Using Abduction to Evolve Inconsistent Requirements Specifications

Bashar Nuseibeh **Alessandra Russo**

Department of Computing,
Imperial College of Science, Technology and Medicine
180, Queen's Gate, London SW7 2BZ, U.K.
Email: {ban, ar3}@doc.ic.ac.uk

Abstract

Requirements specifications are often inconsistent. Inconsistencies may arise because multiple conflicting requirements are embodied in these specifications, or because the specifications themselves are in a transient stage of evolutionary development. In this paper we argue that such inconsistencies, rather than being undesirable, are actually useful drivers for changing the requirements specifications in which they arise. We present a formal technique to reason about inconsistency handling changes. Our technique is an adaptation of logical abduction – adapted to generate changes that address some specification inconsistencies, while leaving others. We represent our specifications in quasi-classical (QC) logic – an adaptation of classical logic that allows continued reasoning in the presence of inconsistency. The paper develops a sound algorithm for automating our abductive reasoning technique and presents illustrative examples drawn from a library system case study.

1 Introduction

A key problem in large software engineering projects is the management of frequently changing requirements. Requirements changes have a particularly significant impact on the consistency of specifications. Changes may introduce inconsistencies; and conversely, requirements changes may be necessary to handle existing inconsistencies. Therefore, the ability to handle inconsistent requirements is crucial to the successful development of requirements specifications.

Researchers have developed a variety of techniques for analysing and managing the impact of changes on systems and their specifications [Bohner & Arnold 96]. However, the starting point of much of this work is that a consistent artefact exists, which will be changed while preserving its consistency. The motivation is based on the intuitive assumption that inconsistencies are undesirable, given the difficulty of deriving useful information from inconsistent specifications. Consequently, the techniques developed are based on rigorous consistency checking and analysis, in an attempt to eradicate inconsistencies as soon as – or soon after – they are detected.

However, in practice, inconsistency is inevitable in real large-scale specifications [Schwanke & Kaiser

88; Balzer 91; Cugola *et al.* 96]. Living with inconsistency during evolutionary development is a fact of life. Therefore, *inconsistency handling* mechanisms are needed to support incremental evolution of specifications. By this we mean identifying changes that address some specification inconsistencies, while leaving others [Finkelstein *et al*. 94].

In this paper, we provide a formal technique for handling inconsistencies that (a) allows such an incremental evolution of (inconsistent) requirement specifications and (b) can be implemented by exploiting existing tools for handling theory change [Kakas & Mancarella 90, Selmen & Levesque 90]. Requirements specifications are, in our approach, partial specifications (possibly developed by different users) related to each other by means of pre-defined "consistency rules" [Nuseibeh *et al.* 94]. Each partial specification may or may not include (logical) inconsistencies. The overall specification is, in our approach, defined to be "inconsistent" when at least one of the pre-defined rules is violated. This violation can be detected by checking if the specification satisfies the negation of the rule. Defining inconsistency in this way allows more flexibility to express a variety of constraints on specifications. It subsumes the notion of logical inconsistency, which is not always the most convenient way of representing the violation of constraints in real-life specifications.

The inconsistency handling technique described in this paper is based on a specific type of formal reasoning, called *abduction* [Kakas *et al.* 98]. If a particular consistency rule is violated, our proposed *abductive reasoning* technique identifies (evolutionary) changes to perform on the specification, such that a particular consistency rule is no longer violated. The partial specification, as well as the consistency rules, is assumed to be expressed in *quasi-classical* (QC) logic [Hunter & Nuseibeh 97; Hunter & Nuseibeh 98] – an adaptation of classical logic that allows reasoning in the presence of inconsistencies without trivialisation[1].

To provide a route towards implementing our technique, we have also developed an algorithm that translates specifications expressed in QC logic into logic programs. This enables us to deploy existing tools for abductive logic programming [Kakas & Mourlas 97]. As shown in the paper, the results given by such tools can be translated back into the QC representation of the specifications, thus providing an automated way of generating changes to resolve inconsistencies.

To summarise, this paper uses abduction to support incremental evolution of inconsistent requirements specifications. The novelty of the work is in adapting existing results on abduction for consistent specifications and applying them in the more realistic setting of inconsistent specifications (section 3). The notions of inconsistency and abduction are formally described in section 2. The abductive technique developed (section 3) is tailored to make use of existing automated tools, and the paper provides a sound

[1] Trivialisation is the inference of arbitrary information from an inconsistent specification.

translation mechanism to logic programs in order to achieve this (section 4). A complete illustrative example (section 5) and a discussion of related work and open research issues (sections 6 and 7, respectively) conclude the paper.

2 Inconsistency and Abduction

In this section, we provide an overview of our notions of inconsistency and abduction, which we use in the remainder of this paper as the basis of our approach.

2.1 Inconsistency

We base our notion of inconsistency on the violation of consistency rules. We treat such consistency rules as "properties" that should be satisfied by a given requirement specification. Within this context, specifications are considered to be *inconsistent* if they satisfy the negation of some of these rules (e.g., because of over-specification). More formally:

> **Inconsistency.** Given a specification S and a set $\Re = \{R_1, R_2, ... R_n\}$ of consistency rules, S is inconsistent if there exists at least one rule R_i, for some $1 \leq i \leq n$, such that $S \vdash \neg R_i$, where $\vdash$ denotes any underlying formal reasoning mechanism. For any violated consistency rule R_i, we say that the specification S is R_i-inconsistent.

Inconsistency handling is the process of making changes that result in particular consistency rules no longer being violated; that is, $S' \nvdash \neg R_i$, where S' is the new (evolved) specification. Since the changes do not guarantee the new specification to be consistent (i.e. some other consistency rules might still continue to be violated), in this paper we represent our specifications in QC logic, which allows continued non-trivial reasoning in the presence of inconsistency.

2.2 Abduction

Abduction is one of the three fundamental modes of reasoning, the others being deduction and induction. The most popular formalisation of abduction in AI defines it as the search for a set of hypotheses (an "explanation") that, combined with a given theory, achieves some given goals (an "observation") without causing contradictions [Kakas *et al.* 98]. This type of inference procedure has been shown to be suitable for addressing different kinds of problems in AI, such as diagnosis [Console *et al.* 96], planning [Esghsi 98], and database updates [Console *et al.* 94; Kakas & Mancarella 90; Inoue & Sakama 95]. Recent work has also shown its application in knowledge-based software engineering [Menzies 96].

Using abduction for theory change (e.g., database updates), the observation is a request for a particular change (update), and abduction is the process of identifying the changes to be made to a given theory so that the new theory satisfies the request [Kakas & Mancarella 90; Inoue & Sakama 95]. A request can be, for example, a requirement for information to be a consequence of a given theory, or for it to be no longer inferred from a given theory. In the first case, abduction provides an *explanation* of the given information, whereas in the second case, it provides an *anti-explanation* [Inoue & Sakama 95]. Formally:

Abduction. Let L be a given logical language and let S be a theory written in L. An *abductive framework* is a pair $\langle S, Ab \rangle$ where Ab is a set of abducibles, also written in L. Let R be a request[2]. A pair (Δ^+, Δ^-), where both Δ^+ and Δ^- consist of ground instances of elements from Ab, is an *explanation* of R in the abductive framework $\langle S, Ab \rangle$ if

- $(S \cup \Delta^+) \setminus \Delta^- \models R$
- $(S \cup \Delta^+) \setminus \Delta^-$ is consistent

where the symbol \ denotes the standard operation of subtraction between sets.
A pair (Δ^+, Δ^-) is an *anti-explanation* of R in the abductive framework $\langle S, Ab \rangle$ if

- $(S \cup \Delta^+) \setminus \Delta^- \not\models R$
- $(S \cup \Delta^+) \setminus \Delta^-$ is consistent

In the above definition, the sets Δ^+ and Δ^- denote information that needs to be added and deleted, respectively, from a given theory in order to meet a request. Notice that, whenever the underlying logic is monotonic[3], it will always be the case that explanations (Δ^+, Δ^-) will have $\Delta^- = \emptyset$, whereas anti-explanations will have $\Delta^+ = \emptyset$.

