# Handling Inconsistencies in Z using Quasi-Classical Logic

Ralph Miarka, John Derrick, Eerke Boiten

Computing Laboratory, University of Kent, Canterbury, CT2 7NF, UK
Email: {rm17,jd1,eab2}@ukc.ac.uk

**Abstract.** The aim of this paper is to discuss what formal support can be given to the process of living with inconsistencies in Z, rather than eradicating them. Logicians have developed a range of logics to continue to reason in the presence of inconsistencies. We present one representative of such paraconsistent logics, namely Hunter's quasi-classical logic, and apply it to the analysis of inconsistent Z schemas. In the presence of inconsistency quasi-classical logic allows us to derive less, but more "useful", information. Consequently, inconsistent Z specifications can be analysed in more depth than at present. Part of the analysis of a Z operation is the calculation of the precondition. However, in the presence of an inconsistency, information about the intended application of the operation may be lost. It is our aim to regain this information. We introduce a new classification of precondition areas, based on the notions of definedness, overdefinedness and undefinedness. We then discuss two options to determine these areas both of which are based on restrictions of classical reasoning.

## 1 Introduction

The purpose of this paper is to discuss how to reason in the presence of inconsistencies in a formal setting. Although this might sound strange, specifications, especially large ones, are often inconsistent at some level. Inconsistencies range from contradictory descriptions of the system at hand to contradictions specified in the operations. A significant proportion of the specification analysis process is then devoted to detecting and eliminating such inconsistencies, because, classically (and intuitively), inconsistencies in specifications are regarded as undesirable.

However, those involved in large scale software engineering in practice treat inconsistencies as a fact of life. They occur frequently in large projects and need to be tolerated (possibly for some time) and managed, rather than eradicated immediately. This has led to a considerable amount of research on the development of tools and techniques for living with inconsistencies (Ghezzi and Nuseibeh, 1998), (Balzer, 1991), (Schwanke and Kaiser, 1988), and handling inconsistencies (Finkelstein et al., 1994), (Hunter and Nuseibeh, 1998). The general aim of such work is to provide practical support for deciding if, when, and how to remove inconsistencies, and to possibly reason in the presence of inconsistencies.

Although the techniques and tools developed for this approach have had a certain amount of success they have, however, mainly focused on informal and semi-formal specification techniques. There has been recent work on more formal approaches (Hunter and Nuseibeh, 1997), but these have largely concentrated on purely logical issues, not connecting themselves to current specification languages. We are interested in seeing what support we can give for the process of living with inconsistencies in a specification notation, namely Z.

Our purpose here is to explore the issue (rather than offering any definite solutions), discussing how inconsistencies can arise and how they might be handled, especially those present in operations. A number of options are discussed, all of which have the same general aim, namely, in the presence of inconsistency, not to immediately derive falsehood, but rather allow further, intermediate, reasoning on other aspects of the state, operation, or specification. These options include restricting the standard logic used for reasoning about Z specifications, as well as using alternative, so-called paraconsistent logics.

The paper is structured as follows. In Section 2 we present a small supporting example, illustrating some sources of inconsistencies and the problems of analysing such Z specifications. Following this we introduce, in Section 3, one way of supporting the reasoning process in the presence of inconsistencies, by presenting a paraconsistent logic called quasi-classical logic. Further, in Section 4 we use quasi-classical logic to support the process of reasoning in the presence of inconsistency. We exemplify the methods in terms of our example as we go along in this work. We consider the particular problem of deriving preconditions of operations in Section 5. Finally, we give some concluding remarks with links to related and future work in Section 6.

## 2 Background

In this section we introduce a small example written in the specification language Z. The advantage of Z and other formal methods is the possibility of formally analysing a given specification. We discuss two particular ways of analysing Z specifications. This work is concerned with the notion of inconsistency, hence we present some account of it at the end of this section.

### 2.1 Example

To motivate our work, we present a simplified example from the life of a motorist. The motorist is the owner of a car. To be allowed to drive the car on public roads, the car needs to pass a safety test, part of which is a tyre inspection. The law (in Germany) says that the car must have the same kind of tyre fitted to both the front and rear wheels.

In the state schema, the Boolean *flat* denotes whether any of the tyres are flat. If not the motorist is permitted to *drive* the car. The *Law* states that the *same* tyres should be used front and back. A single operation is specified, that of changing a tyre. Unfortunately, the spare tyre is of a different type, thus we will break the law as a result of a *Change*.

$$[CAR]$$

$$\begin{array}{|l}\multicolumn{1}{l}{\textit{State}}\\ \hline \textit{flat} : \mathbb{B}\\ \textit{drive} : \mathbb{B}\\ \textit{wheels} : \mathbb{N}\\ \hline \neg\, \textit{flat} \Rightarrow \textit{drive}\\ \textit{wheels} = 4\\ \hline\end{array}
\qquad
\begin{array}{|l}\multicolumn{1}{l}{\textit{Law}}\\ \hline \textit{same} : \mathbb{B}\\ \hline \textit{same}\\ \hline\end{array}
\qquad
\begin{array}{|l}\multicolumn{1}{l}{\textit{Change}}\\ \hline \Delta \textit{State}\\ \Xi \textit{Law}\\ x! : \mathbb{N}\\ \hline \textit{flat} \wedge \neg\, \textit{flat}'\\ \neg\, \textit{same}'\\ x! = \textit{wheels}\\ \hline\end{array}$$

The *Change* operation is clearly inconsistent in an intuitive sense. Once the tyre has been changed, the car is not allowed on the road by the law, because the type of tyre on at least one wheel is now different. However, we might wish to reason about aspects of this specification, for example, that the car is still driveable, since this only depends on the fact that no tyre is flat. Although this example is small and rather artificial, it illustrates the type of reasoning one might wish to perform.

Another operation often performed by a motorist is to refuel their car. We distinguish three kinds of cars: electric cars, cars with diesel engines and cars running on petrol. The electric car needs a power supply to re-charge, whereas the other cars need fuel which can be divided into unleaded, four star and diesel.

$$CAR\_TYPE ::= \textit{electric} \mid \textit{diesel} \mid \textit{petrol}$$
$$FUEL\_TYPE ::= \textit{unleaded} \mid \textit{four\_star} \mid \textit{diesel\_type}$$

$$\begin{array}{|l}\multicolumn{1}{l}{\textit{State2}}\\ \hline \textit{charged} : \mathbb{B}\\ \textit{fuel} : FUEL_TYPE\\ \textit{amount} : FUEL_TYPE \rightarrow \mathbb{N}\\ \hline\end{array}
\qquad
\begin{array}{|l}\multicolumn{1}{l}{\textit{Choose}}\\ \hline \Delta \textit{State2}\\ \textit{car}? : CAR_TYPE\\ \hline \textit{car}? = \textit{petrol} \Rightarrow \textit{fuel}' = \textit{unleaded}\\ \hline\end{array}$$

$$\begin{array}{|l}\multicolumn{1}{l}{\textit{Refuel}}\\ \hline \textit{Choose}\\ \hline (\textit{car}? = \textit{electric} \wedge \textit{charged}') \vee\\ (\textit{car}? = \textit{petrol} \wedge \textit{amount}'(\textit{fuel}') = 60 \wedge \textit{fuel}' = \textit{four\_star})\\ \hline\end{array}$$

This refuel operation is partly inconsistent, because we assign two different types of fuel to be taken when the car requires petrol. It is consistent when applied to electric cars; no refuel operation has been specified for diesel cars. Clearly, this looks like a simple specification error, but in a large specification such errors can be hidden.

### 2.2 Analysing Z Specifications

One of the benefits of formal methods is the ability to analyse specifications in a mathematical and logical way. In Z, this kind of analysis includes the calculation of the precondition of an operation and the inference of properties of the system before or after the application of an operation. Here, we review the analysis process in the context of inconsistent specifications.

