Refutation of abductive repair in ontology engineering

Abstract: We evaluate the stated example of complete-debug problem (CDP) in formulas framing the definitions, oracles, and repairs. None is tautologous. This forms a *non* tautologous fragment of the universal logic VŁ4.

We assume the method and apparatus of Meth8/VŁ4 with `T`autology as the designated proof value, **F** as contradiction, `N` as truthity (non-contingency), and `C` as falsity (contingency). The 16-valued truth table is row-major and horizontal, or repeating fragments of 128-tables, sometimes with table counts, for more variables. (See ersatz-systems.com.)

LET ~ Not, ¬ ; + Or, ∨, ∪, ⊔ ; - Not Or; & And, ∧, ∩, ⊓, · , ⊗ ; \ Not And;
> Imply, greater than, →, ⇒, ↦, ≻, ⊃, ↠ ; < Not Imply, less than, ∈, ≺, ⊂, ⊬, ⊭, ↞, ≲ ;
= Equivalent, ≡, :=, ⇔, ↔, ≜, ≈, ≃ ; @ Not Equivalent, ≠, ⊕;
% possibility, for one or some, ∃, ◊, M; # necessity, for every or all, ∀, □, L;
(z=z) `T` as tautology, ⊤, ordinal 3; (z@z) **F** as contradiction, Ø, Null, ⊥ , zero;
(%z>#z) `N` as non-contingency, Δ, ordinal 1; (%z<#z) `C` as contingency, ∇, ordinal 2;
~(y < x) (x ≤ y), (x ⊆ y), (x ⊑ y); (A=B) (A~B).
Note for clarity, we usually distribute quantifiers onto each designated variable.

From: Lambrix, P. (2019). Completing and debugging ontologies: state of the art and challenges. arxiv.org/pdf/1908.03171.pdf

Abstract As semantically-enabled applications require high-quality ontologies, developing and maintaining as correct and complete as possible ontologies is an important, although difficult task in ontology engineering. A key step is ontology debugging and completion. In general, there are two steps: detecting defects and repairing defects. In this paper we formalize the repairing step as an abduction problem and situate the state of the art with respect to this framework. ...

2.1 Formalization

2.1.1 Repair … As an example, consider the CDP [complete-debug problem] in Fig. 1. … Then R1,R2, R3, R4 and R5 are all repairs of the CDP

Figure 1: Example complete-debug problem

T: {ax1: p1 ⊑ p2, ax2: p1 ⊑ p3, ax3: p1 ⊑ ¬p4, ax4: p2 ⊑ p4, ax5: p2 ⊑ p5, ax6: p3 ⊑ p5,
ax7: p3 ⊑ p6, ax8: p4 ⊑ p7, ax9: p5 ⊑ ∀s.p8, ax10: p6 ⊑ ∃s.¬p8} (2.1.1.1.1)

. . .

Or(X) = true for X = ax2, ax3, ax4, ax5, ax7, ax8, ax9, p7 ⊑ p3; (2.1.1.3.1)

Or(X) = false for X = ax1, ax6, ax10, p7 ⊑ p5, p3 ⊑ p8 (2.1.1.4.1)

Remark 2.1.1.1.0: We test (T>((true for X)&(false for X))) as Eqs. 2.1.1.1.1 > ((2.1.1.3.1)&(2.1.1.4.1)). (2.1.1.5.1)

LET p, q, r, s, t, u, v, w, x, y, z:
p1, p2, p3, p4, p5, p6, p7, p8, s, y, z.

((((~(q<p)&~(r<p))&(~(~s<p)&~(s<q)))&((~(t<q)&~(t<r))&(~(u<r)&~(v<s))))&
(~((#x&w)<t)&~((%x&~w)<u)))>
(((((~(r<p)&~(~s<p))&(~(s<q)&~(t<q)))&((~(u<r)&~(v<s))&(~((#x&w)<t)&
~(r<v))))=(z=z))&
((((~(q<p)&~(t<r))&~((%x&~w)<u))&(~(t<v)&~(w<r)))=(z@z))) ;

```
TCTC TCTC TTTC TTTC
TTTT TTTC TTTT TTTC
TTTT TFTF TTTT TTTF
TTTT TTTF TTTT TTTF
TTTT TTTT TTTC TTTC
TTTT TTTT TTTT TTTC
TTTT TTTT TTTT TTTF }x2
TTTT TFTF TTTT TTTF }x2
TTTT TTTF TTTT TTTF }
TTTT TTTT TTTT TTTF }x4

TTTT TTTT TTTT TTTT }x2
TTTT TFTF TTTT TTTF
TTTT TTTF TTTT TTTF
TTTT TTTT TTTT TTTT }x2
TTTT TTTT TTTT TTTF }x2
TTTT TNTN TTTT TTTN }x2
TTTT TTTF TTTT TTTF }
TTTT TTTT TTTT TTTN }x2
TTTT TTTT TTTT TTTF }   107 steps
```
(2.1.1.5.2)

Remark 2.1.2.1.0: We map the unique relations from repairs of R1, R2, R3, R4, R5 of

R1={p4⊑p5, p7⊑p3}, R2={p4⊑p5, p7⊑p3}, R3={p7⊑p3}, R4={p4⊑p5}, R5={p4⊑p5, p7⊑p3},
as p4⊑p5, p7⊑p3 (2.1.2.1.1)

~(t<s)&~(r<v) ; (2.1.2.1.2)

Remark 2.1.3.1.0: We test the CDP to imply the repairs:

Eqs. 2.1.1.5.1 implies 2.1.2.1.1. (2.1.3.1.1)

(((((~(q<p)&~(r<p))&(~(~s<p)&~(s<q)))&((~(t<q)&~(t<r))&(~(u<r)&~(v<s))))&
(~((#x&w)<t)&~((%x&~w)<u)))>(((((~(r<p)&~(~s<p))&(~(s<q)&~(t<q)))&
((~(u<r)&~(v<s))&(~((#x&w)<t)&~(r<v))))=(z=z))&((((~(q<p)&~(t<r))&
~((%x&~w)<u))&(~(t<v)&~(w<r)))=(z@z))))>(~(t<s)&~(r<v)) ;

```
TTTT FNFN TTTT FFFN
FFFF FFFN TTTT FFFN
TTTT FTFT TTTT FFFT
FFFF FFFT TTTT FFFT
TTTT TTTT TTTT TTTT }x2
FFFF FFFF TTTT TTTT }
TTTT FTFT TTTT FFFT }x2
FFFF FFFT TTTT FFFT }
TTTT TTTT TTTT TTTT }x2
FFFF FFFF TTTT TTTT }

TTTT FFFF TTTT FFFF
FFFF FFFF TTTT FFFF
TTTT FTFT TTTT FFFT
FFFF FFFT TTTT FFFT
```

```
TTTT TTTT TTTT TTTT }x2
FFFF FFFF TTTT TTTT }
TTTT FCFC TTTT FFFC }x2
FFFF FFFT TTTT FFFT }
TTTT TTTT TTTT TTTT }x2
FFFF FFFF TTTT TTTT }   115 steps        (2.1.3.1.2)
```

Eqs. 2.1.1.5.2 and 2.1.3.1.2 as rendered are *not* tautologous. This means the example given as 2.1.1.1 is *not* tautologous, the oracles in 2.1.1.3.1 and 2.1.1.4.1 are *not* truthful, and the repairs in 2.1.2.1.1 are incorrect. This refutes the conjecture of ontology engineering.