3 An abductive technique for handling inconsistency

A fundamental premise of standard abduction as defined in section 2 above is that we start and end with a consistent specification. In the real world, this is not always the case. To be of practical use, abductive reasoning should facilitate the identification of explanations and anti-explanations of a given request within inconsistent specifications, without leading to trivialisation. This can be achieved by deploying formal reasoning that allows continued reasoning in the presence of inconsistency. We have therefore considered QC logic – an adaptation of classical logic that allows non-trivialised reasoning about inconsistent specifications – and adapted classical abductive reasoning to handle inconsistent specifications expressed in this logic. In this section, we briefly review QC logic and we then describe our abductive framework for handling inconsistencies, giving some illustrative examples.

3.1 An overview of QC logic

QC logic adapts the proof theory of classical logic to allow the inference of literals or clauses in the

[2] While our definition of abduction uses the term ‘request’ informally, in this paper we use it specifically to denote the request that a (consistency) rule, R, is no longer violated.

[3] A reasoning mechanism is monotonic if the addition of new information to a specification preserves the set of derivable information (i.e. consequences).

presence of inconsistency. Literals are formulae of the form α and $\neg\alpha$, where α is a ground atomic formula. Clauses are formulae of the form $\alpha_1 \vee \alpha_2 \vee \ldots \vee \alpha_n$, with $n \geq 2$, where α_i is a literal, for each $1 \leq i \leq n$. To illustrate QC reasoning, consider the following partial specification[4], S, of the London Ambulance System (LAS) [Finkelstein & Dowell 96]:

(I) Accident $\rightarrow$ MedicaleEmergency

(II) [Call $\wedge$ MedicalEmergency $\wedge$ NearestAmbulanceAvailable] $\rightarrow$ DispatchAmbulance

(III) [Call $\wedge$ $\neg$NearestAmbulanceAvailable] $\rightarrow$ $\neg$DispatchAmbulance

(IV) $\neg$HasCrew $\rightarrow$ $\neg$NearestAmbulanceAvailable

(V) $\neg$NearestAmbulanceAvailable $\rightarrow$ $\neg$NoCallIssue

together with the following scenario:

(VI) Accident

(VII) Call

(VIII) $\neg$HasCrew

(IX) NearestAmbulanceAvailable

It is easy to see that the scenario and specification, when combined, are inconsistent, because we can generate the two contradictory pieces of information:

DispatchAmbulance (using (I), (II), (VI), (VII) and (IX))

$\neg$DispatchAmbulance (using (III), (IV), (VII) and (VIII))

However, using QC logic, we can still continue reasoning with the above specification and generate additional non-trivial information such as:

$\neg$NoCallIssue (using (IV), (V) and (VIII))

So, even though the specification and scenario are inconsistent, we can still generate other useful inferences.

In classical logic, it is always the case that *either* a piece of information or its negation is true. This link between a piece of information and its negation is decoupled in QC logic. Any formula and its negation could equally be evaluated to be true within a QC model. This is due to the fact that satisfiability of clauses in a QC model is defined in terms of two separate notions of satisfiability, called *strong satisfiability* and *weak satisfiability*, depending on whether the clause is part of a given specification or is derivable from the specification, respectively. Weak satisfiability reflects the notion of truth of classical disjunction (i.e. $\alpha_1 \vee \alpha_2$ is *weakly satisfied* if either α_1 is true or α_2 is true). Strong satisfiability complements this definition with extra conditions. A clause $\alpha_1 \vee \alpha_2$ is defined to be *strongly satisfied* in a

[4] For simplicity, clauses are represented in their equivalent implication form.

model X if (a) either α_1 is true or α_2 is true, (b) if $\neg\alpha_1$ is true then α_2 must be true, and (c) if $\neg\alpha_2$ is true then α_1 must be true. The extra conditions (b) and (c) allow for sound resolution steps in the QC reasoning process[5]. The notion of semantic entailment is also given in terms of strong and weak satisfiability. A formula is *semantically entailed* from a QC specification if it is weakly satisfied in all models that strongly satisfy the specification. A full account of QC logic, its semantics and proof theory is described in [Hunter & Nuseibeh 98].

3.2 Abduction for QC logic

We now describe how to adapt the notion of classical abduction, as defined in section 2.2, in order to develop an abductive technique for inconsistency handling. Within our approach, theories are requirements specifications that are R-inconsistent for some consistency rule R. Resolving an R-inconsistency means identifying the changes that need to be made on the inconsistent specification so that the negation of the rule R is no longer inferred from that specification. A request for an R-inconsistent specification is a request to delete the negation of the rule R from the set of consequences[6] of the specification. Our abductive technique facilitates the identification of changes (Δ^+, Δ^-) to make on the specification so that such request is satisfied; i.e. so that the R-inconsistency is resolved. Since the underlying QC logic is monotonic, these changes (anti-explanations) will only include information to be deleted from the given specification (i.e. the set Δ^-).

Our approach assumes that specifications are expressed in QC logic. Consistency rules are also represented in this logic and therefore can be either literals or clauses. Consequently, the negation of consistency rules can only be literals, or conjunctions of literals. In this context, resolving an inconsistency means abducing anti-explanations of some of the literals that compose the negation of the violated consistency rule R.

Suppose that $\alpha_1 \vee \alpha_2$ is a rule violated in a given specification S. Both the literals $\neg\alpha_1$ and $\neg\alpha_2$ can be inferred from S. To resolve this inconsistency, one of these two literals needs to be removed from the consequences of S. Assume $\neg\alpha_1$ is chosen. This literal can be derived from S either because it is already part of the specification, or because it has been inferred using resolution, or both. In the first case, our abductive technique identifies the literal itself as an anti-explanation; the changes needed will then be the deletion of this literal from the specification. In the second case, our abductive technique identifies, as an anti-explanation of $\neg\alpha_1$, literals involved in the inference process (i.e. resolution steps) that have lead to the derivation of $\neg\alpha_1$. Changes will then be the deletion of (some of) these literals. In the third case, our abductive technique abduces as an anti-explanation both the literal $\neg\alpha_1$, which is in the specification, and

[5] Resolution is the inference of a fact β from the assumptions $(\alpha\vee\beta)$ and $\neg\alpha$.

any other literal which allows the derivation of $\neg\alpha_1$ from other assumptions in the specification.

So, for the particular request of deleting a specific literal from the consequences of a specification, our QC abductive technique identifies "relevant" literals in the specifications, which, once deleted, satisfy the request. The set of abducibles is therefore given by the set of literals included in the QC specification. Formally:

> **QC abduction.** *Abduction for QC logic* is a pair $\langle S, Ab\rangle$ where S is a QC specification and Ab is the set of literals that appear in S. Let R be a violated rule. An *anti-explanation* of $\neg R$ is a set Δ^- of elements from Ab, such that the new specification $S' = (S \setminus \Delta^-) \nvdash \neg R$.

Note that the condition of consistency given in the definition of "standard" abduction (section 2.2) is no longer necessary within our framework. This is because the logic itself allows for inconsistencies. If any change to a given QC specification introduces logical inconsistencies, the resulting specification would still be a valid QC specification.

