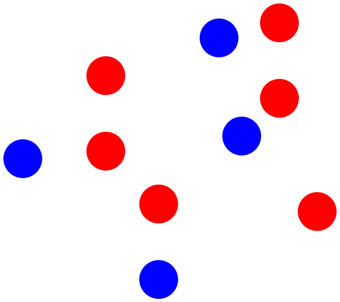
Functional Programming for Logicians - Lecture 5

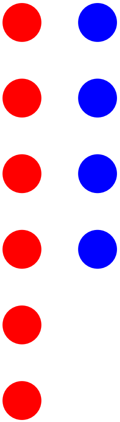
(Symbolic) Model Checking (Dynamic) Epistemic Logic(s)

Malvin Gattinger

19 January 2022



Are there more red or more blue points?



Are there more red or more blue points?

$6\times$ ● $4\times$ ●

Are there more red or more blue points?

Representation matters!

Overview

```
module L5 where
```

Epistemic Logic and Public Announcement Logic

Model Checking

Symbolic Model Checking S5 PAL

Binary Decision Diagrams

Examples and Benchmarks

Beyond S5 PAL

Summary

Two (unsafe!) helper functions we need later.

```
(!) :: Eq a => [(a,b)] -> a -> b
(!) v x = let (Just y) = lookup x v in y

(?) :: Eq a => [[a]] -> a -> [a]
(?) lls x = head (filter (x `elem`) lls)
```

Epistemic Logic and Public Announcement Logic

Example: Muddy Children

Imagine three children playing together outside.
Some of them get mud on their foreheads.
Each can see the face of others but not on their own forehead.

Example: Muddy Children

Imagine three children playing together outside.
Some of them get mud on their foreheads.
Each can see the face of others but not on their own forehead.

Along comes the father: "At least one of you has mud on your forehead".
The father then asks the following question, over and over: "Does any of you know whether they are muddy?"

Example: Muddy Children

Imagine three children playing together outside.
Some of them get mud on their foreheads.
Each can see the face of others but not on their own forehead.

Along comes the father: "At least one of you has mud on your forehead".
The father then asks the following question, over and over: "Does any of you know whether they are muddy?"

How often will the father repeat the question until any child reacts?

(adapted from Fagin et. al 1995)

Epistemic Logic

Syntax

$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid K_i\varphi$

Kripke Models

$\mathcal{M} = (W, R_i, \mathrm{Val})$ where

- W set of worlds
- $R_i \subseteq W \times W$ indistinguishability
- $\mathrm{Val}\colon W \to \mathcal{P}(P)$ valuation

Semantics

$\mathcal{M}, w \vDash K_i\varphi$ iff wR_iv implies $\mathcal{M}, v \vDash \varphi$

You know φ iff all the worlds you consider make φ true.

Epistemic Logic

Syntax

$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid K_i\varphi$

Kripke Models

$\mathcal{M} = (W, R_i, \mathrm{Val})$ where

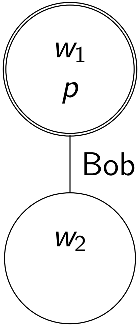
- W — set of worlds
- $R_i \subseteq W \times W$ — indistinguishability
- $\mathrm{Val}\colon W \to \mathcal{P}(P)$ — valuation

Semantics

$\mathcal{M}, w \vDash K_i\varphi \quad$ iff $\quad wR_iv$ implies $\mathcal{M}, v \vDash \varphi$

You know φ iff all the worlds you consider make φ true.

Example



$\neg K_{\mathrm{Bob}}p$

$K_{\mathrm{Alice}}p$

Public Announcement Logic

Add a new dynamic operator:

$$\mathcal{M}, w \models [!\varphi]\psi \quad \text{iff} \quad \mathcal{M}, w \models \varphi \text{ implies } \mathcal{M}^{\varphi}, w \models \psi$$

where $\mathcal{M}^{\varphi}$ is a new model, keeping only the worlds were φ is true.

- originally by Plaza in 1989
- well-known axomatization via reduction axioms
- same expressivity as EL

Public Announcement Logic

Add a new dynamic operator:

$$\mathcal{M}, w \models [!\varphi]\psi \quad \text{iff} \quad \mathcal{M}, w \models \varphi \text{ implies } \mathcal{M}^{\varphi}, w \models \psi$$

where $\mathcal{M}^{\varphi}$ is a new model, keeping only the worlds were φ is true.