**Precondition Calculation.** The precondition of an operation is the domain and inputs on which the operation is guaranteed to perform as specified. In Z, the precondition of an operation schema is implicit and therefore has to be extracted. Formally, the precondition of an operation schema $Op$ acting on a state schema $State$ and outputs $outs!$ is defined as:

$$\mathrm{pre}\, Op = \exists\, State', outs! \bullet Op$$

For operations containing inconsistencies, the precondition calculation determines, correctly, that this domain is either empty or partial. For example, the operation $Change$ described above cannot be successfully applied, and indeed we find:

$$\mathrm{pre}\, Change = [State;\ Law \mid \mathrm{false}]$$

The operation $Refuel$ is only partially applicable because of an inconsistency with the operation $Choose$ that forces unleaded petrol to be used for petrol cars:

$$\mathrm{pre}\, Refuel = [State2;\ car? : CAR \mid car? = electric]$$

**Inferring Properties.** In addition to calculating preconditions for operations, it is often desirable to verify that various properties hold for the system being constructed. For example, one might wish for certain system invariants to hold. In our example, one invariant is that there should always be four wheels attached to the car. For example, it can be shown that:

$$State \vdash wheels = 4$$

Using the completeness of the proof system of predicate logic this is equivalent to the statement:

$$State \nvdash wheels \neq 4$$

However, due to the inconsistency in the operation $Change$ it is possible to reason that there are more or less than four wheels present after the operation has been applied, in particular:

$$Change \vdash wheels' = 3$$

### 2.3 Inconsistencies in Z Specifications

A specification is supposed to be a model of some possible system. If such a specification is inconsistent then it has no models. (Saaltink, 1997) identifies two different types of inconsistency: *global* and *local inconsistency*.

**Global Inconsistency.** Global inconsistencies are serious, because they make an entire specification unsatisfiable. They occur if some axiom schema, generic schema, or constraint is unsatisfiable. Furthermore, they can arise due to a combination of different paragraphs of a specification, each being consistent. However, set declarations, abbreviations, and schema definitions cannot introduce global inconsistency. Note that no theorem can be trusted that has been proved in a globally inconsistent specification, as its proof is possibly based on a set of impossible assumptions.

**Local Inconsistency.** A schema can have an inconsistent, i.e. unsatisfiable, predicate. If such a schema is an operation schema, then the operation may not be applicable at all, or only parts of the operation are applicable. This is due to the fact that contradictions in an operation only restrict the precondition of that operation. In the case of the schema describing the state of the system, the entire part of the system governed by that state is not implementable. These kinds of errors are local in the sense that the specification of other components of the system may still be meaningful (although it is usually assumed implicitly, in a state and operation specification that at least one possible (initial) value of the state should exist).

**Inconsistency versus Falsity.** The distinction between falsity and inconsistency is not made clear within Z. An inconsistent operation behaves like an operation not being specified, i.e. set to false. This in turn makes it much harder to analyse the source of failure of an operation.

Additionally, the way the precondition computation in Z works seems to indicate an ordering of belief, assuming, for example, state schemas to be correct while an operation can be faulty. This leads to operations not being permitted if they are violating the state condition. However, this is not necessarily correct. It could be that the operation is correctly specified but the state specification is flawed.

For example, consider the tyre changing operation described above. The operation *Change* can be considered to be specified correctly. Practically, the specification of the schema *Law* lacks an exception, namely the allowance to drive a car with a replacement tyre of a different type with reduced speed and only to the next garage for the purpose of replacing it.

## 3 Handling Inconsistency using Quasi-Classical Logic

The aim of this work is to develop a way to continue to reason in the presence of inconsistency and to be able to infer valid conclusions from inconsistent Z schemas or specifications. In classical predicate logic, on which Z is based, inconsistent information result in triviality, because everything can be inferred from it. This, in turn, renders the information useless, when in fact there may be further valid inferences we wish to make. However, there are several ways of handling inconsistent information. One is to divide the pieces of information into (possibly maximal) consistent subsets (Rescher and Manor, 1970), another is paraconsistent reasoning. The latter allows the derivation of only non-trivial inferences from inconsistent information, i.e. not everything can be inferred.

A paraconsistent logic is a compromise of classical logic, either a weakening of the classical connectives, particularly negation, or of the inference system. The former often results in useful proof rules (like disjunctive syllogism: $\{\alpha, \neg\, \alpha \vee \beta\} \vdash \beta$) or intuitive equivalences (like $\neg\, \alpha \vee \beta \equiv \alpha \Rightarrow \beta$) to fail. To preserve the behaviour of the classical connectives, we consider a logic with a weaker inference system. We believe that such a paraconsistent logic is more suitable for our application, because specifiers and analysts will already be familiar with the notation and meaning of the connectives. Here, we present these ideas in a purely logical framework, independent of a specification notation. Subsequently, in Section 4, we will discuss how such a framework could be used within the Z notation.

### 3.1 The Idea Behind Quasi-Classical Logic

One representative of paraconsistent logics is quasi-classical logic (QCL), developed by (Besnard and Hunter, 1995). It follows the principle of moving away from the view of information being either true or false. We accept that we may have a number of perspectives on information and that these perspectives may contradict each other.

The key to QCL is that it allows only the derivation of information already present in a given theory, even though that theory might be inconsistent. In classical logic, the combination of disjunction introduction, inconsistency and disjunctive syllogism results in the fact that anything is derivable – not so in QCL. However, to cope with this, the proof theory of QCL is more restricted than the proof theory of classical logic.

The restriction imposed is that compositional proof rules (like disjunction introduction) cannot be followed by decompositional proof rules (like conjunction elimination). This results in a logic that is weaker than classical logic. However, an advantage of QCL is that the logical connectives behave classically. The aim of QCL is not so much to reason about the truth in the real world but about handling beliefs. This seems to be compliant with the idea of formal specification where we gather requirements of a system yet to be built.

To give a flavour of QCL we state the following classical derivations which are also derivations in QCL: $\{\alpha, \alpha \Rightarrow \beta\} \vdash_Q \beta$, $\{\neg\, \beta, \alpha \Rightarrow \beta\} \vdash_Q \neg\, \alpha$, as well as $\{\alpha \wedge \beta, \neg\, \beta \wedge \gamma\} \vdash_Q \gamma \vee \delta$, where $\vdash_Q$ denotes the QCL consequence relation. Further, we use the symbol $\vdash$ to denote the classical consequence.

In general, the classical properties of reflexivity ($\Delta \cup \{\alpha\} \vdash_Q \alpha$), monotonicity ($\Delta \vdash_Q \alpha$ implies $\Delta \cup \{\beta\} \vdash_Q \alpha$), and-introduction ($\Delta \vdash_Q \alpha$ and $\Delta \vdash_Q \beta$ implies $\Delta \vdash_Q \alpha \wedge \beta$), and or-elimination ($\Delta \cup \{\alpha\} \vdash_Q \gamma$ and $\Delta \cup \{\beta\} \vdash_Q \gamma$ implies $\Delta \cup \{\alpha \vee \beta\} \vdash_Q \gamma$) hold for QCL. Further, the following laws show some of the connections between classical logic and QCL:

- The property of consistency preservation holds:
  $\Delta \vdash_Q \alpha \wedge \neg \alpha$ implies $\Delta \vdash \alpha \wedge \neg \alpha$.
  In particular,
  $\Delta \vdash_Q \alpha$ implies $\Delta \vdash \alpha$.
- The property of supraclassicality fails:
  $\Delta \vdash \alpha$ does not imply $\Delta \vdash_Q \alpha$.
  Consider $\Delta$ to be empty, then it is possible to show in classical logic $\vdash \alpha \vee \neg \alpha$ but this does not hold in QCL.