The abductive process for handling inconsistencies in a QC specification is essentially a backwards reasoning mechanism. Suppose that a literal α needs to be removed from the consequences of a given QC specification in order to resolve an R-inconsistency. The abductive process reasons backward from all the resolution steps that have lead to that literal. If this backwards reasoning is "successful" (i.e. it reaches some relevant literals that are in the specification), then the identified literals become the abducible anti-explanation of the initial literal α. Since QC logic is monotonic, the changes will then be deletion of such literals. Also, since the abductive procedure is essentially used for removing literals that are proved from a given specification (as they are part of the inconsistencies), the backward reasoning will always succeed giving one or more answers. Within a more general setting, for example, in which the abductive reasoning is used for performing arbitrary (evolutionary) changes to a given specification, the abductive process might not always provide an answer, for example in the case when the (evolutionary) change requests are not relevant to a given specification.

Example. To illustrate our abductive process, consider the following simple example. Consider the QC specification S of the LAS, given in section 3.1, and the following consistency rule R:

$$R: \quad \text{DispatchAmbulance} \vee \neg\text{DispatchAmbulance}$$

Recall that:

$$S \vdash_{QC} \text{DispatchAmbulance}$$

and

$$S \vdash_{QC} \neg\text{DispatchAmbulance}$$

[6] Consequences denote derivable information from a specification.

that is, the consistency rule R is violated. Resolving this R-inconsistency means eliminating one of the above two literals. Suppose that we want to eliminate:

DispatchAmbulance

The backward reasoning identifies all possible ways of proving this literal from the given specification. In particular, it looks for disjunctive clauses that include this literal. For example, one possible disjunct is:

$\neg$Call $\vee$ $\neg$MedicalEmergency $\vee$ DispatchAmbulance

The next reasoning step would then be looking for the proof of the literals Call and MedicalEmergency since these, together with the above clause, would have inferred DispatchAmbulance. The backward reasoning is then applied again, but this time on either of the two literals Call and MedicalEmergency. Changes that make either of these two literals no longer inferable from the specification would equally resolve the given inconsistency. If we were interested in identifying all possibly changes, the backward reasoning would have been applied on both the two literals Call and Medical Emergency and the resulting abduced information considered as alternative changes. In this example we choose to apply backward reasoning on the clause:

MedicalEmergency

The only clause including this literal is:

$\neg$Accident $\vee$ MedicalEmergency

The next reasoning step would then be looking for the proof of the literal:

Accident

since this, together with the clause under consideration, would have inferred MedicalEmergency. The backward reasoning is now applied to the literal Accident. There is no clause in the specification that includes this literal, but the literal itself is part of the specification. The backward reasoning then succeeds on this literal and the abductive procedure gives the set:

Δ^{-} = {Accident}

as an answer. Deleting this literal from the specification would indeed stop the inference of the initial literal DispatchAmbulance and therefore resolve the detected R-inconsistency.

Extensions of this process can be defined that would consider the abduction of disjuncts as well as literals. It is also important to note that the abductive process becomes more complex the larger the specification. The backward reasoning needs to branch on each possible way of proving a certain clause and/or literal. The final anti-explanation is, in general, given by the union of all the anti-explanations constructed in each individual successful branch[7]. However, our approach to implementation suggests promising ways of tackling this problem.

4 Towards an implementation

To implement the abductive technique described in the previous section, we have chosen the strategy of using (where possible) existing approaches and tools for abduction. In particular, many abductive procedures have been developed for logic programming [Kakas *et al.* 98], together with associated tool support systems [Kakas & Mourlas 97]. To deploy such techniques and tools, we propose a procedure for expressing any given QC specification, as well as any violated rules, as a logic program. Existing abductive logic programming (LP) tools [Kakas & Mourlas 97] can then be used to identify, for each of the violated rules, anti-explanations in logic program form. These anti-explanations, mapped back into the QC framework, would then provide us with QC anti-explanations, and therefore with the changes that need to be made on our specifications to resolve R-inconsistencies.

Our procedure for generating logic programs from a given QC specification consists of two main steps. The first step translates a given QC specification into a first-order theory, using the translation method described in section 4.2, and the second step generates a logic program from the resulting first-order theory. The intermediate translation step preserves all the semantic and proof theoretic features of QC logic. Our method identifies a set of "axioms" in classical logic that correspond to the semantic notions of strong and weak satisfiability of QC logic mentioned in section 3.1. These axioms together with a first-order translation of a given QC specification derive, within standard first-order logic, the same set of consequences that are derivable from the QC specification using the QC reasoning process. The use of this intermediate step also facilitates general automated reasoning about a given (inconsistent) specification, not necessarily related to the inconsistency handling process. For this purpose, we have developed a theorem prover for QC logic, written in Prolog, which takes advantage of this translation using the correspondence theorem stated in section 4.2.

Before describing the algorithm in detail, we briefly introduce some useful basic notions about abduction for logic programming.

4.1 Abduction for logic programming

Given a first-order language *L*, a logic program[8] *P* is any theory written in *L* composed of *facts* and *rules*. Facts are ground atomic predicates and rules are formulae of the form $\beta \leftarrow \alpha_1, \alpha_2, \ldots, \alpha_n$, where β and α_i, for each $1 \leq i \leq n$, are also atomic predicates. The formula $\beta \leftarrow \alpha_1, \alpha_2, \ldots, \alpha_n$ is read as "β is true if α_1 and

[7] Alternative changes are given by the anti-explanations constructed from different branches.

[8] Note that different version of logic programs have been developed in the literature. In this paper we refer to the basic notion of logic programs without classical negation or disjunctive rules (see [Kowalski 79] for further details).

… and α_n are true”. An example of a logic program P is:

```
MedicaleEmergency(Person,Location) ← Accident(Person,Location).
Accident(John,LondonRoad).
```

Logic program goals can be either (ground) atomic predicates or the “negation” of (ground) atomic predicates; e.g., *not* MedicaleEmergency(Person,Location) (where *not* denotes *negation by failure* [Kowalski 79]). Informally, a logic program proof procedure is based on resolution steps between (sub)goals and rules, or between (sub)goals and facts. A goal α is satisfied if either α is in the program or there is a proof of α. Analogously, a goal α is not satisfied if there is no proof of α and it is not stated in the program as a fact. Abduction for LP can be used to remove information from the set of consequences (successful goals) of a given logic program. This is achieved by considering the negation of a goal under consideration and abducing changes among the facts of a given logic program, which satisfy a set consistency constraints and which would make such new negated goal provable from the program. Using negation by failure, the success of such a negated goal implies that the original goal is no longer provable from the program. A little example is given here, where the logic program and request are:

LP program:	**Request:**
P ← B.	delete(P).
Q ← B1.	
B.	
B1.	

The abductive procedure starts with checking if the negation of P, denoted by P* (where the asterisk (*) denotes classical negation), can be proved from the program. This is because deleting P is equivalent to having P* true. Because the program does not include any rule about P*, P* is abduced as a first piece of information. For consistency checking, the procedure verifies that P is not provable. This means asking that B is not provable (because of the first rule in the program), which itself means asking that B* is provable. Since there is no rule about B*, the abductive procedure abduces also B*. Consistency checking on this new (abduced) assumption verifies that B is not provable. This checking succeeds since the procedure has (abduced) B*. The abductive procedure stops, giving as result the set of abduced changes $\Delta = \{P^*, B^*\}$. Therefore, the changes are the deletion, from the program, of the facts marked with an asterisk and the addition of the facts without an asterisk. Making these changes in the above program results in P no longer being provable (as requested).