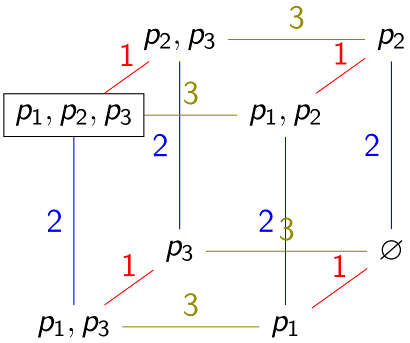
- originally by Plaza in 1989
- well-known axomatization via reduction axioms
- same expressivity as EL

Examples

$[!\ p \wedge q]K_i p$ is valid.

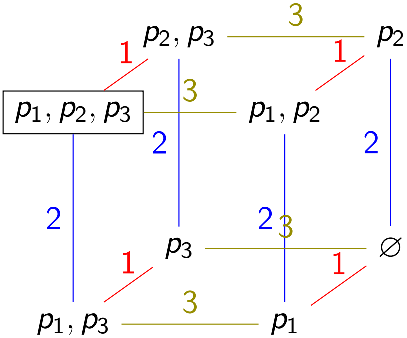
Muddy Children with Kripke Models

Let p_i denote that child i is muddy.



Muddy Children with Kripke Models

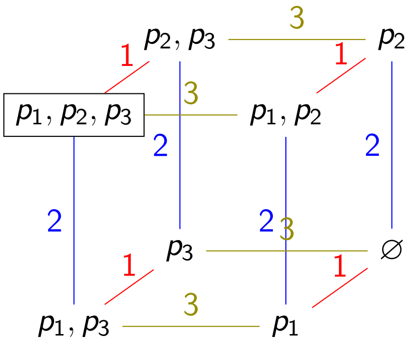
Let p_i denote that child i is muddy.



"At least one of you is muddy!"
$[!\ p_1 \vee p_2 \vee p_3]$
$\Rightarrow$

Muddy Children with Kripke Models

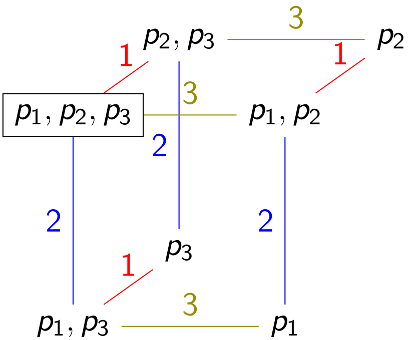
Let p_i denote that child i is muddy.



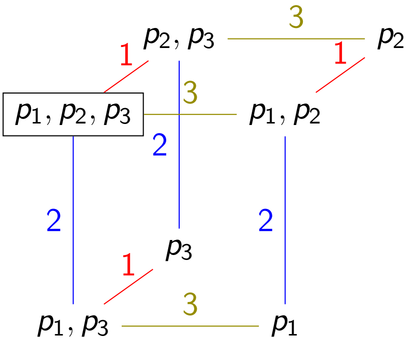
"At least one of you is muddy!"

