A Model Checker for Bilateral State-based Modal Logic (BSML)

Group 2

Tuesday 18th March, 2025

Abstract

Bilateral State-based Modal Logic (BSML), which proposed by Aloni et al. [2024], extends classical modal logic by adopting state-based semantics and introducing a non-emptiness atom to account for free choice inferences in natural language. Despite its expressive power, no automated verification tool exists for BSML. This project aims to develop a model checker for BSML, enabling automated reasoning over its logical properties.

Contents

1 Introduction

2 Motivition

BSML was developed to account for *Free Choice* (FC) inferences, where disjunctive sentences give rise to conjunctive interpretations. For example, the sentence "You may go to the beach or to the cinema" typically implies that "You may go to the beach *and* you may go to the cinema" This inference is unexpected from a classical logical perspective, as disjunction does not typically imply conjunction.

The key idea in BSML is the *neglect-zero tendency*, which posits that humans tend to disregard models that verify sentences by virtue of some empty configuration. BSML formalizes this tendency by introducing the *nonemptiness atom* (NE), which ensures that only nonempty states are considered in the interpretation of sentences. This leads to the prediction of both narrow-scope and wide-scope FC inferences, as well as their cancellation under negation.

BSML has been extended in two ways:

- **BSML**$^{\mathbb{V}}$: This extension adds the *global disjunction* $\mathbb{V}$, which allows for the expression of properties that are invariant under bounded bisimulation.
- **BSML**$^{\oslash}$: This extension adds the *emptiness operator* $\oslash$, which can be used to cancel out the effects of the nonemptiness atom (NE).

These extensions are expressively complete for certain classes of state properties, and natural deduction axiomatizations have been developed for each of these logics.

3 The Syntax of BSML

The syntax of **BSML**$^{\mathbb{V}}$ is defined over a set of propositional variables Prop. The formulas of BSML are generated by the following grammar:

$$\varphi ::= p \mid \bot \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid (\varphi \vee \varphi) \mid \Diamond\varphi \mid \mathrm{NE}$$

where $p \in \mathrm{Prop}$.

BSML$^{\mathbb{V}}$ extends **BSML** with global disjunction $\mathbb{V}$ by adding the clause $\varphi \mathbb{V} \varphi$.

The classical modal logic **ML** is the NE-free fragment of BSML.

Below is the implmentation of the syntax of **BSML**$^{\mathbb{V}}$. For the sake of breviety, we call the type `BSMLForm` for the formulas of **BSML**$^{\mathbb{V}}$. We use `Int` to index the set of propositional variables.

```
module Syntax where


type Prop = Int

data BSMLForm = P Prop | Bot | Neg BSMLForm | Con BSMLForm BSMLForm | Dis BSMLForm BSMLForm
     | Dia BSMLForm | NE | Gdis BSMLForm BSMLForm
  deriving (Eq,Ord,Show)
```

Note that `Dis` is the "$\vee$" disjunction, while `Gdis` is the "$\mathbb{V}$" disjunction.

The box modality $\Box$ is defined as the dual of the $\Diamond$: $\Box\varphi := \neg\Diamond\neg\varphi$.

```
box :: BSMLForm -> BSMLForm
box = Neg . Dia . Neg
```

The pragmatic enrichment function $[\quad]^+$: **ML** $\to$ **BSML**is recursively defined as:

$$
\begin{aligned}
[p]^+ &:= p \wedge \mathrm{NE} \\
[\bigcirc a]^+ &:= \bigcirc([a]^+) \wedge \mathrm{NE} \qquad \text{for } \bigcirc \in \neg, \Diamond, \Box \\
[\alpha \triangle \beta]^+ &:= ([\alpha]^+ \triangle [\beta]^+) \wedge \mathrm{NE} \qquad \text{for } \triangle \in \wedge, \vee
\end{aligned}
$$

