On the Consequences of Acting in the Presence of Inconsistency

Bashar Nuseibeh Alessandra Russo

Department of Computing, Imperial College
180 Queen's Gate, London SW7 2BZ, UK
Email: (ban, ar3}@doc.ic.ac.uk

Abstract

Managing inconsistency in specifications covers a range of activities from consistency checking and inconsistency analysis to inconsistency handling through action. In this paper we argue that inconsistency analysis is insufficient to determine the choice of actions to take in the presence of inconsistency. Rather, we propose that some form of 'hypothetical reasoning' is needed in order to determine the consequences of different actions and thereby facilitate the decision-making process. We suggest some logic-based techniques and associated heuristics for analysing the consequences of acting in the presence of inconsistency.

1. Inconsistencies in specifications

Deciding what action to take in the presence of inconsistency is difficult. Most researchers agree that eradicating inconsistency in specifications is a desirable and worthy goal, but there is an emerging view that it may also be acceptable to live with inconsistency in certain circumstances or for transient periods of time [4]. Whichever strategy one decides to adopt in dealing with inconsistency in specifications, the choice of what action to take and the consequences of taking such an action are, we believe, crucial.

Previously, we focused on analysing inconsistent specifications with a view that the results of such analysis will shed some light on what action to take to remove the inconsistency or to ameliorate the inconsistent specification [2; 6]. While this has indeed been helpful in suggesting possible actions to take in the presence of inconsistency, we have also found that the analysis does not always suggest which action to take, given a choice between alternatives. In this paper, we propose the study of the *consequences* of taking alternative actions by studying their impact on the inconsistent specification, thereby facilitating the inconsistency handling decision process. We believe that a contribution in this area provides a way of managing changes in evolving specifications, by providing means of analysing the consequences of making these changes.

2. Inconsistency implies action

Figure-1 is a schematic illustration of our framework for managing inconsistency in software specifications. It is an elaboration of the inconsistency handling framework proposed in [5], based on the notion of 'inconsistency implies action'. The idea is that a specification, S, may contain a number, n, of inconsistencies (which can be detected in various ways; e.g., using logic-based consistency checking). A number, m, of alternative actions can then be identified to deal with each inconsistency. These actions are normally determined by analysing the inconsistency. The choice of which action to perform is determined by analysing the consequences that each alternative action has on the original specification (including any human factors that may complement or override formal analysis).

Figure-1 also suggests a particular kind of impact analysis based on so-called 'derivable information'. Informally, a specification's derivable information is any piece of information that can be obtained through some inference process from that specification. By examining the relationship, R, between the derivable information, D(S), from the original specification S, and the derivable information, $D(S'_m)$, from each of the possible consequent specifications, a 'measure' of impact of different actions may be obtained. Note that each consequent specification, S'_m, is simply the specification that would be obtained by performing one of the alternative actions, A_m.

In the remainder of this paper, we will focus on actions and their consequences. Clearly, consistency checking and inconsistency analysis are still active and important research areas, however, the focus of this paper is on activities that take place after an analysis has been performed. While there is a body of related work on impact analysis (e.g., [1]) and decision-making (e.g., [8]), we attempt to provide in this paper some formal foundations to underpin further work in both these areas.

3. Consequences of action

One of our main aims is to provide guidance to developers faced with inconsistent specifications and alternative courses of action. The provision of guidance

