My Report

Simon Chiu

Thursday 17th July, 2025

Contents

1 Introduction

This project implements an explicit model checker for the logic of knowledge and safe belief for a single agent, $K\Box$, as presented in [BS08], together with three dynamic belief revision operators: update or truthful public announcement, radical upgrade, and conservative upgrade, which we will call $K\Box+$. We will use the language Haskell, which provides a clean way to model and manipulate abstract structures.

2 Syntax

The language of $K\Box+$ is defined over a set of propositional variables Prop:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \Box\varphi \mid K\varphi \mid [!\varphi]\varphi \mid [\Uparrow \varphi]\varphi \mid [\uparrow \varphi]\varphi \ ,$$

where p is taken from a set of propositional variables.

Below is the implementation of the syntax of $K\Box+$, where we index the propositional variables with integers:

```
module Syntax where

data KSBForm = P Int | Neg KSBForm | Con KSBForm KSBForm | Box KSBForm | K KSBForm
             | Ann KSBForm KSBForm | Rad KSBForm KSBForm | Cons KSBForm KSBForm
```

The other boolean operators $\vee, \rightarrow, \top, \bot$ can be defined in the usual way:

```
dis, implies :: KSBForm -> KSBForm -> KSBForm
dis f g = Neg $ Con (Neg f) (Neg g)

implies f = dis (Neg f)

top, bot :: KSBForm
top = dis (P 0) (Neg $ P 0)

bot = Neg top
```

The *conditional belief*, *belief*, and *strong belief* operators can be defined as follows:

$$\begin{aligned} B^{\varphi}\psi &:= \tilde{K}\varphi \rightarrow \tilde{K}(\varphi \wedge \Box(\varphi \rightarrow \psi)) \ ; \\ B\varphi &:= B^{\top}\varphi \ ; \\ Sb\varphi &:= B\varphi \wedge K(\varphi \rightarrow \Box\varphi) \ , \end{aligned}$$

where $\tilde{K}\varphi := \neg K\neg\varphi$ is the dual of the knowledge operator.

```
cBel :: KSBForm -> KSBForm -> KSBForm
cBel f g = (Neg . K . Neg) f `implies` (Neg . K . Neg) (f `Con` Box (f `implies` g))

bel :: KSBForm -> KSBForm
bel = cBel top

sBel :: KSBForm -> KSBForm
sBel f = Con (bel f) (K (f `implies` Box f))
```

3 Semantics

The semantics of $K\Box+$ are given with respect to *plausibility models*. A (single-agent) *plausibility model* is a tuple $\mathcal{M} = (M, \leq, V)$, where

- M is a set of states or possible worlds;
- $\leq$ is a well-preorder, that is a reflexive, transitive, and well-founded relation. Well-foundedness means that for any non-empty subset $P \subseteq M$, the set of minimal-elements

$$\mathrm{Min}_{\leq} P := \{s \in P \mid s \leq s' \text{ for all } s' \in P\}$$

  is non-empty. Note that this also implies that $\leq$ is connected;
- $V : \mathsf{Prop} \to \mathcal{P}(M)$ is a valuation.

We call $\leq$ the plausibility relation. Following the convention in [BS08], we interpret $s \leq t$ as the state s as being more or equally as plausible as t.

For each plausibility relation, there is a corresponding epistemic indistinguishability relation $\sim$, defined as the union between $\leq$ and it's converse $\geq$. That is, $\sim := \leq \cup \geq$. In the single-agent case, this reduces to the universal relation $s \sim t$ iff $s, t \in M$.

The type for plausibility models is implemented as follows, where we always name the states with integers.

```
type World = Int
type Universe = [World]
type Proposition = Int

type Valuation = IntMap [Proposition]
type Relation = [(World,World)]

data PlausibilityModel = PlM {worlds :: Universe, rel :: Relation, val :: Valuation}
  deriving (Eq,Ord,Show)
```

