Quasi-boolean equivalence

A. Bijlsma
Department of Mathematics and Computing Science
Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, The Netherlands

January, 1993

Keywords Formal semantics, undefinedness.

1 Introduction

In program derivations, sometimes boolean expressions occur that one would like to consider as well-defined although they contain subexpressions to which no value can reasonably be attributed. An example is

$$0 \leq 0 \ \vee \ 0/0 > 0 \quad ,$$

which ought to have the value *true* despite the fact that the second disjunct is undefined. Such expressions, sometimes called *quasi-boolean* [7], are traditionally handled by the conditional connectives **cand** and **cor** [6] [9]. However, these do not satisfy pleasant algebraic laws and their use tends to complicate calculations to an unacceptable degree.

In the present paper, we show that the problem of quasi-boolean expressions may be safely ignored. That is to say, the rules for constructing equational proofs in the style of [8] may be applied as if all subexpressions were well-defined, and there is nothing wrong with, for instance, a proof step

$$\begin{array}{ll} & (0 \leq 0 \ \ \& \ \ \neg \, 0/0 > 0) \ \vee \ 0/0 > 0 \\ \equiv & \quad \{\text{complement rule}\} \\ & 0 \leq 0 \ \vee \ 0/0 > 0 \quad . \end{array}$$

In order to see why this is so, we employ the semantics of quasi-boolean expressions defined in [3], the main points of which will be recapitulated in the next section. This semantics generalizes the conditional connectives in the following sense: if an expression can be made well-defined by the use of **cand** and **cor**, possibly after permutation of terms, it is well-defined in our semantics and denotes the same value. The opposite does not hold. Moreover, if two expressions are equivalent according to the laws of propositional calculus, they denote the same value.

2 Expression semantics

(This section may be skipped by readers familiar with [3].) We suppose a set At of logical *atoms* and a *valuation* $av \in At \to \{F, U, T\}$ given: this is meant to model the fact that formulae without logical connectives can be false, undefined or true respectively. For convenience, it is assumed that F and T do not themselves belong to At. The problem dealt with here is how

to extend valuation av 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 **non** and the binary operators **and**, **or** and **eq**. As yet, no meaning is attached to the elements of Ex; in fact, defining such a meaning in a sensible way is precisely our goal. (The formal operators **not**, **and**, **or**, **eq** must not be confused with those of the ordinary logic in which our proofs will be expressed, which are denoted by the symbols $\neg, \wedge, \vee, \equiv$.)

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

For $v \in At \to \{F,U,T\}$, define $c.v \in Ex \to \{F,U,T\}$ by

$$
\begin{aligned}
E \in \{F,T\} &\;\Rightarrow\; c.v.E = E \quad , && (1)\\
E \in At &\;\Rightarrow\; c.v.E = v.E \quad , && (2)\\
c.v.(\textbf{non}\ E) &\;=\; \sim c.v.E \quad , && (3)\\
c.v.(E0\ \textbf{and}\ E1) &\;=\; c.v.E0 \sqcap c.v.E1 \quad , && (4)\\
c.v.(E0\ \textbf{or}\ E1) &\;=\; c.v.E0 \sqcup c.v.E1 \quad , && (5)\\
c.v.(E0\ \textbf{eq}\ E1) &\;=\; (c.v.E0 \sqcap c.v.E1) \sqcup (\sim c.v.E0 \sqcap \sim c.v.E1) && (6)
\end{aligned}
$$

for all $E, E0, E1 \in Ex$.

We are looking for a mapping $ev \in Ex \to \{F,U,T\}$ with the properties announced in the previous section. An obvious candidate for ev would be $c.av$; this choice is made in [2], [1], [10] and has a history going back as far as Łukasiewicz in the 1920s [5]. However, we are not satified with this solution since, for instance,

$$c.av.(E0\ \textbf{and}\ (\textbf{non}\ E0\ \textbf{or}\ E1)) = c.av.(E0\ \textbf{and}\ E1)$$

fails to hold.

Here is a different approach. For $E0, E1 \in Ex$ we define

$$E0 == E1 \;\equiv\; (\forall w : w \in At \to \{F,T\} \;\&\; f.w : c.w.E0 = c.w.E1) \quad , \tag{7}$$

where $f.w$ means that w maps every atom with a defined boolean value to that value, i.e.,

$$f.w \;\equiv\; (\forall x : x \in At \;\&\; av.x \neq U : w.x = av.x) \quad . \tag{8}$$

(Actually, the $\wedge$ in (8) should be a **cand**, since $av.x$ is undefined when $x \in At$ is false. However, it is the purpose of this paper to show that such scruples are unnecessary.) For $E \in Ex$, we now define

$$ev.E = \begin{cases} F & \text{if } E == F \quad , \\ T & \text{if } E == T \quad , \\ U & \text{otherwise} \quad . \end{cases} \tag{9}$$

For the heuristics underlying this definition and proofs of its properties, the reader is referred to [3].