4.2 Implementing QC abduction using logic programming

As mentioned earlier, our procedure for generating logic programs from a given QC specification consists of translating first the given QC specification into a first-order theory. This intermediate step of translation provides two main advantages: (a) it enables us to reasoning about an inconsistent specification, independently from the abductive process, and (b) it allows us to consider a simple version of logic programs without classical negation or/and disjunctive clauses.

Translating QC logic into first-order logic. The basic idea is to provide a set of "axioms" in classical logic that correspond to the semantic notions of strong and weak satisfiability of QC logic described in section 3.1. These axioms together with a first-order translation of a given QC specification derive, within standard first-order logic, the same set of consequences that are derivable from the QC specification using the QC reasoning process.

Let L_{FOL} be a first-order language composed of a set of term symbols, given by the whole set of QC formulae, two predicates HoldsS and HoldsW, and the set of classical connectives. The underlying semantics is the standard classical semantics. An example of a ground atomic first-order formula is HoldsS(AmbulanceAvailable). This can be read as "the QC ground atomic formula AmbulanceAvailable is strongly satisfied in some QC model". Analogously, the first-order formula HoldsW(AmbulanceAvailable) can be read as "the QC ground atomic formula AmbulanceAvailable is weakly satisfied in some QC model". For the case of any QC literals α, the first-order formulae HoldsS(α) and HoldsW(α) are equivalent. This is captured by axiom (Ax1) in Definition 2 given in Appendix, and reflects the QC logic property that strong and weak satisfiabilities are equivalent when applied to literals. Any QC specification is translated into a first-order specification by defining, for each formula α in the QC specification, a ground atomic formula HoldsS(α). For instance, the QC specification, S, given in section 3.1 is translated into the following first-order specification, S_{FOL}:

HoldsS(Accident $\rightarrow$ MedicaleEmergency)
HoldsS((Call $\wedge$ MedicalEmergency $\wedge$ NearestAmbulanceAvailable)$\rightarrow$DispatchAmbulance)
HoldsS((Call $\wedge$ $\neg$NearestAmbulanceAvailable)$\rightarrow$ $\neg$DispatchAmbulance)
HoldsS($\neg$HasCrew $\rightarrow$ $\neg$NearestAmbulanceAvailable)
HoldsS($\neg$NearestAmbulanceAvailable $\rightarrow$ $\neg$NoCallIssue)

Every QC formula α that is inferred from a given QC specification S is translated into the ground atomic formula HoldsW(α). Since the inference of a negated consistency rule, $\neg$R, in QC logic is equivalent to the inference of a (conjunction of) literal(s), the first-order translation of a negated rule is also equivalent to the inference of the (conjunction of) atomic formula(e) of the form HoldsW(α_c), where α_c is the converse of a literal α appearing in $\neg$R (see formal definition in the Appendix). For example, the translation of the negation of the consistency rule:

DispatchAmbulance $\vee$ $\neg$DispatchAmbulance

is given by:

$$\mathrm{HoldsW}(\neg \mathrm{DispatchAmbulance}) \wedge \mathrm{HoldsW}(\mathrm{DispatchAmbulance})$$

Since HoldsS and HoldsW are equivalent when applied to literals, translations of negation of consistency rules can also be expressed in terms of conjunctions of HoldsS predicates. We will denote the first-order translation of any QC specifications S with the symbol $\tau(S)$, and the first-order translation of the negation of any violated rule R with $\tau(\neg R)$.

To simulate QC reasoning, we have defined a set of axioms, denoted by Ax_{QC}, for the two predicates HoldsS and HoldsW, that corresponds to the QC semantic definitions of strong satisfiability and weak satisfiability. A formal definition of Ax_{QC} is given in the Appendix. Theorem 1 below shows that this set of axioms guarantees that any information is inferred from a QC specification S if and only if its first-order translation is classically inferred from the first-order translation of S. For instance, in the example described in section 3.1, MedicalEmergency is inferred from S together with the fact Accident. Using the axioms, it is easy to show that the first-order formula HoldsW(MedicalEmergency) is also inferred from S_{FOL} together with the translated fact HoldsS(Accident).

Theorem 1 (Correspondence): Let S be a QC specification and let $\tau(S)$ be its first-order translation. For any QC formula α, α is provable from S if and only if $\mathrm{HoldsW}(\alpha)$ is provable from the first-order translation $\tau(S)$ and the axioms Ax_{QC}. This is,

$$S \vdash_{QC} \alpha \iff \tau(S), Ax_{QC} \vdash_{FOL} \mathrm{HoldsW}(\alpha)$$

A proof of this theorem is outlined in the Appendix, and described in detail in [Russo & Nuseibeh 98]. We have also developed a theorem prover for QC logic to facilitate reasoning over QC specifications. This is written in Prolog and takes advantage of the classical translation of QC specifications into first-order logic, using the correspondence theorem above.

Generating a logic program. The basic idea of this procedure is to construct from the first-order translation of a given QC specification a logic program that includes (a) the translated specification (as facts) and (b) rules, which allow the simulation of QC inferences from the specifications. The logic program resolution mechanism thus captures the resolution mechanism of QC logic. The resulting logic program also includes some integrity constraints to preserve its own consistency.

Suppose that the first-order translation of a given QC specification is the following set:

$$S_{FOL} = \{\mathrm{HoldsS}(\alpha_1), \mathrm{HoldsS}(\alpha_2), \ldots, \mathrm{HoldsS}(\alpha_n)\}$$

where α_i are either literals or clauses appearing in the QC specification. Suppose also that the largest clause in the QC specification has m literals (i.e. size m), with $m \geq 2$. The generation of a logic program from S_{FOL} is an m-step process. Each step constructs a partial logic program that extends the one generated in the immediately previous step. The construction of such a partial program is as follows:

Step 1 A partial program, called $\mathbf{P_1}$, is constructed by considering just the set $\{HoldsS(\alpha_1), HoldsS(\alpha_2),\ldots, HoldsS(\alpha_n)\}$.

Step 2 A partial program $\mathbf{P_2}$ is constructed by considering each $HoldsS(\alpha)$ in P_1, where α is a QC clause with size **m**, and adding to P_1 a rule of the form $HoldsS(\beta)\leftarrow HoldsS(\neg\alpha_i),HoldsS(\alpha)$ for each literal α_i in α, for which $HoldsS(\neg\alpha_i)$ is in P_1. The term β is the QC clause obtained from α by cancelling the literal α_i. If $HoldsS(\neg\alpha_i)$ is not in P_1 for all α_i in α, then $HoldsS(\alpha_j)$ is added to P_1 for some α_j in α.

Step 3 A partial program $\mathbf{P_3}$ is constructed in a way similar to that shown in Step 2, by considering all the formulae $HoldsS(\alpha)$, where α is a QC clause of size **m-1**, that are in P_2 or for which there is a rule in P_2 with $HoldsS(\alpha)$ in its left side.

Step k For each k, $4 \le k \le m$, a partial program $\mathbf{P_k}$ is constructed in the same way as shown in Step 3, by considering the clause of size (m-k)+2 that are in P_{k-1}.

The final program **P** is given by the program P_m. For QC specifications with no clauses, but only literals, the logic program P is simply given by the set of $HoldsS(\alpha_i)$, for each literal α_i included in the specification. We illustrate the above algorithm with an example.

Example. Let a QC specification be the set $S = \{ p\vee q\vee r, \neg p, \neg q, t\vee s \}$. We generate a logic program for this specification in the following way. First we define its first-order translation S_{FOL}. This is given by the set:

$$S_{FOL} = \{HoldsS(p\vee q\vee r), HoldsS(\neg p), HoldsS(\neg q), HoldsS(t\vee s)\}$$

Then we apply the algorithm described above to generate, from this first-order specification S_{FOL} and the classical axioms simulating QC reasoning, the associated logic program **P**. The construction of P is composed of 3 steps, since the largest term that appears in S_{FOL} has size 3. The final program **P** is given by $\mathbf{P_3}$ below.