To make the representation of the plausibility models more compact, the following functions construct the reflexive-transitive closure of a relation:

```
rClosure :: PlausibilityModel -> PlausibilityModel
rClosure m = PlM (worlds m) closure (val m) where
  closure = rel m ++ [(a,a) | a <- worlds m, (a,a) `notElem` rel m]

tClosure :: Relation -> Relation
tClosure r
  | r == closureStep = r
  | otherwise                = tClosure closureStep
  where
    closureStep = nub $ r ++ [(a,c) | (a,b) <- r, (b',c) <- r, b == b']

rtClosure :: PlausibilityModel -> PlausibilityModel
rtClosure m = rClosure $ PlM (worlds m) (tClosure (rel m)) (val m)
```

The semantic clauses of $K\Box$ are defined recursively as follows, where $\mathcal{M}$ is a plausibility model and $s \in M$ is a state:

$$\begin{array}{lll}
\mathcal{M}, s \vDash p & \text{iff} & s \in V(p) \ ; \\
\mathcal{M}, s \vDash \neg\varphi & \text{iff} & \mathcal{M}, s \nvDash \varphi \ ; \\
\mathcal{M}, s \vDash \varphi \wedge \psi & \text{iff} & \mathcal{M}, s \vDash \varphi \text{ and } \mathcal{M}, s \vDash \psi \ ; \\
\mathcal{M}, s \vDash \Box\varphi & \text{iff} & \mathcal{M}, t \vDash \varphi \text{ for all } t \leq s \ ; \\
\mathcal{M}, s \vDash K\varphi & \text{iff} & \mathcal{M}, t \vDash \varphi \text{ for all } s \sim t \ .
\end{array}$$

Note that $\Box$ is the Kripke modality of the converse plausibility relation $\geq$, due to our convention. We write $\|\varphi\|_{\mathcal{M}}$ to denote the set $\{s \in M \mid \mathcal{M}, s \vDash \varphi\}$.

To define the semantics of the dynamic modalities, we need the following three model transformers corresponding to the three modalities: For a plausibility model $\mathcal{M} = (M, \leq, V)$ and formula φ, the models $\mathcal{M}^{!\varphi}$, $\mathcal{M}^{\Uparrow\varphi}$, and $\mathcal{M}^{\uparrow\varphi}$ are defined as follows:

$\mathcal{M}^{!\varphi} = (M^{!\varphi}, \leq^{!\varphi}, V^{!\varphi})$, where

- $M^{!\varphi} := \{s \in M \mid \mathcal{M}, s \vDash \varphi\}$;
- $\leq^{!\varphi}$ is the restriction of $\leq$ to $M^{!\varphi}$;
- $V^{!\varphi}(p) := V(p) \cap M^{!\varphi}$.

$\mathcal{M}^{\Uparrow\varphi} = (M^{\Uparrow\varphi}, \leq^{\Uparrow\varphi}, V^{\Uparrow\varphi})$, where

- $M^{\Uparrow\varphi} := M$;
- $s \leq^{\Uparrow\varphi} t$ iff either of the following hold:
  - $s \leq t$, where both $s, t \in \|\varphi\|_{\mathcal{M}}$, or both $s, t \notin \|\varphi\|_{\mathcal{M}}$;
  - $\mathcal{M}, s \vDash \varphi$ and $\mathcal{M}, t \nvDash \varphi$;
- $V^{\Uparrow\varphi}(p) := V(p)$.

$\mathcal{M}^{\uparrow\varphi} = (M^{\uparrow\varphi}, \leq^{\uparrow\varphi}, V^{\uparrow\varphi})$, where

- $M^{\uparrow\varphi} := M$;
- $s \leq^{\uparrow\varphi} t$ iff either of the following hold:
  - $s \leq t$, where both $s, t \notin \mathrm{Min}_{\leq}\|\varphi\|_{\mathcal{M}}$;
  - $s \in \mathrm{Min}_{\leq}\|\varphi\|_{\mathcal{M}}$;
- $V^{\uparrow\varphi}(p) := V(p)$.

