

Is the Curry boiling?

A model checker for Inquisitive Semantics

Bo Flachs & Wessel Kroon

January 27, 2022

- Inquisitive semantics is a framework for analysing information exchange.
- We will make a model checker for the most basic version of this framework (InqB).

- Classical formal semantics are suitable for analysing declarative sentences.
- However, they are not equipped to analyse questions such as:
 - *“Is the Curry boiling?”*
- Inquisitive semantics was developed at the ILLC to overcome these issues.

- A first order inquisitive model $M = \langle W, D, I \rangle$ where
 - W is a set of possible worlds;
 - D is a non-empty set of individuals;
 - I is an interpretation function.

```
type World      = Int
```

```
type Universe   = [World]
```

```
type Individual = String
```

```
type Domain     = [Individual]
```

```
type UnRelation = [(World, [Individual])]
```

```
type BiRelation = [(World, [(Individual, Individual)])]
```

```
data Model = Mo { universe :: Universe
                  , dom    :: Domain
                  , unRel  :: [UnRelation]
                  , biRel  :: [BiRelation]
                  , tertRel :: [TertRelation] }
deriving (Eq, Ord, Show)
```

- The semantics of InqB are given in terms of sets of worlds rather than in worlds.
- We call these sets of worlds information states.

```
type InfState = [World]
```

- Propositions are non-empty, downward-closed sets of sets of worlds
- Intuition: A proposition consists of the information states that resolve the issues raised by that proposition.
- We represent a propositions by its maximal elements, called alternatives.
- A proposition has informative content and inquisitive content

Propositions in Haskell

```
type Prop      = [[World]]
```

```
alt :: Model -> Form -> [InfState]
```

```
alt m f = sort [x | x <- p, not (any (strictSubset x) p)]  
    where p = toProp m f
```

```
info :: Model -> Form -> InfState
```

```
info m f = sort . nub . concat $ toProp m f
```


Support of Propositions

- We say that an information state supports a propositions if its an element of it.
- In other words, s supports P iff $s \in P$.

```
supportsProp :: InfState -> Prop -> Bool  
supportsProp s p = s `elem` p
```

- The language of InqB is that of first order logic:

```
type Var          = String
```

```
data Term         = Indv Individual | Var Var
  deriving (Eq, Ord, Show)
```

```
data Form = UnR UnRelation Term
  | BinR BiRelation Term Term
  | TertR TertRelation Term Term Term
  | Neg Form | Con Form Form | Dis Form Form
  | Impl Form Form
  | Forall Var Form | Exists Var Form
  deriving (Eq, Ord, Show)
```

Special Operators

- Furthermore, there are two special operators: ! and ?.
- The operators ! and ? make a formula non-inquisitive and non-informative, respectively.
- As they are abbreviations, we implemented them as functions:

```
nonInq :: Form -> Form
```

```
nonInq = Neg . Neg
```

```
nonInf :: Form -> Form
```

```
nonInf f = Dis f $ Neg f
```

- Formulas can be interpreted in a model as propositions.

```
toProp :: Model -> Form -> Prop
toProp _ (UnR r i) = closeDownward
    [[x | (x, y) <- r, getString i `elem` y]]
toProp m (Con f1 f2)
    = toProp m f1 `intersect` toProp m f2
```

- In the other clauses of this function we use more complicated helper functions. We will not go into those clauses.

- An information state supports a formula if it is an element of the corresponding proposition.

```
supportsForm :: Model -> InfState -> Form -> Bool  
supportsForm m s f = supportsProp s $ toProp m f
```

- Given a model, an information state and a formula, our model checker should check if that information state supports the formula.

Useful functions

```
isInquisitive :: Model -> Form -> Bool
```

```
isInquisitive m f  
    = sort (toProp m f) /=  
      (sort . powerset) (info m f)
```

```
isInformative :: Model -> Form -> Bool
```

```
isInformative m f  
    = (sort . universe) m /= sort (info m f)
```

```
entails :: Model -> Form -> Form -> Bool
```

```
entails m f1 f2 = all (`elem` p2) p1 where  
    p1 = toProp m f1  
    p2 = toProp m f2
```

To do

- Implement QuickCheck;
- Check several well-known theorems of InqB using QuickCheck.