to announce that it must have been raining last night. This clearly illustrates the problems that can arise when causal directionality is ignored.

Now if we try the same thing using the CUE logic (by selecting "Causal" rule style on the KB edit page) instead, Z-log will still conclude that the grass is wet, but it will no longer be able to decide whether it has been raining or not. Thus the chaining of the rules has been successfully blocked.

4.1.2 Diagnostic Inferences

If we set $WetGass$ true and leave everything else undecided, the propositional logic will draw the conclusion that is has been raining, while the CUE logic will be unable to conclude anything (apart from what is given as input). The reason for this is that the CUE logic is very careful about drawing conclusions from evidence; r_2 applies only when there is no other explanation for $WetGrass$, and *Sprinkler* is a possible cause for $WetGrass$, since it has not been determined to be false.

4.1.3 Preferring One Explanation Over Another

Thus far the tested knowledge base has been free from overrides, i.e. all rules have been given the same rank in the partition. This will change as we examine how a default can be used to set a preferred explanation. By adding the rule

$$r_3 : \top \rightarrow \neg Sprinkler$$

we tell the system that the sprinkler is usually not on ($\top$ means"true"). What the resulting partition looks like depends on whether we use Z or $\hat{Z}$ semantics. Table 4.1 shows both partitions.

Now when we return to the querying interface we see that *Sprinkler* is false by default. If we set $WetGrass$ true, both Z and $\hat{Z}$ will conclude that *Rain* is true.

Rank	Z partition	$\hat{\text{Z}}$ partition
2	$r_1 : Sprinkler \rightarrow_C WetGrass$	$r_1 : Sprinkler \rightarrow_C WetGrass$ $r_2 : WetGrass \rightarrow_E Rain$
1	$r_2 : WetGrass \rightarrow_E Rain$ $r_3 : \top \rightarrow \neg Sprinkler$	$r_3 : \top \rightarrow \neg Sprinkler$

Table 4.1: Z and $\hat{\text{Z}}$ partitions

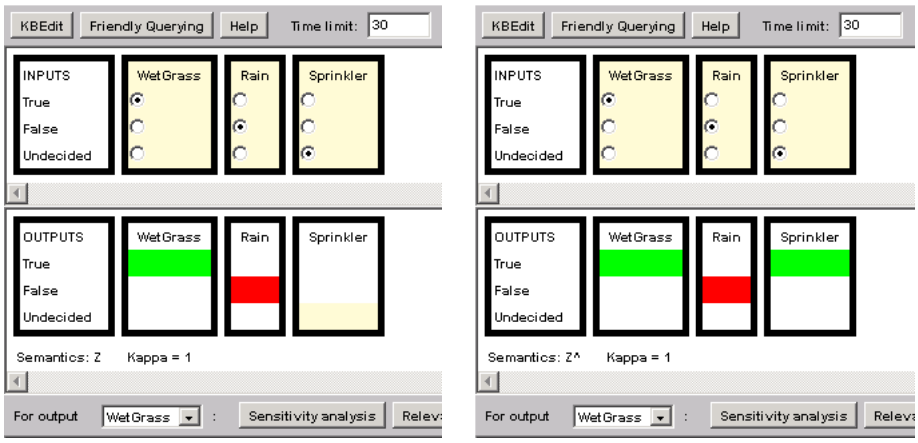

Figure 4.1: Screen shots showing difference in behavior between Z and $\hat{Z}$.

However, if we also set *Rain* false, $\hat{Z}$ will come to the plausible conclusion that *Sprinkler* is true, while Z gets confused and cannot say anything about *Sprinkler*. Figure 4.1 shows screenshots of this. Notice that Z-log reports the level of surprise to be 1.

The reason that Z becomes confused is related to the fact that it treats the violation of non-overridden defaults with the same incredulity as the violation of overridden ones, as described in section 2.1.3. This will be further analyzed in section 5.2.

4.2 Automobile Example

Now we consider a knowledge base for a different domain:

$TurnKey$ = "The ignition key is turned"

$CarStarts$ = "The car starts"

$BatteryDead$ = "The battery is dead"

$LightsOn$ = "The lights have been left on for ten hours"

r_1: $TurnKey \rightarrow_C CarStarts$

r_2: $BatteryDead \rightarrow_C \neg CarStarts$

r_3: $LightsOn \rightarrow_C BatteryDead$

r_4: $BatteryDead \rightarrow_E LightsOn$

This example is due to Goldszmidt and Pearl[2] and is analogous to the famous Yale Shooting Problem[1]. In this knowledge base, every rule is tolerated by all rules, so both Z and $\hat{Z}$ place all rules at the lowest rank.