The semantic clauses of the dynamic modalities are as follows:

$$\begin{array}{lll} \mathcal{M}, s \vDash [!\varphi]\psi & \text{iff} & \mathcal{M}, s \nvDash \varphi \text{ or } \mathcal{M}^{!\varphi}, s \vDash \psi \ ; \\ \mathcal{M}, s \vDash [\Uparrow \varphi]\psi & \text{iff} & \mathcal{M}^{\Uparrow\varphi}, s \vDash \psi \ ; \\ \mathcal{M}, s \vDash [\uparrow \varphi]\psi & \text{iff} & \mathcal{M}^{\uparrow\varphi}, s \vDash \psi \ ; \end{array}$$

The semantics are implemented as follows:

```
pred :: Relation -> World -> [World]
pred r s = [t | (t,s') <- r, s == s']

(|=) :: (PlausibilityModel, World) -> KSBForm -> Bool
(m,s) |= P n      = n `elem` (val m ! s)
(m,s) |= Neg f    = not $ (m,s) |= f
(m,s) |= Con f g  = (m,s) |= f && (m,s) |= g
(m,s) |= Box f    = all (\t -> (m,t) |= f) (pred (rel m) s)
(m,_) |= K f      = all (\t -> (m,t) |= f) (worlds m)
(m,s) |= Ann f g  = (m,s) |= Neg f || (update m f, s) |= g
(m,s) |= Rad f g  = (radical m f, s) |= g
(m,s) |= Cons f g = (conservative m f, s) |= g
```

```
trueEverywhere :: PlausibilityModel -> KSBForm -> Bool
trueEverywhere m f = all (\s -> (m,s) |= f ) (worlds m)

update, radical, conservative :: PlausibilityModel -> KSBForm -> PlausibilityModel
update m f = PlM newWorlds newRel newVal where
  newWorlds = [s | s <- worlds m, (m,s) |= f]
  newRel    = [(s,t) | (s,t) <- rel m, s `elem` newWorlds && t `elem` newWorlds]
  newVal    = restrictKeys (val m) (Data.Intset.fromList newWorlds )

radical m f = PlM newWorlds newRel newVal where
  newWorlds  = worlds m
  newRel     = [(s,t) | (s,t) <- rel m, s `elem` trueWorlds && t `elem` trueWorlds]
             ++ [(s,t) | (s,t) <- rel m, s `notElem` trueWorlds && t `notElem` trueWorlds]
             ++ [(s,t) | s <- trueWorlds, t <- newWorlds, t `notElem` trueWorlds]
  newVal     = val m
  trueWorlds = [s | s <- worlds m, (m,s) |= f]

conservative m f = PlM newWorlds newRel newVal where
  newWorlds  = worlds m
  newRel     = [(s,t) | (s,t) <- rel m, s `notElem` bestWorlds && t `notElem` bestWorlds]
      ++ [(s,t) | s <- bestWorlds, t <- newWorlds]
  newVal     = val m
  trueWorlds = [s | s <- worlds m, (m,s) |= f]
  bestWorlds = [s | s <- trueWorlds, all (\t -> (s,t) `elem` rel m) trueWorlds]
```

4 Examples

In this section, we encode two example models and some formulas, used to test our implementation in section 5

The first is the Albert Winestein example from the lecture slides:

```
d,g :: Int
d = 1
g = 2

w1, w2, w3 :: World
w1 = 1
w2 = 2
w3 = 3

albertModel :: PlausibilityModel
albertModel = rtClosure $ PlM aworlds arel aval where
  aworlds = [w1,w2,w3]
  arel    = [(w1,w2), (w2,w3)]
  aval    = fromList [(w1,[g]), (w2,[d]), (w3, [d,g]) ]

albertModelState :: (PlausibilityModel, World)
albertModelState = (albertModel, w3)

albert1, albert2, albert3, albert4, albert5, albert6, albert7, albert8, albert9 :: KSBForm
albert1 = bel (P g)
albert2 = K (dis (P g) (P d))
albert3 = bel (Neg $ P d)
albert4 = cBel (P d) (Neg $ P g)
albert5 = Con (P d) (P g)
albert6 = Ann (P d) (K $ P d)
albert7 = Rad (P d) (sBel $ P d)
albert8 = Cons (P d) (bel $ P d)
albert9 = Cons (P d) (sBel $ P d)
```