Following the last example, we introduce some classical properties which are not feasible derivations in QCL including the derivations of the counter proofs:

- The property of right modus ponens fails:
  $\Delta \vdash_Q \alpha$ and $\Delta \vdash_Q \alpha \Rightarrow \beta$ does not imply $\Delta \vdash_Q \beta$.
  Consider $\Delta = \{\alpha, \neg \alpha\}$, then $\Delta \vdash_Q \alpha$, and $\Delta \vdash_Q \alpha \Rightarrow \beta$, but $\Delta \nvdash_Q \beta$.
- The property of cut fails:
  $\Delta \cup \{a\} \vdash b$ and $\Gamma \vdash a$ does not imply $\Delta \cup \Gamma \vdash b$.
  Consider that $\{\neg \alpha\} \cup \{\alpha \vee \beta\} \vdash_Q \beta$ and $\{\alpha\} \vdash_Q \alpha \vee \beta$, but $\{\alpha, \neg \alpha\} \nvdash_Q \beta$.
- The property of deduction fails:
  $\Delta \vdash a \Rightarrow b$ does not imply $\Delta \cup \{a\} \vdash b$.
  Consider $\Delta = \{\neg \alpha\}$, then $\Delta \vdash_Q \alpha \Rightarrow \beta$ but $\Delta \cup \{\alpha\} \nvdash_Q \beta$.

More examples of classical properties failing in QCL are given in (Hunter, 2000). In addition, we give a final example of the sort of reasoning from an inconsistent set of information facilitated by QCL: Given $\Delta = \{\alpha \vee \beta, \alpha \vee \neg \beta, \neg \alpha \wedge \delta\}$ possible consequences of $\Delta$ include $\alpha \vee \beta, \alpha \vee \neg \beta, \alpha, \beta, \neg \alpha$, and $\delta$ but not $\gamma, \neg \delta$, or $\phi \vee \psi$, though $\delta \vee \gamma$ would be possible.

### 3.2 Proof Theory of Quasi-Classical Logic

The proof theory for QCL's propositional part has been published by (Hunter, 2000). He also shows that QCL is sound and complete with respect to its semantics. Furthermore, a characterisation of the QCL consequence relation is given, separating those classical properties that do hold in QCL (like reflexivity) from those that do not. (Hunter, 2001) extends this work to first order and introduces a proof theory for QCL based on semantic tableaux.

A semantic tableau is a tree-like structure where nodes are labeled with formulae. The idea is that each branch represents the conjunction of the formulae appearing in it and that the tree itself represents the disjunction of its branches. We refer to (Smullyan, 1968) and (Fitting, 1996) who present a thorough overview of the techniques of the semantic tableaux method.

The semantic tableau proof procedure is based on refutation, i.e. to prove $X$ we begin with $\neg X$ and produce a contradiction. This is done by expanding $\neg X$ such that inessential details of its logical structure are removed until a contradiction appears or no expansion rule can be applied. Such expansion results in a tableau tree. For example, to prove the tautology $q \Rightarrow (p \Rightarrow q)$ we construct the following tree:

$$\begin{array}{c} \neg\,(q \Rightarrow (p \Rightarrow q)) \\ | \\ q, \neg\,(p \Rightarrow q) \\ | \\ q, p, \neg\, q \end{array}$$

and observe the contradiction between $q$ and $\neg\, q$. The tableau is closed and thus the tautology is proven.

However, this approach does not work directly for QCL since the truth and falsehood of a predicate are decoupled. Therefore, $q$ being satisfiable does not mean that $\neg\, q$ is not satisfiable, i.e. it is not possible to construct a contradiction in the same way as above. To overcome this obstacle Hunter introduces signed formulae denoted by $^*$, representing that a formula is unsatisfiable. Then showing $q$ and $q^*$ yields a refutation, as well as $\neg\, q$ and $(\neg\, q)^*$, because a formula cannot be satisfiable and unsatisfiable at the same time.

In the definition of the quasi-classical (QC) semantic tableau, there are two types of tableau expansion rules, the S-rules and the U-rules. All the S-rules assume the formula above the line to be satisfiable and the U-rules assume it to be unsatisfiable. Basically, the S-Rules correspond to the decompositional rules of QCL, and the U-rules are a variant of the compositional rules. Both types of expansion rules are defined in Appendix A, as well as further details, like the definitions of $\sim$ and $\otimes$.

A QC semantic tableau for a database $\Delta$ and a query $\alpha$ is a tree such that: (1) the formulae in $\Delta \cup \{\alpha^*\}$ are at the root of the tree; (2) each node of the tree has a set of signed formulae; and (3) the formulae at each node are generated by an application of one of the decomposition rules on a signed formula at ancestors of that node.

A QC tableau is closed if and only if every branch is closed. A branch is closed if and only if there is a formula $\beta$ for which $\beta$ and $\beta^*$ belong to that branch. A branch is open if there are no more rules that can be applied, and it is not closed. A tableau is open if there is at least one open branch.

Hunter showed that a set of assumptions $\Delta$ implies a query $\alpha$ by QCL denoted $\Delta \vdash_Q \alpha$, if and only if a QC tableau for $\Delta$ and query $\alpha$ is closed.

**Example.** To show that the disjunctive syllogism holds, i.e. $\{\alpha, (\neg\, \alpha \vee \beta)\} \vdash_Q \beta$ we construct the following closed tableau:

$$
\begin{array}{ccc}
 & \alpha, \neg\, \alpha \vee \beta, \beta^* & \\
\swarrow & & \searrow \\
(\sim \neg\, \alpha)^* & & \otimes(\neg\, \alpha \vee \beta, \neg\, \alpha) \\
| & & | \\
(\neg\, \neg\, \alpha)^* & & \beta \\
| & & \\
\alpha^* & &
\end{array}
$$

### 3.3 Non-derivable in QCL

Classical logicians may find that there is one major drawback to QCL, namely that it is not possible to show the classical tautologies from an empty set of assumptions. For example, the tautology $q \Rightarrow (p \Rightarrow q)$ as given above cannot be verified using QCL, e.g. the following tableau is not closed:

$$
\begin{array}{c}
(q \Rightarrow (p \Rightarrow q))^* \\
| \\
(\neg\, q \vee (p \Rightarrow q))^* \\
| \\
(\neg\, q)^*, (p \Rightarrow q)^* \\
| \\
(\neg\, q)^*, (\neg\, p \vee q)^* \\
| \\
(\neg\, q)^*, (\neg\, p)^*, q^*
\end{array}
$$

It is not possible to construct a refutation, because an unsatisfiable formula can only be decomposed into unsatisfiable formulae, hence, no contradiction with a satisfiable formula can be derived. Therefore, no tautology can be shown from the empty set of assumptions. However, it is not clear that it is a drawback for the application of QCL in the context of formal specification, because any derivation is based on a non-empty set of assumptions. Furthermore, when trying to prove a tautology the attempt of performing the proof will indicate a set of necessary assumptions. For example, to close the above tableau, we would need either $q$, $\neg\, q$ or $\neg\, p$ in the set of assumptions. In particular, the formula $q \vee \neg\, q$ is a good assumption to state, because the truth of $q$ does certainly not influence the tautology.

In the future, Hunter's work will have to be further extended to incorporate equality theory to be a truly alternative logic for Z. This can be done following (Fitting, 1996) or (Beckert, 1997). This is, however, a matter of further research and not the focus of the main issue we discuss here.

## 4 Reasoning in the Presence of Inconsistencies

The aim of this section is to demonstrate the use of quasi-classical logic to help us analyse possibly inconsistent Z specifications. QCL allows fewer properties to be derived from inconsistent specifications, in return for an increase in "usefulness" of the properties that can be derived. We illustrate this with the derivation of some properties from the example given in Section 2. We believe that those properties are intuitively valid while others should not hold in any case, not even due to inconsistencies. This approach will enable the specifier to validate the specification in more depth without being forced to remove inconsistencies immediately.