4.2.1 Non-intuitive Conclusions

If we set $TurnKey$ true as the only input information, the system concludes $CarStarts$, $\neg BatteryDead$ and $\neg LightsOn$. The conclusion that the car starts comes directly from rule r_1 and is therefore fully acceptable. The conclusion that the battery is OK, from which it follows that the lights have not been left on, however, is not intuitive: If we know only that the ignition key has been turned we should not be able to draw conclusions about the state of the battery. It gets even worse if we give $BatteryDead$ as the sole input. As the screen shot in figure 4.2 shows, the system then concludes that the ignition key was not turned. This issue will be discussed in more detail in section 5.3.1.

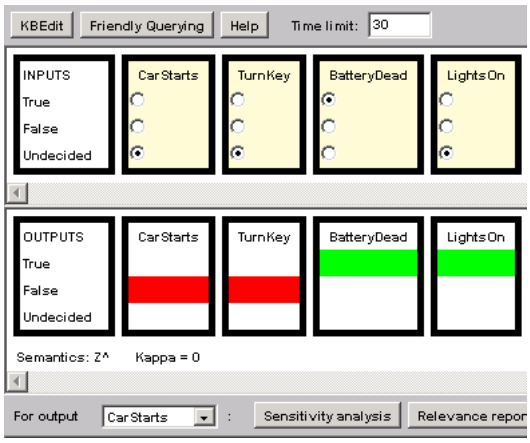

Figure 4.2: Conclusions drawn from the battery being dead

4.2.2 Overrides

If we set both $TurnKey$ and $BatteryDead$ to true, there will be a contradiction between r_1 and r_2, and the system will not be able to draw any conclusions. To make r_2 override r_1 we can add a "helper" rule:

$$hr_1 : TurnKey \wedge BatteryDead \rightarrow \neg CarStarts$$

Note that this rule is a normal implication, i.e. it is neither causal or evidential. The reason that we did not instead define r_2 as

$$r_2 : TurnKey \wedge BatteryDead \rightarrow_C \neg CarStarts$$

is that turning the key has no part in the cause for the car not starting. As we shall show in section 5.1, for $\hat{Z}$ the helper rule affects only how the partition is built[1].

When we rebuild the ranked partition and tell Z-log $TurnKey \wedge BatteryDead$, it reponds by concluding $\neg CarStarts$ and reports that the level of surprise is 1. If the new partition was built using $\hat{Z}$, Z-log will also conclude that the lights have probably been left on.

[1]That is, deleting the helper rule from the $\hat{Z}$ partition will not change the corresponding world ranking.

4.3 Larger car example

In this example we have the following propositions and rules:

$TurnKey$ = "The ignition key has been turned"

$ERuns$ = "The engine runs"

$ETurns$ = "The engine turns (when it is about to start)"

$RadioWork$ = "The radio works"

$BatteryOK$ = "The battery is OK"

Gas = "There is gas in the tank"

$StarterOK$ = "The starter is OK"

$SPlugOK$ = "The spark plug is OK"

$r_1 : \top \rightarrow BatteryOK$

$r_2 : \top \rightarrow Gas$

$r_3 : \top \rightarrow StarterOK$

$r_4 : \top \rightarrow SPlugOK$

$r_5 : ETurns \wedge Gas \wedge SPlugOK \rightarrow_C ERuns$

$r_6 : \neg ETurns \rightarrow_C \neg ERuns$

$r_7 : \neg Gas \rightarrow_C \neg ERuns$

$r_8 : \neg SPlugOK \rightarrow_C \neg ERuns$

$r_9 : TurnKey \wedge BatteryOK \wedge StarterOK \rightarrow_C ETurns$

$r_{10} : \neg TurnKey \rightarrow_C \neg ETurns$

$r_{11} : \neg BatteryOK \rightarrow_C \neg ETurns$

$r_{12} : \neg StarterOK \rightarrow_C \neg ETurns$

$r_{13} : BatteryOK \rightarrow_C RadioWorks$

$r_{14} : RadioWorks \rightarrow_E BatteryOK$

After compiling this knowledge base using $\hat{Z}$ semantics and causal reasoning, we can use the querying interface of Z-log to diagnose car problems. By default, because of rules r_1 thru r_4, the system will assume that the battery, starter and spark plug are OK and that there is gas in the tank. It will conclude $RadioWorks$ from rule r_{13}.