We implment pragmatic enrichment `prag` as a partial function `BSMLForm -> BSMLForm`, leaving it up to the user to only use NE-free inputs.

```
prag :: BSMLForm -> BSMLForm
prag (P n)      = Con (P n) NE
prag (Neg f)    = Con (Neg $ prag f) NE
prag (Con f g)  = Con (Con (prag f) (prag g)) NE
prag (Dis f g)  = Con (Dis (prag f) (prag g)) NE
prag (Dia f)    = Con (prag (Dia f)) NE
prag (Gdis f g) = Con (Gdis (prag f) (prag g)) NE
prag Bot        = Con Bot NE
prag NE         = undefined
```

4 Semantics

The semantics of BSML is based on *team semantics*, where formulas are interpreted with respect to sets of possible worlds (called *states*) rather than single worlds. A *model* M is a triple (W, R, V), where:

- W is a nonempty set of possible worlds,
- $R \subseteq W \times W$ is an accessibility relation,
- V : Prop $\to \mathcal{P}(W)$ is a valuation function.

A *state* s is a subset of W. The support and anti-support conditions for BSML formulas are defined recursively as follows:

$$
\begin{aligned}
&M, s \models p \quad \text{iff} \quad \forall w \in s, w \in V(p)\\
&M, s \dashv p \quad \text{iff} \quad \forall w \in s, w \notin V(p)\\
&M, s \models \bot \quad \text{iff} \quad s = \emptyset\\
&M, s \dashv \bot \quad \text{always}\\
&M, s \models \mathrm{NE} \quad \text{iff} \quad s \neq \emptyset\\
&M, s \dashv \mathrm{NE} \quad \text{iff} \quad s = \emptyset\\
&M, s \models \neg\varphi \quad \text{iff} \quad M, s \dashv \varphi\\
&M, s \dashv \neg\varphi \quad \text{iff} \quad M, s \models \varphi\\
&M, s \models \varphi \wedge \psi \quad \text{iff} \quad M, s \models \varphi \text{ and } M, s \models \psi\\
&M, s \dashv \varphi \wedge \psi \quad \text{iff} \quad \exists t, u \subseteq s \text{ s.t. } s = t \cup u \text{ and } M, t \dashv \varphi \text{ and } M, u \dashv \psi\\
&M, s \models \varphi \vee \psi \quad \text{iff} \quad \exists t, u \subseteq s \text{ s.t. } s = t \cup u \text{ and } M, t \models \varphi \text{ and } M, u \models \psi\\
&M, s \dashv \varphi \vee \psi \quad \text{iff} \quad M, s \dashv \varphi \text{ and } M, s \dashv \psi\\
&M, s \models \varphi \mathbin{\vee\!\!\!\vee} \psi \quad \text{iff} \quad M, s \models \varphi \text{ or } M, s \models \psi\\
&M, s \dashv \varphi \mathbin{\vee\!\!\!\vee} \psi \quad \text{iff} \quad M, s \dashv \varphi \text{ and } M, s \dashv \psi\\
&M, s \models \Diamond\varphi \quad \text{iff} \quad \forall w \in s, \exists t \subseteq R[w] \text{ s.t. } t \neq \emptyset \text{ and } M, t \models \varphi\\
&M, s \dashv \Diamond\varphi \quad \text{iff} \quad \forall w \in s, M, R[w] \dashv \varphi
\end{aligned}
$$

The box modality $\Box$ is defined as the dual of the $\Diamond$, meaning $\Box\varphi$ is equivalent to $\neg\Diamond\neg\varphi$. This leads to the following support and antisupport clauses:

$$
\begin{aligned}
&M, s \models \Box\varphi \quad \text{iff} \quad \forall w \in s, M, R[w] \models \varphi\\
&M, s \dashv \Box\varphi \quad \text{iff} \quad \forall w \in s, \exists t \subseteq R[w] \text{ s.t. } t \neq \emptyset \text{ and } M, t \dashv \varphi
\end{aligned}
$$