A schema in Z consists of a declaration part and a predicate. We assume the predicate to be type correct, i.e. it conforms to the declaration. We reason using QCL in a similar way to reasoning with the usual Z logic. For example, derivations using QCL include those of the form $Schema \vdash_Q p$ which has the same meaning as $Schema \vdash p$, except that QCL has been used in the derivation of the predicate $p$. (See (Woodcock and Davies, 1996) for a description of the formal meaning of $Schema \vdash p$.)

In our example, the operation *Change* conflicts with the schema *Law*. Despite this, we can show that after changing the tyre four wheels are connected to the car.

$$Change \vdash_Q wheels' = 4$$

This follows directly from the root of the tableau (omitting the type definitions):

$$\begin{array}{c} \neg\, flat \Rightarrow drive, wheels = 4, \neg\, flat' \Rightarrow drive', wheels' = 4, same, \\ same', flat \wedge \neg\, flat', \neg\, same', x! = wheels, (wheels' = 4)^* \end{array}$$

However, we cannot derive anymore that:

$$Change \vdash_Q wheels' = 3$$

or that there are any other number of wheels apart from four. Therefore we know that there will be exactly four wheels on our car after changing one tyre.

Similarly, it follows from the root of the tableau that no tyre is flat after applying the operation *Change*:

$$Change \vdash_Q \neg\, flat'$$

Furthermore, the tyres are not flat and, hence, it is possible to drive the car:

$$Change \vdash_Q drive'$$

The proof of this follows from the tree below, where we simplify the tree by removing unnecessary detail from the root of the tree (this simplification, however, does not affect the proof itself).

$$\begin{array}{ccc} & same', \neg\, same', \neg\, flat' \Rightarrow drive', flat \wedge \neg\, flat', (drive')^* & \\ & | & \\ & flat, \neg\, flat' & \\ & | & \\ & \neg\,\neg\, flat' \vee drive' & \\ & | & \\ & flat' \vee drive' & \\ / & & \backslash \\ (\sim flat')^* & & \otimes(flat' \vee drive', flat') \\ | & & | \\ (\neg\, flat')^* & & drive' \end{array}$$

This tree is closed and the proof is, therefore, complete. Observe that the contradiction of $same'$ could not contribute to the proof.

We also introduced the operations *Refuel* and *Choose* which contradicted each other in the choice of petrol for a petrol car. However, we can still infer that the car will be full with petrol, i.e. $amount'(fuel') = 60$ at the end of the *Refuel* operation, or it will be *charged* if it is an electric car:

$$Refuel \vdash_Q amount'(fuel') = 60 \lor charged'$$

The QCL tableau to show this is:

$$\begin{array}{c}
car? = petrol \Rightarrow fuel' = unleaded, \\
(car? = electric \land charged') \lor \\
(car? = petrol \land amount(fuel') = 60 \land fuel' = four_star), \\
(amount'(fuel') = 60 \lor charged')^* \\
| \\
(car? = electric \land charged') \lor car? = petrol, \\
(car? = electric \land charged') \lor (amount'(fuel') = 60 \land fuel' = four_star) \\
| \\
(car? = electric \land charged') \lor amount'(fuel') = 60, \\
(car? = electric \land charged') \lor fuel' = four_star \\
| \\
car? = electric \lor amount'(fuel') = 60, \\
charged' \lor amount'(fuel') = 60
\end{array}$$

$$\begin{array}{cc}
\diagup & \diagdown \\
charged' & amount'(fuel') = 60 \\
| & | \\
(amount'(fuel') = 60)^*, (charged')^* & (amount'(fuel') = 60)^*, (charged')^*
\end{array}$$

Quasi-classical logic allows one to derive non-trivial conclusions from inconsistent information. We demonstrated the application of QCL by analysing an inconsistent Z specification in terms of derivations of properties we considered important to verify.

On the one hand, quasi-classical inferences are a subset of those possible by classical logic. Therefore, using QCL as an underlying logic for Z will not introduce undesired logical consequences. On the other hand, QCL cannot deal with classical tautologies from the empty set of assumptions. Otherwise, QCL yields the same consequences as classical reasoning in the context of consistent specifications. This means that the specifier has to make all assumptions explicit, a task anyway enforced by formal specification development. In further research we will investigate whether QCL's consequence relation can replace the classical consequence relation in Z.

## 5 Preconditions in the Presence of Inconsistencies

The precondition of an operation is the predicate that has to be fulfilled to apply the operation successfully. In Z specifications such preconditions are implicit, i.e. they have to be calculated from the operation. The calculation leads to two situations: either the operation can be applied successfully or not. However, allowing for inconsistencies in specifications we will distinguish three possibilities:

(1) the operation can be applied consistently, i.e. it is defined,
(2) the operation can be applied but is inconsistent, i.e. it is overdefined,
(3) the operation cannot be applied, i.e. it is undefined.

This distinction breaks the situation in Z that an inconsistently specified operation is, in terms of the results of calculating the precondition, the same as an operation that has not been specified. We wish to explore whether this division is more useful when analysing specifications and whether we can use it to support further development (like refinement) without resolving inconsistencies immediately.

For example, the precondition of the *Refuel* operation introduced in Section 2 can be divided into the following three categories: (1) $type(car?) = electric$, (2) $type(car?) = petrol$, and (3) any other situation, which amounts to $type(car?) = diesel$. Observe that the normal precondition $pre$ covers (1), so $\neg\ pre$ is the combination of (2) and (3). The problem we investigate here is how to calculate the regions (2), and consequently (3).

Actually, we concentrate on a slight modification of the above problem. We calculate the combination of the defined and overdefined region. Both approaches presented below use the known definition of the precondition, i.e.

$$\mathrm{pre}\ Op = \exists\ State', outs! \bullet Op$$

but apply a different set of simplification rules to this abstract formula. To obtain the three precondition regions we proceed the following way: Use standard logic to determine the defined region ($\mathrm{pre}_d\ Op$). Use the classical precondition definition but apply a restricted set of simplification rules to determine the combination of the defined and overdefined region, i.e. $\mathrm{pre}_{comb}\ Op = \mathrm{pre}_d\ Op \lor \mathrm{pre}_{od}\ Op$ and construct the intersection of both to obtain only the overdefined region. The undefined area is then the complement of the combined region, i.e. $\neg\ \mathrm{pre}_{comb}\ Op$.

### 5.1 The Two-Point Rule.

One possible way to determine the alternative precondition regions is to use the standard Z logic, but to restrict the inference of false when the operation defines inconsistent after states, i.e. the contradiction law will not be applied to after states. The reasoning process is then continued by distributing the inconsistency. The aim is to determine where the operation was intended to work, rather than where it would be applicable classically.

In standard Z logic, one of the fundamental rules to simplify the precondition of an operation is the One-Point-Rule (Woodcock and Davies, 1996, p.48):

$$\frac{\exists\, x : S \bullet (p(x) \land x = t)}{t \in S \land p(t)} \quad \left[\begin{array}{l}\text{One-Point Rule (OPR),}\\ x \text{ not free in } t\end{array}\right]$$