When we tell the system $\neg ERuns$, i.e. that the engine isn't running, the system will respond by adding $\neg TurnKey$ (the ignition key has not been turned) and the consequence $\neg ETurns$ to the outputs. If we also claim that we did turn the key, the level of surprise in the system will increase and all conclusions will be retracted; more information is needed before the system can make any conclusions.

Among the undecided outputs we find $ETurns$, so we decide to check if we can hear the engine turning when we turn the ignition key. If we tell the system that we did not hear any sound, by adding $\neg ETurns$ to the inputs, the system will add Gas and $SPlugOK$ to the outputs as figure 4.3 shows. The reason is that $\neg ETurns$ is identified as *the* cause for $\neg ERuns$ (remember that the CUE logic allows only one cause for each belief). Now we still have $RadioWorks$ among the undecided outputs so we check the radio next. If we add the fact that the radio is working, the system will conclude that the battery is OK and therefore $\neg StarterOK$ must be the cause of $\neg ETurns$.

Had we heard the engine turning and thus told the system $ETurns$, it would have concluded that the battery, radio and starter are OK, and left Gas and $SPlugOK$ as undecided possible causes for $\neg ERuns$. Figure 4.4 shows what this would have looked like.

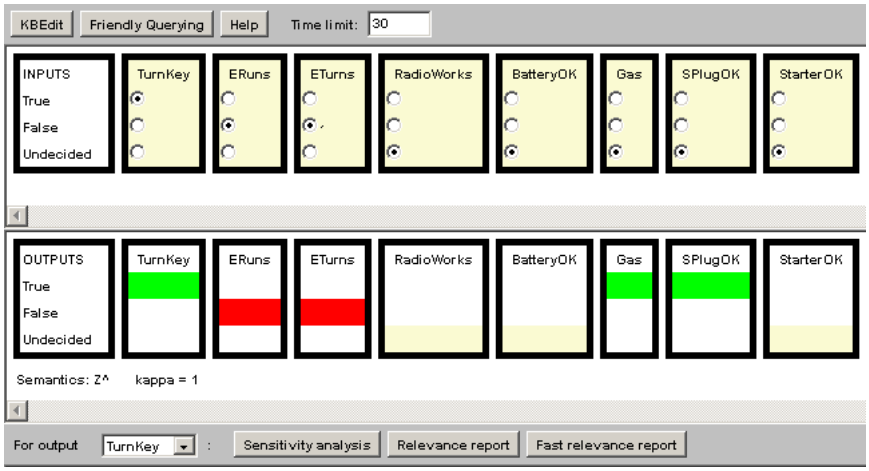

Figure 4.3: Car troubleshooting: Engine not turning

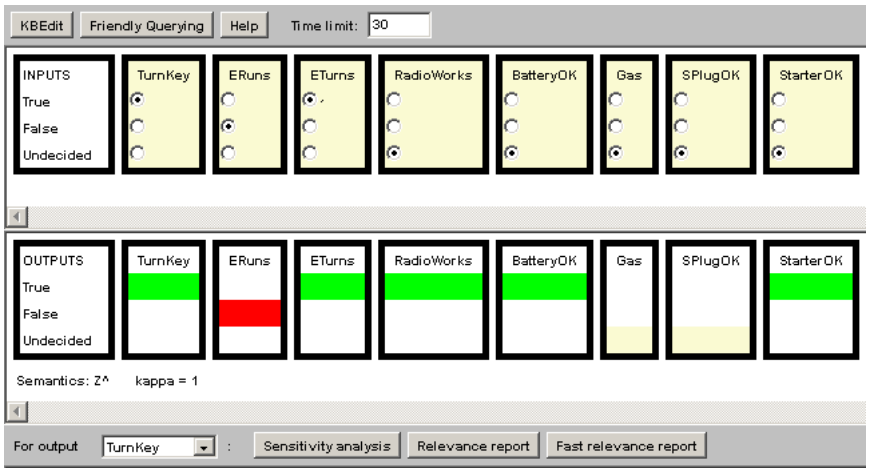

Figure 4.4: Car troubleshooting: Engine turns

Chapter 5

Discussion

5.1 Overriding Causal Rules

As mentioned in section 4.2.2, helper rules can be used to specify exceptions to default causal rules. Once the $\hat{Z}$ partition has been built, the helper rules are no longer significant. This can be demonstrated by using the command-line version of Z-log. First we save the partition to a file, then we edit the partition file, remove the helper rule and save the result in a new file. Then we start the command-line version of Z-log in debug mode and read in the partition files and compare the DNF expressions in both partitions, level by level.

If we take the knowledge base from section 4.2.2, including the helper rule, and build a $\hat{Z}$ partition, we will get the DNF expressions listed in table 5.1. If we build a Z partition instead, the helper rule will not be insignificant after the partition has been built. Tables 5.2 and 5.3 show different DNF expressions in those partitions. What makes the helper rule significant in the Z partition is that the Z partition building algorithm does not place the helper rule (hr_1) at the same rank as the rule it is helping (r_2).