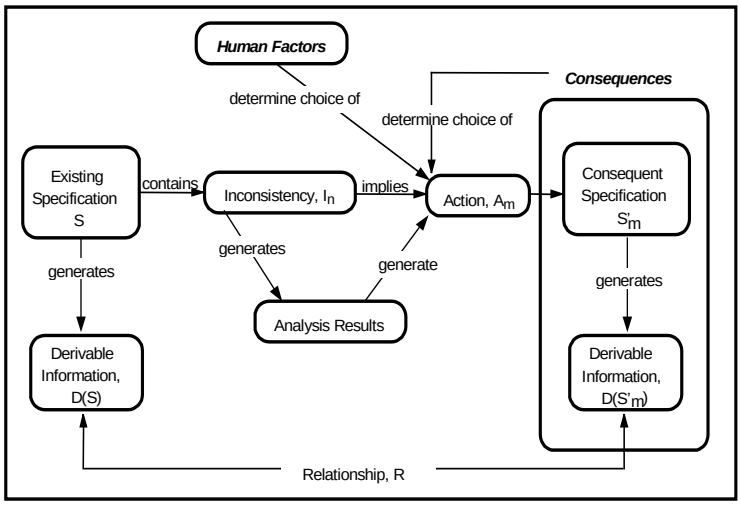

Figure-1: A framework for managing inconsistency in software specification.

requires that we be able to *reason* about the consequences of alternative development actions, so our suggested approach has a logical flavour - although we do not commit ourselves to any particular logic.

Actions. We use the term *action* to mean any update to a given specification. Actions may be 'atomic' or 'composite'. Atomic actions either *add* or *delete* a single piece of information (well-formed formula) to or from a specification, respectively. Composite actions are actions that can be reduced to a sequence of atomic actions. For example, the action of 'replacing a fact X by Y' in a specification is a composite action that is equivalent to 'delete X, then add Y'. In this paper, we focus on atomic actions to illustrate our framework, but we believe that our approach can be extended to deal with composite actions as well.

Atomic actions can be used to implement different strategies for acting in the presence of inconsistency, such as those we identified in earlier work [7]. For example, *ignoring* an inconsistency is equivalent to taking no action, while *ameliorating* an inconsistent specification involves adding or deleting one or more pieces of information to/from the specification without necessarily *removing* all the inconsistencies. Of course, taking no action means leaving a specification unchanged, and this may indeed be a desirable course of action to take. However, the kind of analysis we are considering in this paper assumes that actions will change a specification in some way, and it is the nature (consequence) of this change that we are focusing on. In the end, it is the user who will decide whether to perform particular actions or to take no action at all. Moreover, it is often the case that more than one atomic action is needed to eliminate an inconsistency, and that a specification may contain more then one inconsistency. The aim of our approach, however, is to ensure that each atomic action generates a new specification which, although may still leave the specification inconsistent, 'improves' it in some way.

Derivable Information. In general, the actions we are dealing with are not arbitrary updates [3] to a specification. Rather, they are focussed actions that specifically address inconsistencies identified by previous analysis. Our intention in this work, is to determine a course of actions based on analysis of, and reasoning about, the consequences of possible alternative actions. For this purpose, we introduce the notion of 'derivable information' which can be inferred from a specification.

Intuitively, derivable information from a specification is information that is either already in the specification, or that may be inferred through some reasoning process from that specification. As we said earlier, while we do not commit to any particular kind of logic, we do assume that our logic does not allow trivialisation[1]. In this way, we can continue reasoning in the presence of inconsistency [6] to capture information that is still derivable from the (changed) specification.

It is worth noting at this point that an inconsistency is itself a particular kind of derivable information. A specification is said to be inconsistent if a contradiction ($\alpha \wedge \neg\alpha$) can be inferred from it. We denote the set of all inconsistencies derivable from a specification S by DI(S), where DI(S) is a subset of the set of information derivable from S, D(S).

Consequences. Performing an action on a specification S produces a new specification S', which we call a *consequent specification* (Figure-1). We define the notion of *consequences* of an action in terms of both S and S'. For example, in some cases the consequence of an action

1 *Trivialisation* is the derivation of any arbitrary information from an inconsistency.

is *more* derivable information from S' than S, while in other cases *less* information is derivable. We define this more precisely as follows.

Let S be a formal specification, let A_m be an action on S, and let S'_m be the specification resulting from performing the action A_m on S. We define the positive consequences of A_m as the set $D(S'_m) - D(S)$ of new derivable information from S'_m, and we define the negative consequences of A_m as the set $D(S) - D(S'_m)$ of derivable information from S that are no longer derivable from the new specification S'_m.