However, when applied to an inconsistent predicate it always results in *false*. For example:

$$\begin{array}{ll}
 & \text{pre}\, \mathit{Refuel} \\
\equiv & \{\text{Definition of pre + Schema Expansion}\} \\
 & \exists\, \mathit{charged}', \mathit{amount}', \mathit{fuel}' \bullet \\
 & \quad \mathit{car?} = \mathit{petrol} \Rightarrow \mathit{fuel}' = \mathit{unleaded} \land ((\mathit{car?} = \mathit{electric} \land \mathit{charged}') \lor \\
 & \qquad\qquad (\mathit{car?} = \mathit{petrol} \land \mathit{amount}'(\mathit{fuel}') = 60 \land \mathit{fuel}' = \mathit{four_star})) \\
\equiv & \{\exists\text{-Distribution}\} \\
 & \exists\, \mathit{amount}', \mathit{fuel}' \bullet \mathit{car?} = \mathit{petrol} \Rightarrow \mathit{fuel}' = \mathit{unleaded} \land \\
 & \qquad ((\exists\, \mathit{charged}' \bullet (\mathit{car?} = \mathit{electric} \land \mathit{charged}')) \lor \\
 & \qquad (\mathit{car?} = \mathit{petrol} \land \mathit{amount}'(\mathit{fuel}') = 60 \land \mathit{fuel}' = \mathit{four_star})) \\
\equiv & \{\text{OPR on } \mathit{charged}' + \text{Rewrite}\} \\
 & \mathit{car?} = \mathit{electric} \land \exists\, \mathit{fuel}' \bullet (\neg\, \mathit{car?} = \mathit{petrol} \lor \mathit{fuel}' = \mathit{unleaded}) \lor \\
 & \qquad \exists\, \mathit{amount}', \mathit{fuel}' \bullet ((\neg\, \mathit{car?} = \mathit{petrol} \lor \mathit{fuel}' = \mathit{unleaded}) \land \\
 & \qquad\qquad \mathit{car?} = \mathit{petrol} \land \mathit{amount}'(\mathit{fuel}') = 60 \land \mathit{fuel}' = \mathit{four_star}) \\
\equiv & \{\text{OPR on } \mathit{fuel}' \text{ twice}\} \\
 & (\mathit{car?} = \mathit{electric} \land \neg\, \mathit{car?} = \mathit{petrol}) \lor \\
 & \qquad \exists\, \mathit{amount}' \bullet (\neg\, \mathit{car?} = \mathit{petrol} \lor \mathit{four_star} = \mathit{unleaded}) \land \\
 & \qquad\qquad\qquad \mathit{car?} = \mathit{petrol} \land \mathit{amount}'(\mathit{four_star}) = 60 \\
\equiv & \{\text{Contradiction + Domain Knowledge}\} \\
 & (\mathit{car?} = \mathit{electric} \land (\mathit{car?} = \mathit{diesel} \lor \mathit{car?} = \mathit{electric})) \\
\equiv & \{\text{Absorption}\} \\
 & \mathit{car?} = \mathit{electric}
\end{array}$$

In this derivation, the information that the postconditions of *Refuel* and *Choose* are contradictory for $car? = petrol$ is lost. For this reason we introduce a variant of the OPR that preserves this information and allows one to continue to reason despite the presence of inconsistencies.

Looking at the forward direction of the OPR we actually want to distribute substitution through the predicate $p$. Therefore, we require:

$$\frac{\exists\, x : S \bullet (p(x) \land x = t_1 \land x = t_2)}{t_1 \in S \land t_2 \in S \land p(t_1) \land p(t_2)} \quad \left[\begin{array}{l}\text{Two-Point Rule (2PR),}\\ x \text{ not free in } t_1 \text{ or } t_2\end{array}\right]$$

Applying the 2PR can result in a consistent predicate even though it was inconsistent before, i.e. $p(t_1) \land p(t_2)$ is satisfiable though $t_1 = t_2$ is not. For example, given $p(x) = x \leq 5$, $t_1 = 3$, and $t_2 = 4$ then $3 = 4$ is not satisfiable, but $3 \leq 5 \wedge 4 \leq 5$ is.

Applying the 2PR to our example above we reason:

$$
\begin{array}{l}
\quad \vdots \\
\quad car? = electric \wedge \exists fuel' \bullet (\neg\, car? = petrol \vee fuel' = unleaded) \vee \\
\quad \exists\, amount', fuel' \bullet ((\neg\, car? = petrol \vee fuel' = unleaded) \wedge \\
\qquad car? = petrol \wedge amount'(fuel') = 60 \wedge fuel' = four\_star) \\
\equiv \{\text{OPR on } fuel' \text{ once + Rewrite}\} \\
\quad (car? = electric \wedge \neg\, car? = petrol) \vee \exists\, amount', fuel' \bullet \\
\quad\; (\neg\, car? = petrol \wedge car? = petrol \wedge \\
\qquad\qquad\qquad amount'(fuel') = 60 \wedge fuel' = four\_star) \vee \\
\quad\; (fuel' = unleaded \wedge car? = petrol \wedge \\
\qquad\qquad\qquad amount'(fuel') = 60 \wedge fuel' = four\_star) \\
\equiv \{\text{Contradiction}\} \\
\quad (car? = electric \wedge \neg\, car? = petrol) \vee \exists\, amount', fuel' \bullet car? = petrol \wedge \\
\quad\; fuel' = unleaded \wedge amount'(fuel') = 60 \wedge fuel' = four\_star \\
\Rrightarrow \{\text{2PR on } fuel'\} \\
\quad (car? = electric \wedge \neg\, car? = petrol) \vee \exists\, amount' \bullet \\
\quad\; car? = petrol \wedge amount'(unleaded) = 60 \wedge amount'(four\_star) = 60 \\
\equiv \{\text{OPR on } amount'\} \\
\quad (car? = electric \wedge \neg\, car? = petrol) \vee car? = petrol \\
\equiv \{\text{Rewrite + Contradiction}\} \\
\quad car? = electric \vee car? = petrol
\end{array}
$$

Informally, we are using the 2PR as follows. If $t_1$ and $t_2$ are equivalent in 2PR, this results in the forward direction of the OPR. Otherwise, we have an inconsistent situation, because $x$ cannot take more than one value. In such cases, we split the predicate $p$ into two and substitute $t_1$ in one and $t_2$ in the other occurrence of $p$. This might distribute inconsistency to the several instances of $p$, but it may also remove inconsistency.

As with the One-Point rule, the 2PR is applied to remove the existential quantification from a predicate. In the case of a consistent predicate, we preserve information by applying the One-Point rule, which is indeed an equivalence operation. The Two-Point rule is applied when we have an inconsistency, i.e. "too much information". In this situation, we are interested in reducing the amount of information, possibly removing the inconsistency, and thus the 2PR is applied only in one direction. Using the 2PR in the reverse direction would introduce information, which would be inappropriate. For example, consider the predicate $3 \leq 5 \wedge 4 \leq 5$, we do not infer $\exists x \bullet x \leq 5 \wedge x = 3 \wedge x = 4$ because this introduces an inconsistency.

In the case where the after state of an operation is functionally determined by the before state, the use of the 2PR enables us to determine where the operation was intended to be applied, even though the actual definition may be inconsistent. For example, the defined region of *Refuel*, $\mathrm{pre}_d\, \mathit{Refuel}$, has the predicate $car? = electric$, and the combination of the defined and overdefined region, $\mathrm{pre}_{comb}\, \mathit{Refuel}$, has the predicate $car? = electric \lor car? = petrol$. The overdefined region, $\mathrm{pre}_{od}\, \mathit{Refuel}$, is, therefore, given by the predicate $car? = petrol$.

### 5.2 Using Quasi-Classical Logic's Equivalences

In the previous subsection we considered the use of classical logic with restrictions to support precondition calculation. Quasi-classical logic has already been successfully applied in Section 4 to reason in the presence of inconsistency. Therefore, it seems natural to ask how QCL can be used to simplify the precondition of an inconsistent operation. In Appendix B we state some laws of quasi-classical logic, mainly equivalence laws, which will be applied subsequently.