Rank	DNF
2	$(\neg BatteryDead \wedge \neg LightsOn) \vee$ $(\neg CarStarts^{C}_{BatteryDead} \wedge BatteryDead^{C} \wedge \neg LightsOn) \vee$ $(\neg CarStarts^{C}_{BatteryDead} \wedge BatteryDead^{C}_{LightsOn})$
1	$(\neg BatteryDead \wedge \neg LightsOn \wedge \neg TurnKey) \vee$ $(\neg BatteryDead \wedge \neg LightsOn \wedge CarStarts^{C}_{TurnKey}) \vee$ $(\neg CarStarts^{C}_{BatteryDead} \wedge BatteryDead^{C} \wedge \neg LightsOn \wedge \neg TurnKey) \vee$ $(\neg CaStarts^{C}_{BatteryDead} \wedge BatteryDead^{C}_{LightsOn} \wedge \neg TurnKey)$

Table 5.1: DNF expressions in $\hat{Z}$ partition, with or without helper rule

Rank	DNF
2	$(\neg TurnKey) \vee$ $(\neg BatteryDead) \vee$ $(\neg CarStarts)$
1	$(\neg TurnKey \wedge \neg BatteryDead \wedge \neg LightsOn) \vee$ $(\neg TurnKey \wedge \neg CarStarts^{C}_{BatteryDead} \wedge BatteryDead^{C} \wedge \neg LightsOn) \vee$ $(\neg TurnKey \wedge \neg CarStarts^{C}_{BatteryDead} \wedge BatteryDead^{C}_{LightsOn}) \vee$ $(CarStarts^{C}_{TurnKey} \wedge \neg BatteryDead \wedge \neg LightsOn)$

Table 5.2: DNF expressions in Z partition, with helper rule

Rank	DNF
1	$(\neg TurnKey \wedge \neg BatteryDead \wedge \neg LightsOn) \vee$ $(\neg TurnKey \wedge \neg CarStarts^{C}_{BatteryDead} \wedge BatteryDead^{C} \wedge \neg LightsOn) \vee$ $(\neg TurnKey \wedge \neg CarStarts^{C}_{BatteryDead} \wedge BatteryDead^{C}_{LightsOn}) \vee$ $(CarStarts^{C}_{TurnKey} \wedge \neg BatteryDead \wedge \neg LightsOn)$

Table 5.3: DNF expressions in Z partition, without helper rule

5.2 Z or $\hat{Z}$?

Once a default has been violated, all defaults that cause equal or less surprise when violated (i.e., all defaults at the same and lower ranks) lose their significance, because the level of surprise never decreases as more input facts are added.

Since Z puts all non-overridden defaults at the lowest rank, they will expire as soon as the first default is violated, no matter what rank the violated default is at. For this reason, it is recommended that $\hat{Z}$ be used.

5.3 Limitations

5.3.1 Lack of Modularity

As the example in section 4.2.1 showed, the CUE logic does not solve the problem with non-intuitive conclusions. The problem is recognized by Goldszmidt and Pearl[2] as a problem that plagues most approaches based on conditional interpretation of causal rules.