The following is the definition of our Data Type for Model Checker.

```
-- {-# LANGUAGE InstanceSigs #-}
module Checker where



import Control.Monad
import System.Random
import Test.QuickCheck
import Data.List

import Syntax


type World = Integer
type Universe = [World]
type Proposition = Int
type State = [World]

type Valuation = World -> [Proposition]
type Relation = [(World,World)]

data KripkeModel = KrM Universe Valuation Relation

type ModelState = (KripkeModel,State)

instance Show KripkeModel where
  show (KrM u v r) = "KrM " ++ show u ++ " " ++ vstr ++ " " ++ show r where
    vstr = "(fromJust . flip lookup " ++ show [(w, v w) | w <- u] ++ ")"
```

The following helper function defines the set of all successors of a world:

```
(!) :: Relation -> World -> [World]
(!) r w = map snd $ filter ((==) w . fst) r
```

Here we define the semantics of **BSML** ...

```
-- helper function to find all pairs of worlds t and u that the union of t and u is the
    input s
allPairs :: [World] -> [([World], [World])]
allPairs []     = [([],[])]
allPairs (x:xs) =
  [ (x:ts, x:us) | (ts,us) <- allPairs xs ] ++
  [ (x:ts, us)   | (ts,us) <- allPairs xs ] ++
  [ (ts, x:us)   | (ts,us) <- allPairs xs ]

-- helper function to find all non-empty subsets of a list
subsetsNonEmpty :: [World] -> [[World]]
subsetsNonEmpty [] = []
subsetsNonEmpty (x:xs) =
  let rest = subsetsNonEmpty xs
  in [[x]] ++ rest ++ map (x:) rest

(|=) :: ModelState -> BSMLForm -> Bool
(KrM _ v _, s) |= (P p) = all (\w -> p `elem` v w) s
(_, s) |= Bot = null s
(_, s) |= NE = not $ null s
(KrM u v r, s) |= (Neg f) = (KrM u v r, s) =| f
m |= (Con f g) = m |= f && m |= g
(k, s) |= (Dis f g) = any (\(ts,us) -> (k, ts) |= f && (k, us) |= g) (allPairs s)
m |= (Gdis f g) = m |= f || m |= g
(KrM u v r, s) |= (Dia f) = all (\w -> any (\l -> (KrM u v r,l) |= f ) (subsetsNonEmpty (r
    ! w)))  s


(=|) :: ModelState -> BSMLForm -> Bool
(KrM _ v _, s) =| (P p) = all (\w -> p `notElem` v w) s
(_, _) =| Bot = True
(_, s) =| NE = null s
(KrM u v r, s) =| (Neg f) = (KrM u v r, s) |= f
(k, s) =| (Con f g) = any (\(ts,us) -> (k, ts) =| f && (k, us) =| g) (allPairs s)
m =| (Dis f g) = m =| f && m =| g
m =| (Gdis f g) = m =| f && m =| g
(KrM u v r, s)  =| (Dia f) = all (\w -> (KrM u v r, r ! w) =| f)  s
```

A model state pair (M, s) is indisputable if for all $w, v \in s, R[w] = R[v]$.

```
indisputable :: ModelState -> Bool
indisputable (KrM _ _ r ,s) = any (\w -> any (\v -> sort (r ! w) == sort (r ! v )) s ) s
```

A model state pair is state-based if for all $w \in s, R[w] = s$.

```
stateBased :: ModelState -> Bool
stateBased (KrM _ _ r , s) = any (\w -> sort (r ! w) == sort s) s
```

```
example1 :: KripkeModel
example1 = KrM [0,1,2] myVal [(0,1), (1,2), (2,1)] where
  myVal 0 = [0]
  myVal _ = [4]

example2 :: KripkeModel
example2 = KrM [0,1] myVal [(0,1), (1,1)] where
  myVal 0 = [0]
  myVal _ = [0, 4]

example11 :: ModelState
example11 = (example1, [0,1,2])

example12 :: ModelState
example12 = (example2, [0,1])
```

5 Conclusion

References

Maria Aloni, Aleksi Anttila, and Fan Yang. State-based modal logics for free choice. *Notre Dame Journal of Formal Logic*, 65(4):367–413, 2024. doi: 10.1215/00294527-2024-0027.