Step 1. The program $\mathbf{P_1}$ is just the set:

$$\{HoldsS(p\vee q\vee r), HoldsS(\neg p), HoldsS(\neg q), HoldsS(t\vee s)\}.$$

Step 2. The program $\mathbf{P_2}$ is as follows:

$P_1 \cup \{\ HoldsS(p\vee r) \leftarrow HoldsS(\neg q),HoldsS(p\vee q\vee r).$
$HoldsS(q\vee r) \leftarrow HoldsS(\neg p),HoldsS(p\vee q\vee r).\ \}$

Step 3. The program $\mathbf{P_3}$ is as follows:

$P_2 \cup \{$ HoldsS(r) $\leftarrow$ HoldsS($\neg$p), HoldsS(p$\vee$r).
HoldsS(r) $\leftarrow$ HoldsS($\neg$q), HoldsS(q$\vee$r).
HoldsS(t). $\}$

Given a consistency rule R = $\neg$r, S derives the negation of this rule, (i.e. S $\vdash_{QC}$ r). So, S is R-inconsistent. To resolve this inconsistency, we apply the abductive procedure for logic programs to the above program considering as request the deletion of HoldsS(r)[9].

The procedure starts then with checking if HoldsS*(r) is provable from the program. Since HoldsS*(r) cannot be proved, it is assumed as hypothetical information, i.e. Δ_0 = {HoldsS*(r)}. Consistency for HoldsS*(r) is then checked. Specifically, the program assumes HoldsS*(r) and then verifies that HoldsS(r) fails. With respect to the first rule in the program, HoldsS(r) fails if either HoldsS($\neg$p) or HoldsS(p$\vee$r) fails. Suppose that the procedure checks if HoldsS($\neg$p) fails. This means checking if its negation, HoldsS*($\neg$p), succeeds. Since HoldsS*($\neg$p) cannot be proved, it is added to the set of hypothesis, Δ_1 = {HoldsS*(r), HoldsS*($\neg$p)}. The procedure now uses this set of hypotheses in trying to verify that HoldsS(r) fails also with respect to its second rule in the program. For HoldsS(r) to fail, it is necessary that either HoldsS($\neg$q) or HoldsS(q$\vee$r) fails. Suppose now that the procedure checks if HoldsS(q$\vee$r) fails. Using its rule, this means checking that either HoldsS($\neg$p) or HoldsS(p$\vee$q$\vee$r) fails. Since HoldsS*($\neg$p) is in the set of hypotheses, Δ_1, then HoldsS($\neg$p) fails directly. Hence, the result of the abductive procedure is the set {HoldsS*(r), HoldsS*($\neg$p)}, which correspond to the changes:

{delete HoldsS(r), delete HoldsS($\neg$p)}.

Because HoldsS(r) does not belong to the set of facts in the program, only the second piece of information, HoldsS($\neg$p), is deleted.

The above abductive procedure for logic programs follows a similar backwards reasoning mechanism to that outlined in section 3.2. It is, however, important to formally prove that it does indeed capture the intended result of resolving an R-inconsistency in a given QC specification. The following theorem states this, showing that our algorithm for generating a logic program from a QC specification and the above abductive procedure for logic programs, are sound with respect to our notion of abduction for QC specifications.

Theorem 2 (Correctness): Let S be a specification and let R be a violated consistency rule (i.e. S $\vdash_{QC}$ $\neg$R). Let P be the associated program and $\tau(\neg R)$ the translation of the negated rule. Let $\tau(\Delta)$ be the set of hypothetical changes on the program P determined by the abductive procedure on P. Then:

[9] Since the request is about deleting a consequence, its translated representation should really be HoldsW(r). But because weak and strong satisfiability in QC logic are, for atomic literals, equivalent, the above translation HoldsW(r) is also equivalent to HoldsS(r), as illustrated by

$$\text{If } P \setminus \tau(\Delta) \not\models \tau(\neg R) \text{ then } S \setminus \Delta \not\models_{QC} \neg R.$$

A proof of this theorem is outlined in the Appendix.

5 Example

We now illustrate our approach on a partial specification drawn from a library system. The requirements are that such a system should allow individuals to borrow books from the library if they need these books and if the books are available. Once a book is borrowed, it is no longer available for others to borrow. A representation of this specification in QC logic is:

$$\begin{aligned} S_{QC} = \{ \; & (\text{BookCopy} \wedge \text{BookInLibrary}) \rightarrow \text{BookAvailable}, \\ & (\text{BookCopy} \wedge \text{BookNeeded} \wedge \text{BookAvailable}) \rightarrow \text{BorrowingBook}, \\ & \text{BookCopy}, \\ & \text{BookInLibrary}, \\ & \text{BookNeeded} \; \} \end{aligned}$$

with R_{QC} = BorrowingBook $\rightarrow \neg$BookAvailable as a consistency rule. This specification is inconsistent. The negation of the rule R_{QC}, given by $\neg R_{QC}$ = BorrowingBook $\wedge$ BookAvailable, can be inferred from the specification, S_{QC}. Resolving this inconsistency means eliminating one of the above two literals from the set of consequences of the specification. To eliminate the literal BookAvailable, abduction for QC logic identifies two possible sets of changes: {delete BookCopy} and {delete BookInLibrary}. Either of these two changes, if performed on the specification, would resolve the inconsistency.

Next, we show how this abductive technique is performed by our implementation. First the QC specification, S_{QC}, is translated into a first-order specification, S_{FOL}:

$$\begin{aligned} S_{FOL} = \{ \; & \text{HoldsS}((\text{BookCopy} \wedge \text{BookInLibrary}) \rightarrow \text{BookAvailable}), \\ & \text{HoldsS}((\text{BookCopy} \wedge \text{BookNeeded} \wedge \text{BookAvailable}) \rightarrow \text{BorrowingBook}), \\ & \text{HoldsS}(\text{BookCopy}), \text{HoldsS}(\text{BookInLibrary}), \text{HoldsS}(\text{BookNeeded}) \; \} \end{aligned}$$

The translation of the negated rule (i.e. $\tau(\neg R_{QC})$) is HoldsS(BorrowingBook) $\wedge$ HoldsS(BookAvailable).

Applying the algorithm for generating a logic program from S_{FOL} we get:

$$\begin{aligned} P_S = \{ \; & \text{HoldsS}((\text{BookCopy} \wedge \text{BookInLibrary}) \rightarrow \text{BookAvailable}). \\ & \text{HoldsS}((\text{BookCopy} \wedge \text{BookNeeded} \wedge \text{BookAvailable}) \rightarrow \text{BorrowingBook}). \\ & \text{HoldsS}(\text{BookCopy}). \\ & \text{HoldsS}(\text{BookInLibrary}). \\ & \text{HoldsS}(\text{BookNeeded}). \end{aligned} \quad \Bigg] \; P_1$$

the axiomatisation given in Appendix.

HoldsS((BookNeeded ∧ BookAvailable) → BorrowingBook) ←
  HoldsS(BookCopy),
  HoldsS((BookCopy ∧ BookNeeded ∧ BookAvailable) → BorrowingBook).

HoldsS((BookCopy ∧ BookAvailable) → BorrowingBook) ←
  HoldsS(BookNeeded),
  HoldsS((BookCopy ∧ BookNeeded ∧ BookAvailable) → BorrowingBook).

P_2

HoldsS(BookInLibrary → BookAvailable) ←
  HoldsS(BookCopy),
  HoldsS((BookCopy ∧ BookInLibrary) → BookAvailable).