3. Choosing the right action

There may be a number of different ways for choosing what action to take in order to handle inconsistencies in specifications. One option is to perform a *comparative* analysis of the positive and negative consequences of each individual action, and then make a choice based on some 'desirable' consequences. Since the reality of requirements specification means that, inevitably, changes to partial specifications lead to the introduction of further inconsistencies and the loss of information, we may, for example, want to *minimise additional inconsistency* (that is, minimise inconsistent positive consequences) and/or *minimise loss of information* (that is, minimise negative consequences).

One way to achieve this is to define an ordering relation between alternative actions in terms of their positive and negative consequences. This ordering provides us with a 'measure' of inconsistency in a specification, which in turn provides us with further guidance in deciding what action(s) to take. Thus, for example, we could prescribe that maintaining or reducing the number of additional inconsistencies is preferable to minimising loss of information. Therefore, one action would be preferable to another if it introduces fewer new inconsistencies to a specification, or if, given the same number of new inconsistencies, it causes less loss of information. Within this scenario, two different actions may well be equally preferable. This is the case if a pair of different actions have the same consequences (i.e. they introduce the same number of additional inconsistencies, and result in the same loss of information). In such cases, human intervention is necessary.

Of course, there may be other heuristics for choosing between alternative actions, such as choosing actions that remove more than one inconsistency at the expense of losing information from the specification.

4. On the consequences of living with inconsistency

We aim to develop a general (logic-based) approach to inconsistency handling in specifications, and to apply the results to the management of evolving requirements specifications. Thus, *generating* alternative inconsistency handling actions remains an open issue. We intend to address this by considering abductive ('hypothetical') reasoning. Abduction can help identify, for each inconsistency, which facts to add and/or delete in order to resolve the inconsistency.

Moreover, we intend to develop techniques for analysing the consequences of *sequences* of atomic actions (that is, the consequences of composite actions). For example, we would like to examine the notion of 'equivalent' sequences of actions, and assess the 'costs' of following different alternative sequences, even if they generate the same consequent specifications.

We believe that the scope of this work is wider than inconsistency handling alone. It addresses the problems of analysing and handling change in evolving specifications. What we have proposed in this paper is a preliminary impact analysis of inconsistent specifications. An important next step is to develop tractable tools that can guide developers engaged in this activity. We believe that restructuring large specifications into more modular 'viewpoints' [9] can help us address some issues of scale.

Acknowledgements

We would like to thank Jeff Kramer for his feedback, and to acknowledge financial support from the UK EPSRC (GR J15483 & GR/L 55964), the British Council and the EU.

References

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

[2] S. Easterbrook and B. Nuseibeh, "Managing Inconsistencies in an Evolving Specification", *Proc. of 2nd IEEE Int. Symp. in Req. Eng. (RE '95)*, York, 48-55, 1995.

[3] R. Fagi, J.D. Ullman and M.Y. Vardi, "On the semantics of updates in databases", *Proc. of 2nd ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, Atlanta, 352-265, 1983.

[4] S. Fickas, M. Feather and J. Kramer (Eds.), *Proc. of ICSE-97 Workshop on 'Living with Inconsistency'*, Boston, USA, May 1997.

[5] A. Finkelstein, D. Gabbay, A. Hunter, J. Kramer, and B. Nuseibeh, "Inconsistency Handling in Multi-perspective Specifications", *IEEE TSE*, 20(8): 569-578, 1994.

[6] A. Hunter and B. Nuseibeh, "Analysing Inconsistent Specifications", *Proc. of 3rd Int. Symp. on Requirements Engineering*, 78-86, Annapolis, MD, USA, January 1997.

[7] B. Nuseibeh, "To be and not to be: On managing inconsistency in software development", *Proc. of the 8th IEEE International Workshop on Software Specification and Desig*n (IWSSD-8), 164-169, Germany, 1996.

[8] W.N. Robinson, "Interpreting Multiple Specifications Using Domain Goals", *Proc. of 5th IEEE Int. Workshop on Soft. Spec. and Design* (IWSSD-5), 219-225, 1989.

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