The second example is the robot example from Homework 3:

```
m,e :: Int
m = 1
e = 2
```

```
s1, s2, s3, s4 :: World
s1 = 1
s2 = 2
s3 = 3
s4 = 4

robotModel :: PlausibilityModel
robotModel = rtClosure $ PlM rworlds rrel rval where
  rworlds = [s1,s2,s3,s4]
  rrel    = [(s1,s2), (s1,s3), (s2,s3), (s3,s2), (s2,s4), (s3,s4)]
  rval    = fromList [(s1, []), (s2, [e]), (s3,[m]),(s4,[m,e])]

robot1, robot2, robot3, robot4, robot5, robot6, robot7, beFalse :: KSBForm
robot1 = bel (Neg $ Con (P e) (Neg $ P d))
robot2 = sBel (Neg $ Con (P e) (Neg $ P d))
robot3 = Rad (P e) $ sBel (P e)
robot4 = Rad (P e) $ Rad (P m) $ sBel (P m)
robot5 = Rad (P e) $ Rad (P m) $ Con (bel $ P e) (Neg $ sBel $ P e)
-- whatever you currently believe about the enemy is false
beFalse = Con (bel (P e) `implies` Neg (P e)) (bel (Neg $ P e) `implies` P e)
robot6 = Rad (P e) $ Rad (P m) $ Ann beFalse $ Con (K $ Neg $ P e) (bel $ P m)
robot7 = Ann beFalse $ Rad (P e) $ Rad (P m) $ Con (K $ P e) (bel $ P m)
```

5 Simple Tests

We now perform static tests on two example models in 4 using the Hspec library.

First, we have the Albert Winestein model:

```
main :: IO ()
main = hspec $ do
  describe "Albert Winestein Example" $ do
    it "Initially, Albert believes he is a genius;" $
      albertModelState |= albert1 `shouldBe` True
    it "he knows he's either a genius or drunk;" $
      albertModelState |= albert2 `shouldBe` True
    it "he believes he's not drunk;" $
      albertModelState |= albert3 `shouldBe` True
    it "but conditional on him being drunk, he would believe he's not a genius;" $
      albertModelState |= albert4 `shouldBe` True
    it "in the actual world, he is both drunk and a genius." $
      albertModelState |= albert5 `shouldBe` True
    it "Suppose he took a blood test, proving he is drunk beyond doubt. Then he knows he's
        drunk." $
      albertModelState |= albert6 `shouldBe` True
    it "Suppose his trusted friend Mary Curry tells him he's drunk. Then he strongly
        believes he's drunk." $
      albertModelState |= albert7 `shouldBe` True
    it "Suppose Mary Curry tells him he's drunk, but he doesn't trust her too much. Then he
         only believes he's drunk," $
      albertModelState |= albert8 `shouldBe` True
    it "but this belief is not strong." $
      albertModelState |= albert9 `shouldBe` False
```

Then, we test the robot model:

```
  describe "Robot Example" $ do
    it "Initially, the robot believes there is neither a mine nor an enemy;" $
      trueEverywhere robotModel robot1 `shouldBe` True
    it "but this belief is not strong." $
      trueEverywhere robotModel robot2 `shouldBe` False
    it "After its sensors indicate vibrations, it strongly believes there is an enemy." $
      trueEverywhere robotModel robot3 `shouldBe` True
    it "Then, its metal detector indicates a mine, so it strongly believes there is a mine.
        " $
      trueEverywhere robotModel robot4 `shouldBe` True
    it "It still believes there is an enemy, but this belief is no longer strong." $
```

```
    trueEverywhere robotModel robot5 `shouldBe` True
  it "Afterwards, the controller announces that whatever it currently believes about the
      enemy is false. Then it knows that there is no enemy, and believes that there is a
      mine." $
    trueEverywhere robotModel robot6 `shouldBe` True
  it "If instead the robot first receives the controller's message, then senses vibration
      , and finally detects a mine, then it knows that there is an enemy, and believes
      there is a mine." $
    trueEverywhere robotModel robot7 `shouldBe` True
```

To run the tests, use `stack test`.

6 Future Work

Currently, this model checker can only be operated using a code-editor and through the command line. In the future, it would be nice to add a UI to make it more accessible to those who are not well-versed in Haskell.

The full code can be found on `https://github.com/shosukeyuu/KSBMC`.

References

[BS08] Alexandru Baltag and Sonja Smets. A Qualitative Theory of Dynamic Interactive Belief Revision. In Giacomo Bonanno, Wiebe Van Der Hoek, and Michael Wooldridge, editors, *Logic and the Foundations of Game and Decision Theory (LOFT 7)*, pages 11–58. Amsterdam University Press, 2008.