**Example.** Again, we calculate the precondition of the operation *Refuel*, this time using the QCL equivalences. We abbreviate the declared names by their first letter, with the exception of *four_star* being denoted $4^*$. The derivation then proceeds as follows:

$$
\begin{array}{l}
\quad \exists\, c', a', f' \bullet c? = p \Rightarrow f' = u \land \\
\qquad ((c? = e \land c') \lor (c? = p \land a'(f') = 60 \land f' = 4^*)) \\
\equiv_Q \{\exists\text{-Distribution}\} \\
\quad \exists\, a', f' \bullet (c? = p \Rightarrow f' = u) \land \\
\qquad ((\exists\, c' \bullet c? = e \land c') \lor (c? = p \land a'(f') = 60 \land f' = 4^*)) \\
\equiv_Q \{\text{Idempotency of } c',\ \text{OPR on } c',\ \text{One Law for Equality}\} \\
\quad \exists\, a', f' \bullet (c? = p \Rightarrow f' = u) \land \\
\qquad (c? = e \lor (c? = p \land a'(f') = 60 \land f' = 4^*)) \\
\equiv_Q \{\land\text{-Distribution},\ \exists\text{-Distribution}\} \\
\quad (\exists f' \bullet (c? = p \Rightarrow f' = u) \land c? = e) \lor \\
\quad (\exists\, a', f' \bullet (c? = p \Rightarrow f' = u) \land (c? = p \land a'(f') = 60 \land f' = 4^*)) \\
\equiv_Q \{\land\text{-Distribution},\ \exists\text{-Distribution}\} \\
\quad (c? = e \land \neg\, c? = p) \lor (\exists f' \bullet c? = e \land f' = u) \lor \\
\quad (\exists\, a', f' \bullet (c? = p \Rightarrow f' = u) \land (c? = p \land a'(f') = 60 \land f' = 4^*)) \\
\equiv_Q \{\text{Idempotency of } f',\ \text{OPR on } f',\ \text{One Law for Equality}\} \\
\quad (c? = e \land \neg\, c? = p) \lor c? = e \lor \\
\quad (\exists\, a', f' \bullet (c? = p \Rightarrow f' = u) \land (c? = p \land a'(f') = 60 \land f' = 4^*)) \\
\equiv_Q \{\text{OPR on } f',\ \text{One Law for Equality}\} \\
\quad (c? = e \land \neg\, c? = p) \lor c? = e \lor
\end{array}
$$

$$(\exists\, a' \bullet ((c? = p \Rightarrow 4^* = u) \wedge c? = p \wedge a'(4^*) = 60)$$

$$\equiv_Q \{\text{OPR on } a', \text{ One Law for Equality}\}$$

$$(c? = e \wedge \neg\, c? = p) \vee c? = e \vee ((c? = p \Rightarrow 4^* = u) \wedge c? = p)$$

$$\equiv_Q \{\text{Absorption, Implication Law}\}$$

$$c? = e \vee (c? = p \wedge 4^* = u)$$

i.e.

$\text{pre}_{comb}\, \mathit{Refuel} =_Q (\mathit{car}? = \mathit{electric} \vee (\mathit{car}? = \mathit{petrol} \wedge \mathit{four\_star} = \mathit{unleaded}))$.

We interpret this result as follows. The operation *Refuel* is applicable if the given car is an electric car, or it is a petrol car but four star and unleaded are the same. Note, this calculation used laws of equality which have not been incorporated into QCL yet. We therefore emphasise that the final form of this proof will depend on the exact shape of a modified logic $\text{QCL}_=$, i.e. quasi-classical logic with equality.

We separate the defined area, i.e. $\text{pre}_d\, \mathit{Refuel}$, from the result of the above derivation and determine the overdefined area:

$$\text{pre}_{od}\, \mathit{Refuel} =_Q (\mathit{car}? = \mathit{petrol} \wedge \mathit{four\_star} = \mathit{unleaded}).$$

The overdefined area derived using quasi-classical equivalences is different to the one derived using the Two-Point rule. It provides more information on the source of inconsistency, e.g. that unleaded should be the same as four star.

# 6 Conclusion and Future Work

## 6.1 Conclusion

The aim of this work was to discuss how to reason in the presence of inconsistent Z schemas and still to be able to derive useful information from them. We introduced quasi-classical logic to support the process of non-trivial reasoning despite inconsistencies. We replaced the classical proof system by QCL's version and derived properties from our example specification. We also indicated that certain undesired inferences are not possible anymore and, therefore, demonstrated the usefulness of this approach. However, we only considered local inconsistencies, although this method can be extended to deal with global inconsistency as well.

We also decided to split the precondition of an operation into three areas: the defined, overdefined and undefined one. This enhanced separation of the precondition allows the analyst to investigate where an operation has been overdefined more clearly. We assume that this distinction will be beneficial in a theory of refinement to be developed later. We introduced two possible ways to determine these regions. However, using either way resulted in a combination of the defined and overdefined regions. This is different to the classical approach which does not distinguish between the overdefined and undefined area. Combining classical precondition calculation with our results enabled us to separate all three regions.

Conclusion drawn from inconsistent specifications using QCL are useful in the sense that they only represent information present in the specification rather than information introduced due to triviality. It is clearly understood that inconsistencies need to be removed at some stage of the development process but it may well be beneficial to delay such step. QCL itself is very similar to classical logic apart from its treatment of inconsistencies. Classical consistent specifications can be analysed using QCL almost without any changes. However, inconsistent specifications need a more subtle treatment due to the failure of several inference rules in the presence of inconsistency. In particular, to represent *false* the specifier cannot use a contradiction anymore.

### 6.2 Related Work

There are three categories of related work we will discuss: previous publications on handling inconsistency in Z, the problems of managing inconsistency in other specification languages, and work on other paraconsistent logics to handle inconsistencies in a more general setting.

Present literature on handling inconsistency in Z is largely concerned with preventing and eliminating inconsistencies, rather than managing them. For example, (Valentine, 1998) presents "sufficient conditions and stylistic guidelines for achieving [consistency]" and proofs for the success of this approach. This work is based on the assumption that "consistency is essential for a Z specification to have any useful meaning". This is correct in the standard approach to Z. However, considering non-standard interpretations, this is not necessarily the case anymore. Our work shows that it is possible to derive useful conclusions from inconsistent Z specifications.

(Arthan, 1991) describes work in progress on a high integrity proof tool for Z specifications. One concern is the inconsistency of sets of axiomatic and generic Z schemas, i.e. global inconsistency. He proposes a rule of *conservative extensions* allowing new objects to be defined only if an appropriate consistency condition has been proved. If not, the new objects will be redefined such that no inconsistency occurs. The approach of (Arthan, 1991) could be adapted to local inconsistent schemas by weakening their schema property to *true* so that no conclusions can be drawn from a locally inconsistent predicate. Although this avoids the problem of triviality, it comes at a loss of information, whereas our approach is based on the information present in a specification.

The work by (Saaltink, 1997) is concerned with the analysis of Z specifications using the automated theorem prover Eves. Consistency checking is one possible analysis that can be performed. Further, preconditions calculated using Z/Eves remove all information based on inconsistencies. This work is related to finding inconsistencies, rather than inconsistency management.

In the introduction to this paper we presented links to the current work on managing inconsistencies in the software development process in general terms. However, it seems impossible to find specific work on inconsistency management for any particular specification language, like B (Abrial, 1996), VDM, LOTOS, CSP, or others. All formal approaches on living with inconsistencies deal with the use of (non-standard) predicate logic as specification language.

Work on developing paraconsistent logics is also relevant. We refer to (Hunter, 1998) and (Batens et al., 2000) for a brief overview. Batens' inconsistency-adaptive logics (Batens, 1999; Batens, 2000) are a set of paraconsistent logics that aim at handling consistent theories exactly like classical logic but to be adaptive to inconsistency and not to infer everything from it. Its dynamic proof theory, however, is not as close to common logical reasoning as the proof theory of QCL. Further, work on applying paraconsistent logics to mathematics (Mortensen, 1995; da Costa, 2000) may be of value to the work on inconsistency management in Z because of Z's foundation in set theory.

### 6.3 Future Work

One major motivation for this work is the belief in a theory that allows continued development of specifications despite the presence of inconsistencies. Refinement is one of the processes of specification development from an abstract form to a more concrete representation. Refinement is also the process of adding information. This can, however, lead to the introduction of inconsistencies. The idea behind the alternative precondition regions is to support refinement in the presence of overdefinedness. Current investigations suggest that a combination of classical and quasi-classical refinement rules can support detection and controlled removal of inconsistencies. However, this relation requires further investigation.