HoldsS(BookCopy → BookAvailable) ←
  HoldsS(BookInLibrary),
  HoldsS((BookCopy ∧ BookInLibrary) → BookAvailable).

HoldsS(BookAvailable → BorrowingBook) ←
  HoldsS(BookNeeded),
  HoldsS((BookNeeded ∧ BookAvailable) → BorrowingBook).

HoldsS(BookAvailable → BorrowingBook) ←
  HoldsS(BookCopy),
  HoldsS((BookCopy ∧ BookAvailable) → BorrowingBook).

P_3

HoldsS(BookAvailable) ←
  HoldsS(BookInLibrary),
  HoldsS(BookInLibrary → BookAvailable).

HoldsS(BookAvailable) ←
  HoldsS(BookCopy),
  HoldsS(BookCopy → BookAvailable).

P_4

Since HoldsS(BookAvailable) is derivable from the above program P_S, our abductive goal is to delete it. The abductive procedure considers the negated query HoldsS*(BookAvailable), and reasons backwards performing the associated consistency checking. The procedure generates two alternative sets of (smallest) anti-explanations:

Δ_1 = { HoldsS*(BookAvailable), HoldsS*(BookCopy) }

and

Δ_2 = { HoldsS*(BookAvailable), HoldsS*(BookInLibrary) }

These two sets generate, respectively, two alternative sets of changes:

{delete HoldsS(BookCopy)}

and

{delete HoldsS(BookInLibrary)}

Performing the change in either of these two sets on the above program, would make HoldsS(BookAvailable) no longer provable from P_S. These two sets, mapped back into QC logic, give the two sets of changes {delete BookCopy} and {delete BookInLibrary}, which were also identified by the abductive technique for QC logic.

6 Related Work

A number of logic-based approaches for handling inconsistency have been proposed in the literature. None, however, appear to tolerate inconsistencies explicitly. Zowghi and Offen suggest belief revision for default theories as a formal approach for resolving inconsistencies arising during the evolution of requirements specifications [Zowghi & Offen 97]. Change actions are implicitly given by the definition of a belief revision operator, which basically changes the status of information from defeasible to non-defeasible or vice-versa, to remove the derivation of inconsistencies. Similarly, Ryan defines ordering relations on default information [Ryan 93]. Conflicting defaults are resolved not by changing the specification but by considering only scenarios or models of the inconsistent specification, which satisfy as much of the preferable information as possible (in order of preference). Anderson and Durney on the other hand, propose “relative utility” analysis for guiding the choice among possible alternative changes in order to resolve a detected inconsistency [Anderson & Durney 93]. Changes that “enable” scenarios showing “desirable transitions” are preferred to changes that do not, and changes that “disable” scenarios showing “prohibited transitions” are also preferred to those that do not. Such heuristics can also be interpreted as preferring changes that do not cause additional inconsistencies to those that do.

A closely related logic-based approach is proposed by van Lamsweerde *et al.* [van Lamsweerde and Letier 98; van Lamsweerde *et al.* 98]. In their work they offer a goal-driven approach to requirement engineering in which “obstacles” denote parts of a specification that lead to a negated goal. van Lamsweerde’s notion of goals is comparable to our notion of consistency rules, and his notion of obstacles corresponds to the abduced facts detected by our abductive technique. Both approaches guarantee that changes performed on abduced information or obstacles would resolve a detected inconsistency. The main difference, which is partly due to the differing logical representations used, is that whereas van Lamsweerde *et al.* adopt goal regression to identify possible obstacles, we use abduction. Since we also use QC logic as the underlying formal reasoning, it is not necessary to resolve all inconsistency in order to continue to do meaningful reasoning.

Duffy *et al.* also propose a logic-based approach for reasoning about requirements specifications based on the construction of goal tree structures[10] [Duffy *et al.* 95]. However, their approach is less systematic, in that analysis of alternative changes is carried out by investigating which goals would be satisfied and which would not, after adding or removing facts to a specification. Recent work by Menzies has also demonstrated the applicability of abductive reasoning in knowledge-based software engineering, using an inference procedure for “knowledge-level modeling” that can support prediction, explanation, planning, etc. [Menzies 96]. However, the technique is tailored for specifications represented as and-or graphs, where vertices are atomic facts and edges are causal relationships between facts.

7 Conclusions and Future Work

The abductive technique outlined in this paper is another tool in our growing toolbox for handling inconsistency in requirements specifications. While this work provides a novel contribution to problems of managing evolving specifications, a number of critical research issues still need to be explored. To make the work more accessible, we need to extend our approach to allow (a) arbitrary QC formulae to represent requirements specifications and (b) first-order reasoning. This could be achieved by introducing automatic rewrites of any QC specification into disjunctive normal form, which would express universal and existential quantifiers, on a finite domain, in terms of conjunction and disjunction of ground formulae. This extension would facilitate an easier and more direct representation of requirements specification into QC logic.

The abductive technique described can also be extended in a variety of ways. Most pressing is the need to abduce not only literals, but also more complex formulae – changes to a specification are typically more than just atomic changes.

Of course, choosing the right changes to perform, even given a simple choice of atomic changes, is a difficult task. The impact of each action must be evaluated, and actions with the most ‘desirable’ consequences chosen. We have started to explore ways of comparing the consequences of performing different actions on (inconsistent) specifications [Nuseibeh & Russo 98]. However, like abducing the inconsistency handling changes described in this paper, the crucial issues of complexity and scale must also be addressed[11].

We are attempting to address these issues in three ways. First, using the translation technique of QC specifications into first-order logic, we are investigating the possibility of using existing theorem provers

---

[10] This approach is similar to logic-based model checking. However, model checkers (such as SPIN [Holzmann 97]) differ from our tools in that they assume (and require) a (logically) *consistent* specification as input before they can analyse it.

[11] It is well known that abductive reasoning is NP-hard [Selman & Levesque 90]. However, by exploiting modularity of specifications and limiting the use of abduction to particular specification structures and domains, we believe that we can still perform some useful reasoning about inconsistent specifications.

for classical logic in order to allow automated reasoning on large inconsistent specifications. Second, and building on the work of Menzies [Menzies *et al.* 98], we are conducting controlled experiments to investigate the limits to (abductive) reasoning about inconsistent specifications [Menzies *et al.* 99]. These limits include attributes such as the size of the specifications and their structure. Third, we are continuing case study work [Russo *et al.* 98] to determine how to (re)structure requirements specifications using "ViewPoints" [Nuseibeh *et al.* 94] in ways that make them more amenable to the kind of reasoning we have described. The case studies are also providing valuable domain-specific knowledge and heuristics that are essential for pruning the abductive reasoning search space and focusing development actions.

Nevertheless, and despite the difficult issues of scalability above, we believe that using abductive reasoning as outlined in this paper provides useful guidance to the management of inconsistency in evolving specifications. The paper has laid the foundations for our approach and demonstrated it on a small, but illustrative, example. Further work is still needed to explore the application contexts in which the approach is most suited.

8 Acknowledgements

We would like to thank Steve Easterbrook, Tony Kakas, Jeff Kramer, Tim Menzies and Axel van Lamsweerde for their comments on earlier drafts of this paper, and Mark Tan for his work on the tool support. We would also like to acknowledge the financial support of the UK EPSRC for the projects MISE (GR/L 55964) and VOICI (GR/M 38582).

9 References

[Anderson & Durney 93] J.S. Anderson and B. Durney. "Using Scenario in Deficiency-driven Requirements Engineering", *IEEE Proceedings of International Symposium on Requirements Engineering (RE '93)*, 134-141, San Diego, January 1993.