The intention of rules r_1, r_2 and rh_1 in the knowledge base of our example, is merely to describe how the car responds to various combinations of key/battery conditions, and are not meant to imply any dependencies among those conditions. In general, if our understanding of the relationships between the ignition key and the battery is encoded in some knowledge base Δ, we certainly do not expect that adding rules r_1, r_2 and rh_1 to Δ would modify those relationships. If the state of the battery is assumed to be independent from the ignition key prior to specifying these rules, then it ought to remain independent from the ignition key after the rules have been added. This is a prevailing general pattern of causal reasoning: contemplating possible developments of future events should not affect our belief in past and present events [2]. This pattern is called *modularity*, and it follow from the Markov property of causal organization, which states that the occurrence of each event is independent of all past events, once we determine the direct causes of the event [7].

5.4 Future Work

5.4.1 Removing the Need for Helper Rules

Helper rules would not be necessary if causal and evidential rules could be specified more accurately, so that literals that are part of a cause or an evidence could be distinguished from literals that merely specify the context for which the rule holds. This would be rather easy to implement, once a suitable syntax has been found.

5.4.2 Restoring Horn Optimizations

The horn and q-horn optimizations incorporated in the first version of Z-log[3] have not been implemented in the current version of Z-log. No investigation has been made as to if, and how, these optimizations can be used with logics other than classic propositional logic.

5.4.3 Implementing Other Experiments

As noted in appendix A, the new version of Z-log provides a good foundation for experimenting with new semantics, logics or SAT solvers.

5.5 Acknowledgements

I want to thank my instructor Michael Minock for giving great advice in the theoretical parts and for his assistance in the process of writing and presenting this thesis. I would also like to thank my friends, especially Anna Hopfgarten, and my family for their support and patience.

Appendix A

Implementation

This chapter describes the redesign and implementation of Z-log version 2, including the CUE logic. The first version of the Z-log application suffered from poor design, so before the CUE logic could be appropriately applied, the Z-log application had to be redesigned.

A.1 History

Although it had been created using a powerful software design tool, the design in the first version of Z-log was rather poor and did not allow for new features to be implemented easily. Key features of the Java language, such as inheritance and interfaces had not been taken advantage of. For instance, the obvious class–subclass relationship between the Z and $\hat{Z}$ semantics had not been recognized; instead the entire System-Z and $\hat{Z}$ implementation was contained in one large class called ZEngine.

These design problems could in part have been related to the fact that the System-Z prototype was written in LISP, a language that is very different from Java.

A.2 Developing Version 2 of Z-log

Since the code for Z-log version 1 was rather messy (in part due to the poor design) and hard to read, a lot of time had to be spent on tidying up the existing code and

figuring out how the system really worked before a new design could be created. Once the new design was ready, some parts of the code were entirely rewritten, while other parts were reinserted by "cut and paste". The result was a new program that on the outside looked and worked just like the first version. Once the new implementation was working properly, development of some new features could begin.

A.3 The New Design

Version 2 of Z-log has a more intuitive class design that takes much more advantage of the features of object oriented design than the first version did. One of the primary deisgn goals with version 2 was to make the system more usable to developers who whish to experiment and/or add new features to it.

A.3.1 System Overview

The graphs in figure A.1 show the classes and interfaces in the different parts of the system. Interfaces are displayed as plain text, while classes are represented by boxes. Solid arrows illustrate inheritance or implementation, and dashed arrows represent collaboration. For instance, StandardOutput implements OutputInterface, and is used by Zlog.

To prevent figure A.1 from becoming too cluttered, dashed arrows between different parts of the system have been left out. Instead, the box in the lower right corner gives a rough picture of how the different parts interact.

Logics

Literals in propositional logic can interact with each other by causing contradictions in conjunctions or subsumption in disjunctions. In an object oriented implementation of propositional logic, checking for contradiction and subsumption can be performed by the literals themselves (i.e., in the Literal class).