A theory of refinement in the presence of inconsistency will then contribute to work on viewpoint specifications (Boiten et al., 1999), where the unification of two or more viewpoints is defined as the least common refinement of the viewpoints. So far, the verification of this property also contains a consistency check between the viewpoint specifications. However, this forces removal of the inconsistency to unify the viewpoints. Our aim is to support viewpoint unification and the analysis of the resulting specification without necessarily removing the inconsistency.

In Section 3 we mentioned that we need to enhance quasi-classical logic to be a true contender for an alternative paraconsistent logic for Z. An essential extension to QCL is the incorporation of a theory of equality.

In connection to our previous work on un(der)definedness in Z (Miarka et al., 2000), it seems worthwhile to investigate further the duality of un(der)definedness and overdefinedness, i.e. inconsistency. Also, we identified in that piece of work that inconsistency issues can arise between the schema components. It will be interesting to see whether our current work can be beneficial to our previous work and whether both can be combined.

Though the work presented here is mainly concerned with local inconsistencies, we also mentioned the problem of global inconsistencies. It will be interesting to investigate the application of paraconsistent logics, like QCL, to develop a schema calculus which is more robust in the presence of inconsistencies.

**Acknowledgement:** We like to thank Anthony Hunter for the discussions about QCL. Further, we acknowledge all the anonymous referees for their corrections and helpful comments to improve this work.

## References

Abrial, J.-R. (1996). *The B-Book: Assigning Programs to Meanings.* Cambridge University Press.

Arthan, R. D. (1991). Formal Specification of a Proof Tool. In Prehn, S. and Toetenel, H., editors, *Proceedings of Formal Software Development Methods (VDM '91)*, Lecture Notes in Computer Science 552, pages 356–370, Berlin, Germany. Springer.

Balzer, R. (1991). Tolerating Inconsistency. In *Proceedings of the 13th International Conference on Software Engineering*, pages 158–165. IEEE Computer Society Press / ACM Press.

Batens, D. (1999). Inconsistency-Adaptive Logics. In Orlowska, E., editor, *Logic at Work. Essays Dedicated to the Memory of Helena Rasiowa*, Studies in fuzziness and soft computing, Volume 24, pages 445–472, Heidelberg, New York. Physica-Verlag.

Batens, D. (2000). A Survey of Inconsistency-Adaptive Logics. In (Batens et al., 2000), pages 49–73.

Batens, D., Mortensen, C., Priest, G., and Bendegem, J.-P. V., editors (2000). *Frontiers of Paraconsistent Logic*. Research Studies Press Ltd., Baldock, Hertfordshire, England.

Beckert, B. (1997). Semantic Tableaux with Equality. *Journal of Logic and Computation*, 7(1):39–58.

Besnard, P. and Hunter, A. (1995). Quasi-Classical Logic: Non-Trivializable Classical Reasoning from Inconsistent Information. In Froidevaux, C. and Kohlas, J., editors, *Proceedings of the ECSQARU European Conference on Symbolic and Quantitive Approaches to Reasoning and Uncertainty*, Lecture Notes in Artificial Intelligence 946, pages 44–51, Berlin. Springer Verlag.

Boiten, E. A., Derrick, J., Bowman, H., and Steen, M. W. A. (1999). Constructive consistency checking for partial specification in Z. *Science of Computer Programming*, 35(1):29–75.

da Costa, N. C. (2000). Paraconsistent Mathematics. In (Batens et al., 2000), pages 165–179.

Finkelstein, A., Gabbay, D., Hunter, A., Kramer, J., and Nuseibeh, B. (1994). Inconsistency Handling in Multi-Perspective Specifications. *IEEE Transactions on Software Engineering*, 20(8):569–578.

Fitting, M. C. (1996). *First-Order Logic and Automated Theorem Proving*. Graduate Texts in Computer Science. Springer-Verlag, Berlin, 2nd edition.

Ghezzi, C. and Nuseibeh, B. (1998). Guest Editorial: Introduction to the Special Section: Managing Inconsistency in Software Development. *IEEE Transactions on Software Engineering*, 24(11):906–907.

Hunter, A. (1998). Paraconsistent Logic. In Besnard, P. and Hunter, A., editors, *Reasoning with Actual and Potential Contradictions*, Volume II of Handbook of Defeasible Reasoning and Uncertain Information (Gabbay, D. and Smets, Ph., editors), pages 13–44. Kluwer Academic Publishers, Dortrecht, The Netherlands.

Hunter, A. (2000). Reasoning with Contradictory Information Using Quasi-Classical Logic. *Journal of Logic and Computation*, 10(5):677–703.

Hunter, A. (2001). A Semantic Tabeau Version of First-Order Quasi-Classical Logic. In Benferhat, S. and Besnard, P., editors, *Symbolic and Quantitative Approaches to Reasoning with Uncertainty, Proceedings of the 6th European Conference, ECSQARU 2001, Toulouse, France*, Lecture Notes in Artificial Intelligence 2143, pages 544–555. Springer Verlag.

Hunter, A. and Nuseibeh, B. (1997). Analysing Inconsistent Specifications. In *Proceedings of the 3rd International Symposium on Requirements Engineering (RE'97)*, pages 78–86. Annapolis, USA, IEEE Computer Society Press.

Hunter, A. and Nuseibeh, B. (1998). Managing Inconsistent Specifications: Reasoning, Analysis, and Action. *ACM Transactions on Software Engineering and Methodology*, 7(4):335–367.

Miarka, R., Boiten, E., and Derrick, J. (2000). Guards, Preconditions, and Refinement in Z. In Bowen, J. P., Dunne, S., Galloway, A., and King, S., editors, *ZB2000: Formal Specification and Development in Z and B / First International Conference of B and Z Users*, Lecture Notes in Computer Science 1878, pages 286–303, Berlin Heidelberg New York. Springer-Verlag Berlin.

Mortensen, C. (1995). *Inconsistent Mathematics.* Kluwer Academic Publishers Group, Dordrecht, The Netherlands.

Rescher, N. and Manor, R. (1970). On Inference from Inconsistent Premisses. *Theory and Decision*, 1:179–217.

Saaltink, M. (1997). The Z/EVES User's Guide. Technical Report TR-97-5493-06, ORA Canada, 267 Richmond Road, Suite 100, Ottawa, Canada.

Schwanke, R. W. and Kaiser, G. E. (1988). Living with Inconsistency in Large Systems. In *Proceedings of the International Workshop on Software Version and Configuration Control*, pages 98–118, Grassau, Germany.

Smullyan, R. M. (1968). *First-Order Logic*, Volume 43 of *Ergebnisse der Mathematik und ihrer Grenzgebiete.* Springer-Verlag, New York.

Valentine, S. H. (1998). Inconsistency and Undefinedness in Z – A Practical Guide. In Bowen, J. P., Fett, A., and Hinchey, M. G., editors, *ZUM'98: The Formal Specification Notation*, Lecture Notes in Computer Science 1493, pages 233–249, Berlin Heidelberg New York. Springer Verlag.

Woodcock, J. and Davies, J. (1996). *Using Z - Specification, Refinement, and Proof.* Prentice Hall International Series in Computer Science. Prentice Hall Europe. Online: `http://softeng.comlab.ox.ac.uk/usingz/` (last access 18/10/2001).

## Appendix A: The QCL Expansion Rules

In this appendix we introduce the tableaux expansion rules for quasi-classical logic (see Section 3). The rules are divided into S-rules and U-rules. The S-rules consider a formula above the line as satisfiable, whereas the U-rules consider a formula above the line as unsatisfiable. The presentation of these rules is preceded by the definition of necessary terminology.

Given a language $\mathcal{L}$, the set of tableaux of all formula over $\mathcal{L}$ is denoted $\mathcal{T}(\mathcal{L})$. Let $\alpha$ be an atom and $\sim$ a complementation operation such that $\sim \alpha$ is $\neg\, \alpha$ and $\sim (\neg\, \alpha)$ is $\alpha$. The abbreviation $\otimes(\alpha_1 \vee \ldots \vee \alpha_n, \alpha_i)$ is defined as the clause obtained by removing $\alpha_i$ from $\alpha_1 \vee \ldots \vee \alpha_n$.