[Balzer 91] R. Balzer, "Tolerating Inconsistency", *IEEE Proceedings of 13th International Conference on Software Engineering (ICSE-13)*, Austin, Texas, 1991.

[Bohner & Arnold 96] S.A. Bohner and R.S. Arnold, *Software Change Impact Analysis*, IEEE CS Press, 1996.

[Console *et al.* 94] L. Console, M.L. Sapino and D. Theseider Dupre, "The role of abduction in database view updates", *Journal of Intelligent Systems*, 1994.

[Console *et al.* 96] L. Console, L. Portinale and D. Theseider Dupre, "Using Compiled Knowledge to Guide and Focus Abductive Diagnosis", *IEEE Transaction on Knowledge and Data Engineering,* 8(5): 690-706, 1996.

[Cugola *et al.* 96] G. Cugola, E. Di Nitto, A. Fuggetta and C. Ghezzi, "A framework for formalising inconsistencies and deviations in human-centred systems", *ACM Transaction on Software*

Engineering and Methodology, 5(3):191-230, 1996.

[Duffy *et al.* 95] D. Duffy, C. MacNish, J. McDermid and P. Morris, "A Framework for Requirements Analysis Using Automated Reasoning", *Proceedings of 7th International Conference on CaiSE'95,* LNCS 932, Springer-Verlag, 68-81, 1995.

[Esghsi 93] K. Esghsi, "A Tractable Class of Abductive Problems", *Proceedings of 5th International Conference on Logic Programming*, 1998.

[Esghsi 98] K. Esghsi, "Abductive Planning with the Event Calculus", *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI '95)*, 1:3-8, 1998.

[Finkelstein & Dowell 96] A. Finkelstein and J. Dowell, "A comedy of Errors: the London Ambulance Service case study", *IEEE Proceedings of 8th International Workshop on Software Specification and Design (IWSSD-8),* 2-4, Schloss Velen, 22-23rd March 1996, Germany, IEEE CS Press.

[Finkelstein *et al.* 94] A. Finkelstein, D. Gabbay, A. Hunter, J. Kramer and B. Nuseibeh, "Inconsistency Handling in Multi-Perspective Specifications", *IEEE Transaction on Software Engineering,* 20(8): 569-578, 1994.

[Holzmann 97] J. Holzmann, "The Model Checker SPIN", *IEEE Transaction on Software Engineering,* 23(5): 279-295, 1997.

[Hunter & Nuseibeh 98] A. Hunter and B. Nuseibeh, "Managing Inconsistent Specifications: Reasoning, Analysis and Action", *ACM Transaction on Software Engineering and Methodology,* October 1998.

[Inoue & Sakama 95] K. Inoue and C. Sakam, "Abductive Framework for Non-monotonic Theory Change". *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI '95),* 1:204-210, 1995.

[Kakas *et al.* 98] A.C. Kakas, R.A. Kowalski and F. Toni, "The Role of Abduction in Logic Programming", *Handbook of Logic in Artificial Intelligence and Logic Programming,* 5:235-324, D.M. Gabbay, C.J. Hogger and J.A. Robinson eds., Oxford University Press, 1998.

[Kakas & Mancarella 90] A.C. Kakas and P. Mancarella, "Database Updates Through Abduction", *Proceedings of 16th International Conference on Very Large Database (VLDB '90),* Brisbane, Australia, 1990.

[Kakas & Mourlas 97] A.C. Kakas and C. Mourlas, "ACLP: Flexible Solutions to Complex Problems", *Proceedings of the International Conference on Logic Programming and Non-Monotonic Reasoning,* 1997.

[Kowalski 79] R.A. Kowalski, *Logic for problem solving*, Elsevier, New York, 1979.

[Menzies 96] T. Menzies, "Applications of Abduction: Knowledge Level Modeling.", *International Journal of Human Computer Studies,* 1996.

[Menzies *et al.* 98] T. Menzies, A. Ghose and S. Waugh, "Abduction: Experiments and Implications", *Knowledge Acquisition,* 1998.

[Menzies *et al.* 99] T. Menzies, S. Easterbrook, B. Nuseibeh and T. Waugh, "An empirical investigation of reasoning with multiple viewpoints', *IEEE Proceedings of 4th International Symposium on Requirements Engineering (RE '99), Limerick, Ireland, June 1999.*

[Nuseibeh *et al.* 94] B. Nuseibeh, J. Kramer and A. Finkelstein, "A Framework for Expressing the Relationships Between Multiple Views in Requirements Specification", *IEEE Transaction on Software Engineering,* 20(10): 760-773, 1994.

[Nuseibeh & Russo 98] B. Nuseibeh and A. Russo, "On the consequences of acting in the presence of inconsistency", *Proceedings of 6th IEEE International Workshop on Software Specification & Design (IWSSD-9),* Japan, 1998.

[Russo *et al.* 98] A. Russo, B. Nuseibeh and J Kramer, "Restructuring requirements Specifications for Managing Inconsistency and Change: A Case Study", *IEEE Proceedings of 3rd International Conference on Requirements Engineering (ICRE '98),* Colorado Springs, USA, April 1998.

[Russo & Nuseibeh 98] A. Russo and B. Nuseibeh, "An abductive technique for QC logic", Technical Report, Department of Computing, Imperial College, 1998.

[Ryan 93] M. Ryan, "Default in Specification", *IEEE Proceedings of International Symposium on Requirements Engineering (RE'93),* 266-272, San Diego, California, January 1993.

[Selmen & Levesque 90] B. Selman and H.J. Levesque, Abductive and Default Reasoning: a computational core, *AAAI '90*, 343-348, 1990.

[Schwanke & Kaiser 88] R.W. Schwanke and G.E. Kaiser, "Living with Inconsistency in Large Systems", *Proceedings of the International Workshop on Software Version and Configuration Control*, Grassau, Germany, 1988.

[van Lamsweerde & Letier 98] A. van Lamsweerde and E. Letier, "Integrating Obstacles in Goal-Driven Requirements Engineering", *IEEE Proceedings of 20th International Conference of Software Engineering (ICSE-20)*, 53-62, 1998.

[van Lamsweerde et al. 98] A. van Lamsweerde, R. Darimont and E. Letier, "Managing Conflicts in Goal-Driven Requirements Engineering", *IEEE Transaction on Software Engineering,* Nov. 1998.

[Zowghi & Offen 97] D. Zowghi and R. Offen, "A Logical Framework for Modelling and Reasoning about the Evolution of Requirements", *IEEE Proceedings of 3rd International Symposium on Requirements Engineering (RE '97),* Annapolis, January 1997.

Appendix

This appendix outlines proofs of Theorems 1 and 2 of the paper, together with some basic definitions. For a full account of the proofs, the reader is referred to [Russo & Nuseibeh 98].

Definition 1. (Focus): Let $\alpha_1 \vee \ldots \vee \alpha_n$ be a QC clause and let α_i for some $1 \leq i \leq n$ be one of the literals in this clause. $\mathsf{Focus}(\alpha_1 \vee \ldots \vee \alpha_n, \alpha_i)$ is the clause $\beta_1 \vee \ldots \vee \beta_{n-1}$, where, for each j, $1 \leq j \leq n$, β_j is a literal in $\alpha_1 \vee \ldots \vee \alpha_n$ different from α_i.

Definition 2. (First-order axioms of QC specifications): The first-order axiomatisation AX_{QC} for a QC specification is the following three axiom schemas:

(Ax1) $\mathsf{HoldsS}(\alpha) \leftrightarrow \mathsf{HoldsW}(\alpha)$.