Since the CUE logic differs from ordinary propositional logic only in how contradictions and subsumptions arise, the CUE logic was implemented by simply extending

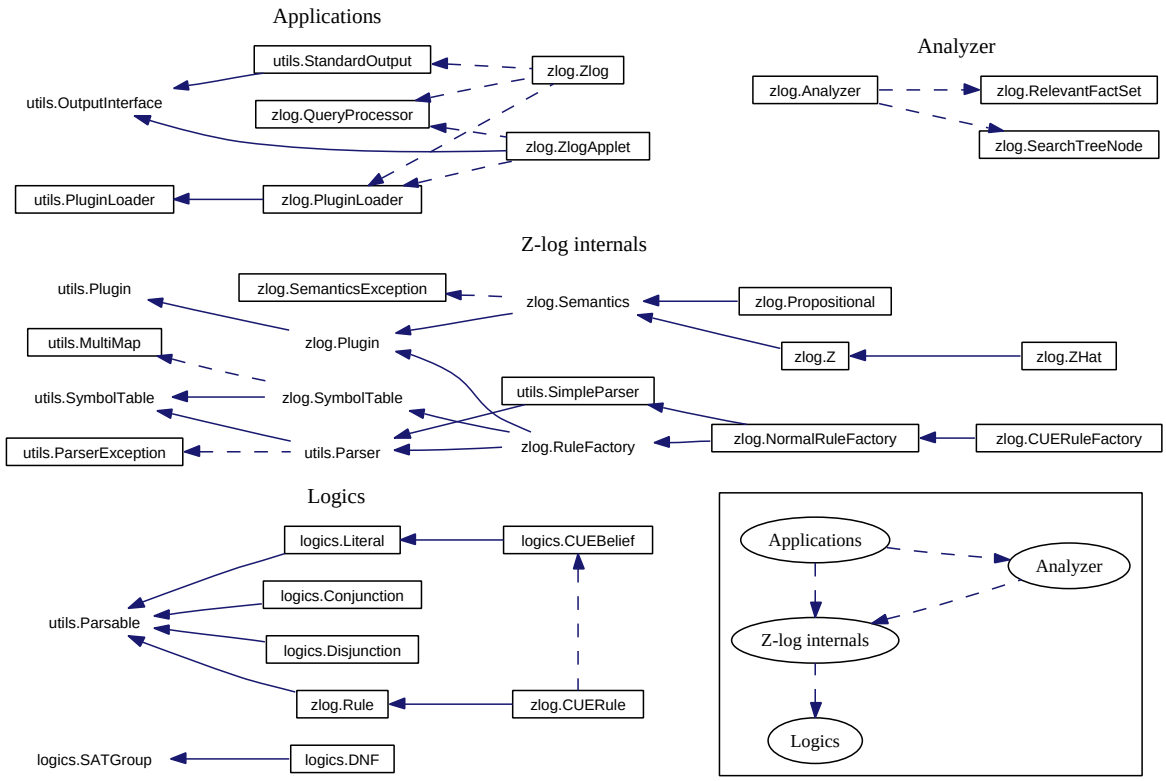

Figure A.1: Classes and System Overview

the Literal class to create a CUEBelief class. Expressions are realized by the classes Conjunction (conjunctions of Literals), Disjunction (disjunctions of Literals) and DNF (Disjunctive Normal Form expressions).

SATGroup is an interface for collections of expressions that can be tested for satisfiability, and it is implemented by DNF since DNF expressions can be used for this. Rules (implications) are realized by the Rule class, which is extended by the CUERule class to realize CUE rules. Most of the logic classes implement the Parsable interface, which means that they take care of their own parsing (with a little help from a Parser).

Z-log Internals

The two most important interfaces in the Z-log internals part are Semantics and RuleFactory. The both extend zlog.Plugin, which extends the generic utils.Plugin interface. Z-log version 2 allows any class that implements Semantics to be loaded at run-time and used as a semantics. The system comes with the implementations Propositional (simple semantics without default reasoning), Z (System-Z) and ZHat (System-$\hat{Z}$). Semantics are supposed to throw SemanticsExceptions if something goes wrong.

Rule factories (implementations of RuleFactory) are used to parse KB definitions and create rules to be used with the semantics. NormalRuleFactory parses standard definitions and creates ordinary Rules, while CUERuleFactory was invented to parse CUE rule definitions and create CUERules that obey the CUE logics. The RuleFactory interface extends interfaces zlog.SymbolTable and utils.Parser. zlog.SymbolTable is an extension of the generic utils.SymbolTable interface, and uses MultiMap to allow symbols to have multiple "tags". Parser implementations are supposed to throw ParserExceptions if anything goes wrong. NormalRuleFactory is implemented as an extension to SimpleParser.

