3

Semantics of Quasi-Boolean Expressions

A. Bijlsma

Introduction

In deriving programs, it is often useful to consider certain expressions as well-defined although they contain subexpressions to which no value can reasonably be attributed. For instance, one wishes to consider the expression

\[ 0 \leq i < n \quad \text{and} \quad a[i] = 0 \]

as false when \( i = n \), without worrying whether or not \( n \) is in the subscript range of array \( a \). Expressions like these are called quasi-boolean in [3]. Traditionally, they are treated by introducing the conditional connectives \texttt{cand} and \texttt{cor} [2, chapter 4] [4, §4.1].

However, \texttt{cand} and \texttt{cor} do not satisfy pleasant algebraic laws: they are not commutative and do not in general distribute over each other. This makes calculations involving these connectives exceedingly burdensome; for this reason their introduction was termed “a strategic mistake” in [3]. It is the purpose of this note to suggest a possible alternative.

Let it be clear from the outset that the semantics of quasi-boolean expressions defined below is not intended for implementation in a programming language. Its aim is merely to facilitate the calculational derivation of programs. That these derivations are most easily expressed in a richer language than the programs themselves, one that need not be implementable, is a well-known observation.

On the other hand, in order to be suited for its purpose, any proposed semantics of quasi-boolean expressions should satisfy the following criteria:

(i) if all atomic subexpressions of a certain expression are well-defined, the value attached to that expression should be the one that would follow from ordinary logic;
(ii) no expression should be left undefined if it can be given a meaning by the introduction of conditional connectives;

(iii) the relation "having the same value" between expressions should satisfy simple calculational laws.

The option of considering an expression undefined when any of its subexpressions is, satisfies (i) and (iii) but not (ii). The use of conditional connectives satisfies (i) and (ii) but not (iii). It is possible to do a bit better by considering two-sided conditional connectives [5] [1, §A0.3.2]. This restores commutativity, but, as we shall see in the next section, there remain important laws that fail to hold. The semantics defined in this note has the property that all laws of propositional calculus carry through.

Conventions and notations

We suppose a set \( At \) of logical atoms and a valuation \( \alpha v \in At \rightarrow \{F, U, T\} \) given: this is meant to model the fact that formulae without logical connectives can be false, undefined or true respectively. For convenience’s sake it is assumed that \( F \) and \( T \) do not themselves belong to \( At \). The problem dealt with here is how to extend the given valuation \( \alpha v \) if logical connectives are introduced. Let \( Ex \) be the set of all finite expressions that can be formed with the elements of \( At \cup \{F, T\} \), the unary operator \( \text{non} \) and the binary operators \( \text{and} \) and \( \text{or} \). As yet, no meaning is attached to the elements of \( Ex \); in fact, discovering how to define such a meaning in a sensible way is precisely our purpose. (The formal operators \( \text{non} \), \( \text{and} \), \( \text{or} \) must not be confused with those of the ordinary logic in which our proofs will be expressed. These will be denoted by the symbols \( \neg, \land, \lor \).)

We introduce a linear ordering \( \sqsubseteq \) on the set \( \{F, U, T\} \) such that \( F \sqsubseteq U \sqsubseteq T \). For \( p \) and \( q \) in \( \{F, U, T\} \), denote by \( p \cap q \) the minimum and by \( p \sqcup q \) the maximum of \( p \) and \( q \) with respect to the ordering \( \sqsubseteq \). Furthermore, the prefix operator \( \sim \) is defined by \( \sim F = T, \sim U = U, \sim T = F \).

For \( v \in At \rightarrow \{F, U, T\} \), we define \( c.v \in Ex \rightarrow \{F, U, T\} \) by

\[
\begin{align*}
(0) \quad E \in \{F, T\} & \Rightarrow c.v.E = E, \\
(1) \quad E \in At & \Rightarrow c.v.E = v.E, \\
(2) \quad c.v.(\text{non } E) & = \sim(c.v.E), \\
(3) \quad c.v.(E0 \ \text{and} \ E1) & = c.v.E0 \cap c.v.E1, \\
(4) \quad c.v.(E0 \ \text{or} \ E1) & = c.v.E0 \sqcup c.v.E1
\end{align*}
\]

for all \( E, E0, E1 \in Ex \).

What we are looking for is a mapping \( ev \in Ex \rightarrow \{F, U, T\} \) with