The Surprise Paradox

Joachim Breitner
Programming Paradigms Group
Karlsruhe Institute for Technology
breitner@kit.edu

October 11, 2017

Abstract

In 1964, Fitch showed that the paradox of the surprise hanging can be resolved by showing that the judge’s verdict is inconsistent. His formalization builds on Gödel’s coding of provability.

In this theory, we reproduce his proof in Isabelle, building on Paulson’s formalisation of Gödel’s incompleteness theorems.

Contents

1 Excluded or
2 Formulas with variables
3 Fitch’s proof
4 Substitution, quoting and V-quoting

theory Surprise-Paradox
imports
Incompleteness.Goedel-I
Incompleteness.Pseudo-Coding
begin

The Surprise Paradox comes in a few variations, one being the following:

A judge sentences a felon to death by hanging, to be executed at noon the next week, Monday to Friday. As an extra punishment, the judge does not disclose the day of the hanging and promises the felon that it will come at a surprise.

The felon, probably a logician, then concludes that he cannot be hanged on Friday, as by then it would not longer be a surprise. Using this fact and similar reasoning, he cannot be hanged on
Thursday, and so on. He reaches the conclusion that he cannot be hanged at all, and contently returns to his cell.

Wednesday, at noon, the hangman comes to the very surprised felon, and executes him.

Obviously, something is wrong here: Does the felon reason wrongly? It looks about right. Or is the judge lying? But his prediction became true!

It is an interesting exercise to try to phrase the Surprise Paradox in a rigorous manner, and see this might clarify things.

In 1964, Frederic Fitch suggested a formulation that refines the notion of “surprise” as “cannot be proven from the given assumptions” [1]. To formulate that, we need propositions that reference their own provability, so just as Fitch builds on Gödel’s work, we build on Paulson’s formalisation of Gödel’s incompleteness theorems in Isabelle [2].

1 Excluded or

Although the proof goes through with regular disjunction, Fitch phrases the judge’s proposition using exclusive or, so we add syntax for that.

abbreviation Xor :: fm ⇒ fm ⇒ fm (infix XOR 120)
  where Xor A B ≡ (A OR B) AND ((Neg A) OR (Neg B))

2 Formulas with variables

In Paulson’s formalisation of terms and formulas, only terms carry variables. This is sufficient for his purposes, because the proposition that is being diagonalised needs itself as a parameter to PfP::tm ⇒ fm, which does take a term (which happens to be a quoted formula).

In order to stay close to Fitch, we need the diagonalised proposition to occur deeper in a quotation of a few logical conjunctions. Therefore, we build a small theory of formulas with variables (“holed” formulas). These support substituting a formula for a variable, this substitution commutes with quotation, and closed holed formulas can be converted to regular formulas.

In our application, we do not need holes under an quantifier, which greatly simplifies things here. In particular, we can use datatype and fun.

datatype hfm =
  HVar name
  | HFm fm
  | HDисj hfm hfm (infixr HOR 130)
  | HНeg hfm

abbreviation HImp :: hfm ⇒ hfm ⇒ hfm (infixr HIMP 125)
where \( \text{HImp} \ A \ B \equiv \text{HDisj} \ (\text{HNeg} \ A) \ B \)

**definition** \( \text{HConj} :: \text{hfm} \Rightarrow \text{hfm} \Rightarrow \text{hfm} \) \(\text{infixr} \ \text{HAND} \ 135\)
where \( \text{HConj} \ A \ B \equiv \text{HNeg} \ (\text{HDisj} \ (\text{HNeg} \ A) \ (\text{HNeg} \ B)) \)

**abbreviation** \( \text{HXor} :: \text{hfm} \Rightarrow \text{hfm} \Rightarrow \text{hfm} \) \(\text{infix} \ \text{HXOR} \ 120\)
where \( \text{HXor} \ A \ B \equiv (A \ \text{HOR} \ B) \ \text{HAND} \ (A \ \text{HOR} \ B) \)

fun \( \text{subst-hfm} :: \text{hfm} \Rightarrow \text{name} \Rightarrow \text{fm} \Rightarrow \text{hfm} \) \(\text{-}’(\text{-}::=-’) [1000, \ 0, \ 0] 200\)
where \( (\text{HVar} \ \text{name})(i::=x) = (\text{if} \ i = \ \text{name} \ \text{then} \ \text{HFm} \ x \ \text{else} \ \text{HVar} \ \text{name}) \)
| \( (\text{HDisj} \ A \ B)(i::=x) = \text{HDisj} \ (A(i::=x)) \ (B(i::=x)) \)
| \( (\text{HNeg} \ A)(i::=x) = \text{HNeg} \ (A(i::=x)) \)
| \( (\text{HFm} \ A)(i::=x) = \text{HFm} \ A \)

lemma \( \text{subst-hfml-Conj}[\text{simp}]: \)
\( (\text{HConj} \ A \ B)(i::=x) = \text{HConj} \ (A(i::=x)) \ (B(i::=x)) \)

unfolding \( \text{HConj-def} \) by \( \text{simp} \)