(Ax2) $\mathsf{HoldsS}(\alpha_1 \vee \ldots \vee \alpha_n) \leftrightarrow$

$$[\ (\mathsf{HoldsS}(\alpha_1) \vee \ldots \vee \mathsf{HoldsS}(\alpha_n)) \wedge$$
$$(\mathsf{HoldsS}(\neg\alpha_1) \rightarrow \mathsf{HoldsS}(\mathsf{Focus}(\alpha_1 \vee \ldots \vee \alpha_n, \alpha_1))) \wedge \ldots \wedge$$
$$(\mathsf{HoldsS}(\neg\alpha_n) \rightarrow \mathsf{HoldsS}(\mathsf{Focus}(\alpha_1 \vee \ldots \vee \alpha_n, \alpha_n)))\ ]$$

(Ax3) $\mathsf{HoldsW}(\alpha_1 \vee \ldots \vee \alpha_n) \leftrightarrow (\mathsf{HoldsW}(\alpha_1) \vee \ldots \vee \mathsf{HoldsW}(\alpha_n))$

Definition 3. (Translation of a violated rule): Let S be a QC specification. Let R be a QC violated rule. $\tau(\neg R)$ is a ground atomic first-order formula given by $\mathsf{HoldsW}(\alpha_c)$ if R is the single literal α, or $\mathsf{HoldsW}(\alpha_{ci})$ for some $1 \leq i \leq n$, if R is the clause $\alpha_1 \vee \ldots \vee \alpha_n$. Note that, because of (Ax1) in the above definition, $\mathsf{HoldsW}(\alpha_c)$ and $\mathsf{HoldsW}(\alpha_{ci})$ can equally be replaced by $\mathsf{HoldsS}(\alpha_c)$ and $\mathsf{HoldsS}(\alpha_{ci})$.

Proof of Theorem 1. The two derivability relations $\vdash_{QC}$ and $\vdash_{FOL}$ are sound and complete with respect to their semantic entailment relations. Therefore we prove the following equivalent statement,

$$S \vDash_{QC} \alpha \quad \Leftrightarrow \quad \tau(S), Ax_{QC} \vDash_{FOL} \mathsf{HoldsW}(\alpha).$$

If half. Proof by contradiction. Assume that $\tau(S), Ax_{QC} \vDash_{FOL} \mathsf{HoldsW}(\alpha)$ and that $S \nvDash_{QC} \alpha$. There exists a QC model X such that for all formulae $\beta \in S$, $X \vDash_s \beta$ and $X \nvDash_s \alpha$. (Note that the symbol $\vDash_s$ denotes QC strong satisfiability.) The proof consists of constructing, from X, a classical interpretation I, such that I satisfies Ax_{QC}, $\mathsf{HoldsS}(\beta)$ for each $\mathsf{HoldsS}(\beta) \in \tau(S)$, and $\neg\mathsf{HoldsW}(\alpha)$. Therefore, I does not satisfy $\mathsf{HoldsW}(\alpha)$, so contradicting the initial hypothesis. I is defined as follows. $\|\mathsf{HoldsS}\|_I = \{\beta \mid X \vDash_s \beta\}$, and $\|\mathsf{HoldsW}\|_I = \{\beta \mid X \vDash_w \beta\}$.

Only-if half. Proof by contradiction. Assume that $S \vDash_{QC} \alpha$ and that $\tau(S), Ax_{QC} \nvDash_{FOL} \mathsf{HoldsW}(\alpha)$. Therefore, there exists a classical model I of Ax_{QC}, which satisfies $\mathsf{HoldsS}(\beta)$, for each $\mathsf{HoldsS}(\beta) \in \tau(S)$, and which does not satisfy $\mathsf{HoldsW}(\alpha)$. The proof consists of showing that there exists a QC model X such that $X \vDash_s \beta$, for each $\beta \in S$, and $X \nvDash_w \alpha$, so contradicting the initial hypothesis. (Note that the symbol $\vDash_w$ denotes QC weak satisfiability.) X is constructed from the classical interpretation I, as follows. For each atomic ground predicate α_i, of the QC language,

$$X=\{+\alpha_i, \mid \alpha_i \in \|\text{HoldsS}\|_I\}\cup\{-\alpha_i, \mid \neg\alpha_i \in \|\text{HoldsS}\|_I\}$$

X satisfies the following properties. (1) For each literal α, $X\models_s \alpha$ if and only I satisfies HoldsS(α). (2) For each literal α, $X\models_w \alpha$ if and only if I satisfies HoldsW(α). (3) For each disjunct $\alpha_1\vee\ldots\vee\alpha_n$, $X\models_s \alpha_1\vee\ldots\vee\alpha_n$ if and only if I satisfies HoldsS($\alpha_1\vee\ldots\vee\alpha_n$). (4) For each disjunct $\alpha_1\vee\ldots\vee\alpha_n$, $X\models_w \alpha_1\vee\ldots\vee\alpha_n$ if and only if I satisfies HoldsW($\alpha_1\vee\ldots\vee\alpha_n$). These properties, together with the initial hypothesis, imply that, for each $\beta\in S$, $X\models_s\beta$ and $X \not\models_w \alpha$.

Proof of Theorem 2. Let S'= S \ Δ and let P'= P \ τ(Δ). Two main steps: (1) show that if $P'\not\models\tau(\neg R)$ then $\tau(S'),Ax_{QC}\not\models_{FOL}\tau(\neg R)$, and (2) show that if $\tau(S'),Ax_{QC}\not\models_{FOL}\tau(\neg R)$ then $S'\not\models_{QC}\neg R$. The second step is already given by Theorem 1. The first step is proved as follows. Let Tr_Δ be the new set of facts obtained by performing τ(Δ) changes on the facts of P. By the correctness theorem in [Kakas & Mancarella 90], there exists a stable model, called M, of the new program P', which satisfies ¬HoldsS(¬R) and therefore which does not satisfy HoldsS(¬R). The rest of the proof consists of showing that there exists a classical model I of Ax_{QC} that satisfies τ(S') and that does not satisfy τ(¬R). I is constructed as follows:

- For each literal α, $\alpha\in \|\text{HoldsS}\|_I$ if and only if $\alpha\in \|\text{HoldsS}\|_M$.
- For each literal α, $\alpha\in \|\text{HoldsW}\|_I$ if and only if $\alpha\in \|\text{HoldsS}\|_I$.
- For each clause $\alpha_1\vee\ldots\vee\alpha_n$,
  if $\alpha_1\vee\ldots\vee\alpha_n\in \|\text{HoldsS}\|_M$ then $\alpha_1\vee\ldots\vee\alpha_n\in \|\text{HoldsS}\|_I$;
  if $\alpha_1\vee\ldots\vee\alpha_n\notin \|\text{HoldsS}\|_M$ then,
    if for all α_i, $1\leq i\leq n$, $\alpha_i\notin \|\text{HoldsS}\|_I$ then
      $\alpha_1\vee\ldots\vee\alpha_n\notin \|\text{HoldsS}\|_I$;
    if for some α_i, $1\leq i\leq n$ $\alpha_i\in \|\text{HoldsS}\|_I$, then
      $\alpha_1\vee\ldots\vee\alpha_n\in \|\text{HoldsS}\|_I$ if and only if for all α_i,
      $1\leq i\leq n$ such that $\alpha_{ci} \in \|\text{HoldsS}\|_I$, Focus($\alpha_1\vee\ldots\vee\alpha_n$, α_i) $\in \|\text{HoldsS}\|_I$.
- For each clause $\alpha_1\vee\ldots\vee\alpha_n$, $\alpha_1\vee\ldots\vee\alpha_n\in \|\text{HoldsW}\|_I$ if and only if for some α_i, $1\leq i\leq n$, $\alpha_i\in \|\text{HoldsS}\|_I$.