The following are the S-rules for QC semantic tableaux, where $t$ is in $\mathcal{T}(\mathcal{L})$ and $t'$ is in $\mathcal{T}(\mathcal{L})$ but not occurring in the tableau constructed so far. The $|$ symbol denotes the introduction of a branch point in the QC semantic tableau.

The disjunction S-rules:

$$\frac{\alpha_1 \vee \ldots \vee \alpha_n}{(\sim \alpha_i)^* \mid \otimes(\alpha_1 \vee \ldots \vee \alpha_n, \alpha_i)} \text{[where } \alpha_1, \ldots, \alpha_n \text{ are literals]}$$

$$\frac{\alpha_1 \vee \ldots \vee \alpha_n}{\alpha_1 \mid \ldots \mid \alpha_n} \text{[where } \alpha_1, \ldots, \alpha_n \text{ are literals]}$$

The rewrite S-rules:

$$\frac{\alpha \wedge \beta}{\alpha, \beta} \qquad \frac{\neg\, \neg\, \alpha \vee \gamma}{\alpha \vee \gamma} \qquad \frac{\neg\, (\alpha \wedge \beta) \vee \gamma}{\neg\, \alpha \vee \neg\, \beta \vee \gamma}$$

$$\frac{\neg\, (\alpha \vee \beta) \vee \gamma}{(\neg\, \alpha \wedge \neg\, \beta) \vee \gamma} \quad \frac{\alpha \vee (\beta \wedge \gamma)}{(\alpha \vee \beta) \wedge (\alpha \vee \gamma)} \quad \frac{\alpha \wedge (\beta \vee \gamma)}{(\alpha \wedge \beta) \vee (\alpha \wedge \gamma)}$$

The quantification S-rules:

$$\frac{(\forall X \bullet \alpha(X)) \vee \gamma}{\alpha(t) \vee \gamma} \quad \frac{(\neg\, \exists X \bullet \alpha(X)) \vee \gamma}{\neg\, \alpha(t) \vee \gamma}$$

$$\frac{(\exists X \bullet \alpha(X)) \vee \gamma}{\alpha(t') \vee \gamma} \quad \frac{(\neg\, \forall X \bullet \alpha(X)) \vee \gamma}{\neg\, \alpha(t') \vee \gamma}$$

The first disjunction S-rule links to the following U-rules for the QC semantic tableaux, where $t$ is in $\mathcal{T}(\mathcal{L})$ and $t'$ is in $\mathcal{T}(\mathcal{L})$ but not occurring in the tableau constructed so far. The $|$ symbol denotes the introduction of a branch point in the QC semantic tableau.

The disjunction U-rule: $\dfrac{(\alpha \vee \beta)^*}{\alpha^*, \beta^*}$

The conjunction U-rule: $\dfrac{(\alpha \wedge \beta)^*}{\alpha^* \mid \beta^*}$

The rewrite U-rules:

$$\frac{(\neg\, \neg\, \alpha \vee \gamma)^*}{(\alpha \vee \gamma)^*} \quad \frac{(\neg\, (\alpha \wedge \beta) \vee \gamma)^*}{(\neg\, \alpha \vee \neg\, \beta \vee \gamma)^*} \quad \frac{(\neg\, (\alpha \vee \beta) \vee \gamma)^*}{((\neg\, \alpha \wedge \neg\, \beta) \vee \gamma)^*}$$

The quantification U-rules:

$$\frac{((\forall X \bullet \alpha(X)) \vee \gamma)^*}{(\alpha(t') \vee \gamma)^*} \quad \frac{((\neg\, \exists X \bullet \alpha(X)) \vee \gamma)^*}{(\neg\, \alpha(t') \vee \gamma)^*}$$

$$\frac{((\exists X \bullet \alpha(X)) \vee \gamma)^*}{(\alpha(t) \vee \gamma)^*} \quad \frac{((\neg\, \forall X \bullet \alpha(X)) \vee \gamma)^*}{(\neg\, \alpha(t) \vee \gamma)^*}$$

Please note that $\gamma$ can be "empty". Furthermore, we will add the following two rewrite rules: the S-rule for implication, and the U-rule for implication:

$$\frac{(\alpha \Rightarrow \beta) \vee \gamma}{(\neg\, \alpha \vee \beta) \vee \gamma} \qquad \frac{((\alpha \Rightarrow \beta) \vee \gamma)^*}{((\neg\, \alpha \vee \beta) \vee \gamma)^*}$$

Rules for equality: $(x = x)^*$ closes a branch

$$\frac{\begin{array}{c} x = y \\ \alpha(x) \end{array}}{\alpha(y)} \qquad \frac{\begin{array}{c} x = y \\ \alpha(y) \end{array}}{\alpha(x)} \qquad \frac{(x = y \wedge \alpha(x))^*}{(\alpha(y))^*} \qquad \frac{(x = y \wedge \alpha(y))^*}{(\alpha(x))^*}$$

## Appendix B: Quasi-Classical Laws

Here we present a list of laws which are valid in QCL. We use the notation $a \dashv\vdash_Q b$ to denote that $\{a\} \vdash_Q b$ and $\{b\} \vdash_Q a$. These equivalences can be proved using the tableaux rules from Appendix A.

Commutativity:
- $a \vee b \dashv\vdash_Q b \vee a$
- $a \wedge b \dashv\vdash_Q b \wedge a$

De Morgan Laws:
- $\neg\,(a \wedge b) \dashv\vdash_Q (\neg\, a) \vee (\neg\, b)$
- $\neg\,(a \vee b) \dashv\vdash_Q (\neg\, a) \wedge (\neg\, b)$

Idempotent Laws:
- $a \vee a \dashv\vdash_Q a$
- $a \wedge a \dashv\vdash_Q a$

Double Negation Law:
- $a \dashv\vdash_Q \neg\,\neg\, a$

One Law for Equality:
- $a \wedge x = x \dashv\vdash_Q a$

One-Point Rule:
- $(\exists\, x \bullet p(x) \wedge x = t) \dashv\vdash_Q p(t)$

Associativity:
- $(a \vee b) \vee c \dashv\vdash_Q a \vee (b \vee c)$
- $(a \wedge b) \wedge c \dashv\vdash_Q a \wedge (b \wedge c)$

Distributivity:
- $a \vee (b \wedge c) \dashv\vdash_Q (a \vee b) \wedge (a \vee c)$
- $a \wedge (b \vee c) \dashv\vdash_Q (a \wedge b) \vee (a \wedge c)$

Absorption Laws:
- $a \vee (a \wedge b) \dashv\vdash_Q a$
- $a \wedge (a \vee b) \dashv\vdash_Q a$

Implication Laws:
- $(a \Rightarrow b) \wedge (a \Rightarrow c) \dashv\vdash_Q a \Rightarrow (b \wedge c)$
- $(a \Rightarrow b) \vee (a \Rightarrow c) \dashv\vdash_Q a \Rightarrow (b \wedge c)$
- $(b \Rightarrow a) \vee (c \Rightarrow a) \dashv\vdash_Q (b \wedge c) \Rightarrow a$
- $(b \Rightarrow a) \wedge (c \Rightarrow a) \dashv\vdash_Q (b \vee c) \Rightarrow a$
- $a \wedge (a \Rightarrow b) \dashv\vdash_Q a \wedge b$

Some Quantification Laws:
- $\exists\, x \bullet (a(x) \vee b(x)) \dashv\vdash_Q (\exists\, x \bullet a(x)) \vee (\exists\, x \bullet b(x))$
- $\exists\, x \bullet (a(x) \wedge b) \dashv\vdash_Q (\exists\, x \bullet a(x)) \wedge b$