**instantiation** \( \text{hfm} :: \text{quot} \)
begin
fun \( \text{quot-hfm} :: \text{hfm} \Rightarrow \text{tm} \)
where \( \text{quot-hfm} \ (\text{HVar} \ \text{name}) = (\text{Var} \ \text{name}) \)
| \( \text{quot-hfm} \ (\text{HFm} \ A) = \lfloor A \rfloor \)
| \( \text{quot-hfm} \ (\text{HDisj} \ A \ B) = \text{HPair} \ (\text{HTuple} \ 3) \ (\text{HPair} \ (\text{quot-hfm} \ A) \ (\text{quot-hfm} \ B)) \)
| \( \text{quot-hfm} \ (\text{HNeg} \ A) = \text{HPair} \ (\text{HTuple} \ 4) \ (\text{quot-hfm} \ A) \)

instance ..
end

lemma \( \text{subst-quot-hfm}[\text{simp}]: \text{subst} \ i \ [P] \ [A] = [A(i::=P)] \)
by \( \text{(induction} \ A) \ \text{auto} \)

fun \( \text{hfm-to-fm} :: \text{hfm} \Rightarrow \text{fm} \)
where \( \text{hfm-to-fm} \ (\text{HVar} \ \text{name}) = \text{undefined} \)
| \( \text{hfm-to-fm} \ (\text{HFm} \ A) = A \)
| \( \text{hfm-to-fm} \ (\text{HDisj} \ A \ B) = \text{Disj} \ (\text{hfm-to-fm} \ A) \ (\text{hfm-to-fm} \ B) \)
| \( \text{hfm-to-fm} \ (\text{HNeg} \ A) = \text{Neg} \ (\text{hfm-to-fm} \ A) \)

lemma \( \text{hfm-to-fm-Conj}[\text{simp}]: \)
\( \text{hfm-to-fm} \ (\text{HConj} \ A \ B) = \text{Conj} \ (\text{hfm-to-fm} \ A) \ (\text{hfm-to-fm} \ B) \)

unfolding \( \text{HConj-def} \ \text{Conj-def} \) by \( \text{simp} \)

fun \( \text{closed-hfm} :: \text{hfm} \Rightarrow \text{bool} \)
where \( \text{closed-hfm} \ (\text{HVar} \ \text{name}) \longleftrightarrow \text{False} \)
| \( \text{closed-hfm} \ (\text{HFm} \ A) \longleftrightarrow \text{True} \)

3
\[\text{closed-hfm}(\text{HDisj} \ A \ B) \leftrightarrow \text{closed-hfm} \ A \land \text{closed-hfm} \ B\]

\[\text{closed-hfm}(\text{HNeg} \ A) \leftrightarrow \text{closed-hfm} \ A\]

**Lemma closed-hfm-Conj[simp]:**

\[\text{closed-hfm}(\text{HConj} \ A \ B) \leftrightarrow \text{closed-hfm} \ A \land \text{closed-hfm} \ B\]

**Unfolding HConj-def by simp**

**Lemma quot-closed-hfm[simp]:**

\[\text{closed-hfm} \ (\text{HConj} \ A \ B) \leftrightarrow \text{closed-hfm} \ A \land \text{closed-hfm} \ B\]

**Unfolding HConj-def by simp**

**Lemma quot-closed-hfm[simp]:**

\[\text{closed-hfm} \ A = \Rightarrow \lceil A \rceil = \lceil \text{hfm-to-fm} \ A \rceil\]

**By (induction A) (auto simp add: quot-fm-def)**

**Declare quot-hfm.simps[simp del]**

### 3 Fitch’s proof

For simplicity, Fitch (and we) restrict the week to two days. Propositions \(Q_1\) and \(Q_2\) represent the propositions that the hanging occurs on the first resp. second day, but these can obviously be any propositions.

**Context**

- \(\text{fixes} \ Q_1 :: \text{fm} \) and \(\text{fixes} \ Q_2 :: \text{fm}\)
- \(\text{assumes} \ Q\text{-closed:} \ \text{supp} \ Q_1 = \{\} \ \text{supp} \ Q_2 = \{\}\)

**Begin**

In order to define the judge’s proposition, which is self-referential, we apply Gödel’s diagonalisation lemma.