3 Various notions of equivalence

Valuation ev defined in (9) respects the laws of propositional calculus in the sense that, for instance,

$$ev.(E0 \textbf{ and } (\textbf{non } E0 \textbf{ or } E1)) = ev.(E0 \textbf{ and } E1) \quad . \tag{10}$$

Identities like (10) already make it possible, at least up to a point, to forget about undefined values: we may interpret the proof step quoted in the Introduction as

$$\begin{array}{ll} & ev.((0 \leq 0 \textbf{ and non } 0/0 > 0) \textbf{ or } 0/0 > 0) \\ = & \quad \{(10)\} \\ & ev.(0 \leq 0 \textbf{ or } 0/0 > 0) \quad . \end{array}$$

However, this approach only works for straight-line derivations. A problem with the translation occurs when a previously proved equivalence is to be used in the next stage. To take the simplest case, what if the next derivation uses

$$\begin{array}{ll} & (0 \leq 0 \ \ \& \ \ \neg\, 0/0 > 0) \ \vee \ 0/0 > 0 \ \ \equiv \ \ 0 \leq 0 \ \vee \ 0/0 > 0 \\ \equiv & \quad \{\text{previous derivation}\} \\ & true \ \ ? \end{array}$$

Define the relation $\cong$ between expressions by

$$E0 \cong E1 \ \ \equiv \ \ ev.(E0 \textbf{ eq } E1) = T \quad . \tag{11}$$

Then the result proved in our first derivation is of the form $ev.E0 = ev.E1$, whereas the result used in the second one is of the form $E0 \cong E1$. We are thus led to ask whether

$$ev.E0 = ev.E1 \ \Rightarrow \ E0 \cong E1 \quad ,$$

and, unfortunately, this is not the case. As a counterexample, consider distinct x and y in At satisfying $av.x = av.y = U$. Then $ev.x = ev.y = U$, yet $E0 \ncong E1$.

In view of this difficulty, we would like to be able to interpret equivalences in a derivation as the relation $\cong$. This, however, poses several problems. In the first place, we do not yet know whether $\cong$ is transitive, and without transitivity it is impossible to draw any conclusion from a derivation. In the second, we do not yet know whether $\cong$ satisfies the rules of propositional calculus in the sense that

$$E0 \textbf{ and } (\textbf{non } E0 \textbf{ or } E1) \cong E0 \textbf{ and } E1 \tag{12}$$

and so on. In the third, we do not yet know whether such an interpretation would retain the possibility of deducing the equality of ev-values, that is, whether

$$E0 \cong E1 \ \Rightarrow \ ev.E0 = ev.E1 \quad . \tag{13}$$

In the next section we prove a theorem that yields positive answers to all three questions.

4 The main theorem

Theorem 1 *For* $E0, E1 \in Ex$ *,*

$$E0 \cong E1 \ \ \equiv \ \ E0 == E1 \quad .$$

Proof In this proof, we omit the domain of dummy w, which is always

$$w \in At \to \{F,T\} \ \& \ f.w \quad . \tag{14}$$

Observe that for such w we have $c.w \in Ex \to \{F,T\}$. Now for $E0, E1 \in Ex$,

$$\begin{array}{ll} & E0 \cong E1 \\ \equiv & \quad \{\text{definition (11) of } \cong\} \\ & ev.(E0 \ \mathbf{eq} \ E1) = T \\ \equiv & \quad \{\text{definition (9) of } ev\} \\ & E0 \ \mathbf{eq} \ E1 == T \\ \equiv & \quad \{\text{definition (7) of } ==\} \\ & (\forall w :: c.w.(E0 \ \mathbf{eq} \ E1) = c.w.T) \\ \equiv & \quad \{(1)\} \\ & (\forall w :: c.w.(E0 \ \mathbf{eq} \ E1) = T) \\ \equiv & \quad \{(6)\} \\ & (\forall w :: (c.w.E0 \sqcap c.w.E1) \sqcup (\sim c.w.E0 \sqcap \sim c.w.E1) = T) \quad , \end{array}$$

so

$$E0 \cong E1 \ \equiv \ (\forall w :: (c.w.E0 \sqcap c.w.E1) \sqcup (\sim c.w.E0 \sqcap \sim c.w.E1) = T) \quad . \tag{15}$$