Analyzer

The Analyzer class provides methods for "black box" analysis using a knowledge base and a semantics. It uses the classes SearchTreeNode and RelevantFactSet. The code in these classes have pretty much been left as it was in Z-log version 1.

Applications

Z-log comes with two rather different user interfaces. The first one is Zlog, a command-line interface that outputs its results to standard output using the StandardOutput implementation of OutputInterface. The other user interface is ZlogApplet, an applet that implements OutputInterface by itself. Both of these user interfaces use zlog.PluginLoader, which is an extension of the generic utils.PluginLoader, to load semantics and rule factory plugins. They also use QueryProcessor to parse and execute query definitions.

A.4 Extending the System

To add and use your own semantics with Z-log version 2, all you need to do is to create a class that implements the Semantics interface and then load that class into the application or applet using command-line arguments or applet parameters. To create a new definition interpreter, you just need to create a class that implements RuleFactory. Plugins of these kinds can be created without modifying or recompiling any existing part of the system. With little effort the system could even be modified to let the user choose any SAT solver implementation, as long as it implements the SATGroup interface.

Appendix B

Glossary

- **Causal** – A causal rule predicts the effect of an observed cause.
- **Clause** – A rule or constraint expressed as a disjunction.
- **Conjunct** – One of the components of a conjunction.
- **Conjunction** – Logical AND expression.
- **Disjunct** – One of the components of a disjunction.
- **Disjunction** – Logical OR expression.
- **DNF** – Disjunctive Normal Form. An expression that is in DNF has the form of a disjunction of conjunctions of literals.
- **Exceptional subclass** – A subclass that does not inherit all the attributes of its base class. Penguin is an example of an exceptional subclass of bird, since penguins are birds but do not fly.
- **Evidential** – An evidential rules explains the cause for observed evidence.
- **Hard rule** – A rule that must not be violated.
- **KB** – Knowledge Base, set of rules and facts for a domain.
- **Literal** – A propositional symbol with or without preceding negation sign.

- **Pinned defaults** – In System-$\hat{Z}$, defaults that are pinned cannot be promoted to higher ranks in the partition.
- **Proposition** – A statement that is either true or false.
- **SAT** – Satisfiability. An expression is satisfiable iff it is possible to assign truth-values to the propositional variables so that the expression is true.
- **Default rules** – A rule that can be violated with surprise.
- **Transitivity** – Chaining of rules, e.g. if $p \rightarrow q$ and $q \rightarrow r$, then $p \rightarrow r$.
- **World** – An assignment of truth-values to a set of propositional variables.
- **World ranking** – Integer denoting the level of surprise associated with a world.

Bibliography

[1] H. Geffner. *Default reasoning: causal and conditional theories.* MIT Press., 1992.

[2] Goldszmidt and Judea Pearl. Qualitative probabilities for default reasoning, belief revision, and causal modeling. *Artificial Intelligence, 84(1-2):57–112*, 1996.

[3] Hansi Kraus. Z-log: A practical implementation of System-Z (and $\hat{Z}$). Master's thesis, University of Umeå, Sweden, 2002.

[4] J. McCarthy. Applications of circumscription to formalizing commonsense knowledge. *Artificial Intelligence, 28:89–116.*, 1986.

[5] Michael Minock and Hansi Kraus. Z-log: A system-Z (and $\hat{Z}$) 'system'. Technical Report 02.05, The Univeristy of Umea, Umea, Sweden, May 2002.

[6] J. Pearl. Embracing causality in formal reasoning. *Artificial Intelligence, 35:259–271.*, 1988.

[7] J Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann, San Mateo, CA., 1988.

[8] J. Pearl. A natural ordering of defaults with tractable applications to default reasoning. *Proceedings of the 3rd Conference on Theoretical Aspects of Reasoning about Knowledge, pages 121–135.*, 1990.

[9] R. Reiter. A logic for default reasoning. *Artificial Intelligence, 26:27–47.*, 1988.

[10] S. Russel and P. Norvig. *Artificial Intelligence: A Modern Approach*, chapter 15. Probabilistic Reasoning Systems. Prentice Hall, 1995.

[11] E. H. Shortliffe. *Computer-based medical consultation: MYCIN.* New York: Elsevier, 1976.

[12] Johan Winberg. 'what if?', 'are you sure?' and 'why?' over default reasoning systems. Master's thesis, University of Umeå, Sweden, 2003.