**Definition \(H :: \text{fm}\) where**

\[H = Q_1 \land \text{Neg} (PfP \ [\text{HVar} \ X_0 \ \text{HIMP} \ \text{HFm} \ Q_1]) \lor \]

\[Q_2 \land \text{Neg} (PfP \ [\text{HVar} \ X_0 \ \text{HAND} \ \text{HNeg} (\text{HFm} \ Q_1) \ \text{HIMP} \ (\text{HFm} \ Q_2)])\]

**Definition \(P\) where**

\[P = (\text{SOME} \ P. \ \{\} \vdash \ P \IFF H(X_0 ::= \lceil P \rceil))\]

**Lemma \(P':: \{\} \vdash P \IFF H(X_0 ::= \lceil P \rceil)\)**

**Proof**

- **from diagonal[where \(\alpha = H\) and \(i = X_0\)]**
- **obtain \(\delta\) where \(\{\} \vdash \delta \IFF H(X_0 ::= \lceil \delta \rceil)\)**.
- **thus \text{thesis} unfolding \(P\text{-def}\) by (rule someI)**

**Qed**

From now on, the lemmas are named after their number in Fitch’s paper, and correspond to his statements pleasingly closely.

**Lemma 7: \(\{\} \vdash P \IFF\)**

\[(Q_1 \land \text{Neg} (PfP [P \text{IMP} Q_1]) \lor \]

\[Q_2 \land \text{Neg} (PfP [P \land \text{Neg} Q_1 \text{IMP} Q_2]))\]

**Using \(P'\) unfolding \(H\text{-def}\)**

**By (simp add: Q-closed forget-subst-fm[unfolded fresh-def])**

**Lemmas 7-E = 7[THEN thin0, THEN Iff-MP-left', OF Conj-E, OF thin2]**
lemmas propositional-calculus =
  AssumeI Neg-I Imp-I Conj-E Disj-E ExFalso[OF Neg-E]
  ExFalso[OF rotate2, OF Neg-E] ExFalso[OF rotate3, OF Neg-E]

lemma 8: {} ⊢ (P AND Neg Q) IMP Q
  by (intro propositional-calculus 7-E)

lemma 10: {} ⊢ PIMP [(P AND Neg Q) IMP Q]
  using 8 by (rule proved-imp-proved-PIMP)
lemmas 10-I = 10[THEN thin0]

lemma 11: {} ⊢ P IMP Q
  by (intro propositional-calculus 7-E 10-I)

lemma 12: {} ⊢ PIMP [P IMP Q]
  using 11 by (rule proved-imp-proved-PIMP)
lemmas 12-I = 12[THEN thin0]

lemma 13: {} ⊢ Neg P
  by (intro propositional-calculus 7-E 10-I 12-I)
end

4 Substitution, quoting and V-quoting

In the end, we did not need the lemma at the end of this section, but it may
be useful to others.

lemma trans-tm-forgets: atom i set is * t t = trans-tm is t = trans-tm [] t
  by (induct t rule: tm.induct)
    (auto simp: lookup-notin fresh-star-def fresh-at-base)

lemma vquot-dbtm-fresh: atom i V * t t = vquot-dbtm V t = quot-dbtm t
  by (nominal-induct t rule: dbtm.strong-induct)
    (auto simp add: fresh-star-def fresh-at-base)

lemma subst-vquot-dbtm-trans-tm[simp]:
  atom i * is t = atom i set is * t =
  subst i t (vquot-dbtm {i} (trans-tm is t)) =
  quot-dbtm (trans-tm is (subst i t t))
  by (nominal-induct t avoiding: is i rule: tm.strong-induct)
    (auto simp add: quot-tm-def lookup-notin fresh-imp-notin-env
      vquot-dbtm-fresh lookup-fresh
      intro: trans-tm-forgets[symmetric])

lemma subst-vquot-dbtm-trans-fm[simp]:
  atom i * is t = atom i set is * t =
  subst i t (vquot-dbfm {i} (trans-fm is A)) =
  quot-dbfm (trans-fm is (subst-fm A i t))
  by (nominal-induct A avoiding: is i i rule: fm.strong-induct)
(auto simp add: quot-fm-def fresh-Cons)

lemma subst-vquot[simp]:
  subst i [t] [A]i = [A(i ::= t)]
  by (nominal-induct A avoiding: i t rule: fm.strong-induct)
    (auto simp add: vquot-fm-def quot-fm-def fresh-Cons)

end

References