Now for $x, y \in \{F,U,T\}$ we have

$$\begin{array}{ll} & (x \sqcap y) \sqcup (\sim x \sqcap \sim y) = T \\ \equiv & \quad \{\, p \sqcup q = T \ \equiv \ p = T \ \vee \ q = T \,\} \\ & x \sqcap y = T \ \vee \ \sim x \sqcap \sim y = T \\ \equiv & \quad \{\, p \sqcap q = T \ \equiv \ p = T \ \& \ q = T \,\} \\ & (x = T \ \& \ y = T) \ \vee \ (\sim x = T \ \& \ \sim y = T) \\ \equiv & \quad \{\, \sim x = T \ \equiv \ x = F \,\} \\ & (x = T \ \& \ y = T) \ \vee \ (x = F \ \& \ y = F) \\ \equiv & \quad \{\text{substitute second conjunct into first one, twice}\} \\ & (x = y \ \& \ y = T) \ \vee \ (x = y \ \& \ y = F) \\ \equiv & \quad \{\text{distribution of } \wedge \text{ over } \vee\} \\ & x = y \ \& \ (y = T \ \vee \ y = F) \\ \equiv & \quad \{\, y \in \{F,U,T\} \,\} \\ & x = y \ \& \ y \neq U \quad . \end{array}$$

Hence

$$(x \sqcap y) \sqcup (\sim x \sqcap \sim y) = T \ \equiv \ x = y \ \& \ y \neq U \quad . \tag{16}$$

Therefore

$$\begin{array}{ll} & E0 \cong E1 \\ \equiv & \quad \{\text{substitution of (16) into (15)}\} \\ & (\forall w :: c.w.E0 = c.w.E1 \ \& \ c.w.E1 \neq U) \\ \equiv & \quad \{\, c.w \in Ex \to \{F,T\} \,\} \\ & (\forall w :: c.w.E0 = c.w.E1) \\ \equiv & \quad \{\text{definition (7) of } ==\} \\ & E0 == E1 \quad . \end{array}$$

□

This does indeed answer the three questions posed at the end of the previous section, since the desired properties of $\cong$ have been proved for $==$ in [3].

Yet another consequence of Theorem 1 is

$$E0 \cong E1 \quad \equiv \quad E0 \textbf{ eq } E1 \cong T \quad . \tag{17}$$

To see this, transform the right hand side of (11) with the aid of definition (9) of ev and apply Theorem 1.

We now return to the examples given before and show how our results may be used. The derivation

$$\begin{array}{ll} & (0 \leq 0 \ \& \ \neg\, 0/0 > 0) \ \vee \ 0/0 > 0 \\ \equiv & \quad \{\text{complement rule}\} \\ & 0 \leq 0 \ \vee \ 0/0 > 0 \end{array}$$

should be interpreted as

$$\begin{array}{ll} & (0 \leq 0 \textbf{ and non } 0/0 > 0) \textbf{ or } 0/0 > 0 \\ \cong & \quad \{(12)\} \\ & 0 \leq 0 \textbf{ or } 0/0 > 0 \quad , \end{array}$$

and the following derivation

$$\begin{array}{ll} & (0 \leq 0 \ \& \ \neg\, 0/0 > 0) \ \vee \ 0/0 > 0 \quad \equiv \quad 0 \leq 0 \ \vee \ 0/0 > 0 \\ \equiv & \quad \{\text{previous derivation}\} \\ & \mathit{true} \end{array}$$

should be interpreted as

$$\begin{array}{ll} & ((0 \leq 0 \textbf{ and non } 0/0 > 0) \textbf{ or } 0/0 > 0) \textbf{ eq } (0 \leq 0 \textbf{ or } 0/0 > 0) \\ \cong & \quad \{(17), \text{ using the result of the previous derivation}\} \\ & T \quad . \end{array}$$

5 On Leibniz's rule

The results of the previous section allow us to import a previously proved equivalence into a derivation, but so far only as a full line, not as a subformula. To give a simple case once again, what if our example derivations are followed by

$$\begin{array}{ll} & (0 \leq 0 \ \& \ \neg\, 0/0 > 0) \ \vee \ 0/0 > 0 \ \vee \ 0 \geq 0 \quad \equiv \quad 0 \leq 0 \ \vee \ 0/0 > 0 \ \vee \ 0 \geq 0 \\ \equiv & \quad \{\text{Leibniz, using previous derivation}\} \\ & \mathit{true} \quad ? \end{array}$$

One is tempted to state a version of Leibniz's rule for the relation $\cong$ in the form

$$E0 \cong E1 \quad \Rightarrow \quad \varphi.E0 \cong \varphi.E1 \tag{18}$$

for all φ in $Ex \to Ex$. However, this is too strong a demand. For instance, we might define $\varphi.E$ to be T if the number of occurrences of **non** in expression E is precisely 2, and to be F otherwise. Then $\varphi.(\textbf{non non } T) = T$ and $\varphi.T = F$, yet clearly $\textbf{non non } T \cong T$.

The counterexample shows that in (18) we must restrict ourselves to mappings between expressions that 'respect the semantics'; however, the simplest formulation of what this means is (18) itself. Hence we choose a different tack and show that the operators of the object language fall into this class, as does substitution.

Theorem 2 *For* $E0, E1 \in Ex$ *such that* $E0 \cong E1$ *,*

$$\mathbf{non}\ E0 \ \cong\ \mathbf{non}\ E1 \tag{19}$$
$$E0\ \mathbf{or}\ E2 \ \cong\ E1\ \mathbf{or}\ E2 \tag{20}$$
$$E0\ \mathbf{and}\ E2 \ \cong\ E1\ \mathbf{and}\ E2 \tag{21}$$
$$E0\ \mathbf{eq}\ E2 \ \cong\ E1\ \mathbf{eq}\ E2 \ . \tag{22}$$

Proof Note that (21) and (22) follow from (19) and (20), since we have already observed that $\cong$ follows the laws of propositional calculus.

To prove (19), we let w be constrained by (14) once again and write

$\mathbf{non}\ E0 \cong \mathbf{non}\ E1$
$\equiv$ {Theorem 1}
$\mathbf{non}\ E0 == \mathbf{non}\ E1$
$\equiv$ {definition, (7), of $==$}
$(\forall w :: c.w.(\mathbf{non}\ E0) = c.w.(\mathbf{non}\ E1))$
$\equiv$ {(3)}
$(\forall w :: \sim c.w.E0 = \sim c.w.E1)$
$\Leftarrow$ {Leibniz}
$(\forall w :: c.w.E0 = c.w.E1)$
$\equiv$ {(7); Theorem 1}
$E0 \cong E1$.

The proof of (20) is similar, merely using (5) instead of (3). It is left to the reader, but can be found in [4].

□

The example at the beginning of this section may now be interpreted as

$((0 \leq 0\ \mathbf{and}\ \mathbf{non}\ 0/0 > 0)\ \mathbf{or}\ 0/0 > 0\ \mathbf{or}\ 0 \geq 0)\ \mathbf{eq}\ (0 \leq 0\ \mathbf{or}\ 0/0 > 0\ \mathbf{or}\ 0 \geq 0)$
$\cong$ {(17); (20) and the result of the previous derivation}
T .

A frequent application of Leibniz's rule is the substitution of truth values for atoms. For this case, Theorem 1 implies the following strengthening of Theorem 3 of [3]:

Theorem 3 *For* $E \in Ex$ *and* $x \in At$ *with* $av.x \neq U$ *we have*

$$E \cong (x := av.x).E \ .$$

□

The proof uses induction over the grammar; the base case is treated much like the proof of (19), this time using (1) and (2) instead of (3). It is given in [4].

As example of the use of Theorem 3, the derivation

$$
\begin{array}{ll}
 & 0 < 0 \ \&\ 0/0 > 0 \\
\equiv & \quad \{\, \neg(0 < 0) \,\} \\
 & \mathit{false} \ \&\ 0/0 > 0
\end{array}
$$

may be interpreted as

$$
\begin{array}{ll}
 & 0 < 0 \ \mathbf{and}\ 0/0 > 0 \\
\cong & \quad \{\text{Theorem 3, using } \mathit{av}.(0 < 0) = F \,\} \\
 & F \ \mathbf{and}\ 0/0 > 0 \quad .
\end{array}
$$

References

[1] R.L. Baber, *The spine of software*. John Wiley & Sons, Chichester, 1987.

[2] H. Barringer, J.H. Cheng, & C.B. Jones, 'A logic covering undefinedness in program proofs'. *Acta Inf.* **21** (1984), 251–269.

[3] A. Bijlsma, 'Semantics of quasi-boolean expressions', in: W.H.J. Feijen et al. (eds.), *Beauty is our business: a birthday salute to Edsger W. Dijkstra*. Springer, New York, 1990; pp. 27–35.

[4] A. Bijlsma, *Quasi-boolean equivalence*. Memorandum AB30, Eindhoven University of Technology, 1992.

[5] J.H. Cheng & C.B. Jones, 'On the usability of logics which handle partial functions', in: C. Morgan and J.C.P. Woodcock (eds.), *Third refinement workshop*. Springer, London, 1991; pp. 51–72.

[6] E.W. Dijkstra, *A discipline of programming*. Prentice-Hall, Englewood Cliffs, 1976.

[7] E.W. Dijkstra & W.H.J. Feijen, 'The linear search revisited'. *Struct. Prog.* **10** (1989), 5–9.

[8] E.W. Dijkstra & C.S. Scholten, *Predicate calculus and program semantics*. Springer, New York, 1990.

[9] D. Gries, *The science of programming*. Springer, New York, 1981.

[10] E.C.R. Hehner, L.E. Gupta, & A.J. Malton, 'Predicative methodology'. *Acta Inf.* **23** (1986), 487–505.