$[!\, p_1 \vee p_2 \vee p_3]$

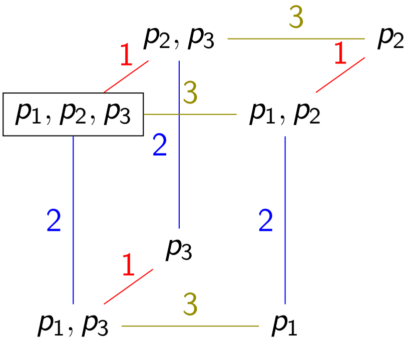
$\Rightarrow$



Muddy Children with Kripke Models II

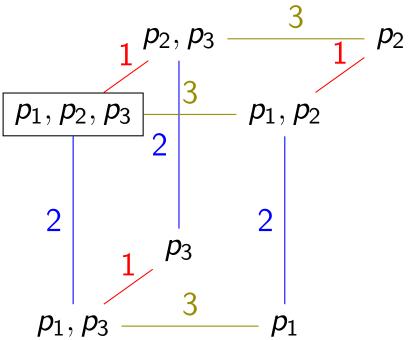


Muddy Children with Kripke Models II

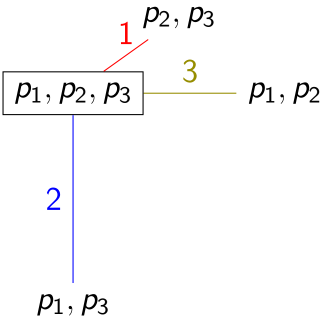


"Nobody knows
whether they
are muddy!"
$[! \bigwedge_i \neg K_i^? p_i]$
$\Rightarrow$

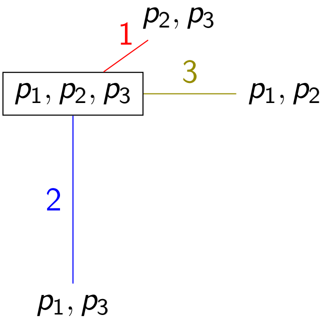
Muddy Children with Kripke Models II



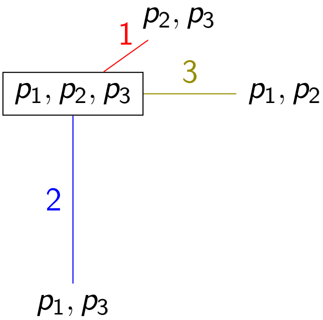
"Nobody knows whether they are muddy!"
$[! \bigwedge_i \neg K_i^? p_i]$
$\Rightarrow$



Muddy Children with Kripke Models III



Muddy Children with Kripke Models III

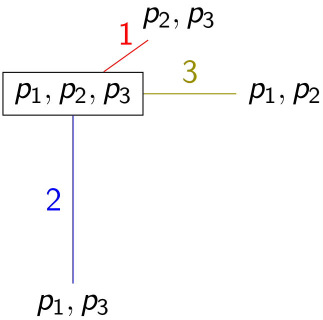


"Nobody knows whether they are muddy!"

$[! \bigwedge_i \neg K_i^? p_i]$

$\Rightarrow$

Muddy Children with Kripke Models III



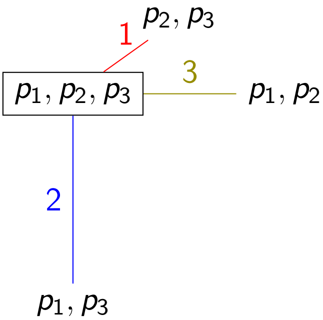
"Nobody knows whether they are muddy!"

$[!\ \bigwedge_i \neg K_i^? p_i]$

$\Rightarrow$

$\boxed{p_1, p_2, p_3}$

Muddy Children with Kripke Models III



"Nobody knows whether they are muddy!"

$[! \bigwedge_i \neg K_i^? p_i]$

$\Rightarrow$

$\boxed{p_1, p_2, p_3}$

Hence, in the original model $\mathcal{M}$ at the world w where all three are muddy:

$$\mathcal{M}, w \vDash [! \bigvee_i p_i][! \bigwedge_i \neg K_i^? p_i][! \bigwedge_i \neg K_i^? p_i] C(p_1 \wedge p_2 \wedge p_3)$$

Dynamic Epistemic Logic: Action Models

Action Models

$\mathcal{A} = (A, R, \mathrm{pre}, \mathrm{post})$ where

- A — set of atomic events
- $R_i \subseteq A \times A$ — indistinguishability relation
- $\mathrm{pre}\colon A \to \mathcal{L}$ — precondition function
- $\mathrm{post}\colon A \to P \to \mathcal{L}$ — postcondition function

Dynamic Epistemic Logic: Action Models

Action Models

$\mathcal{A} = (A, R, \mathsf{pre}, \mathsf{post})$ where

- A — set of atomic events
- $R_i \subseteq A \times A$ — indistinguishability relation
- $\mathsf{pre}\colon A \to \mathcal{L}$ — precondition function
- $\mathsf{post}\colon A \to P \to \mathcal{L}$ — postcondition function

Product Update

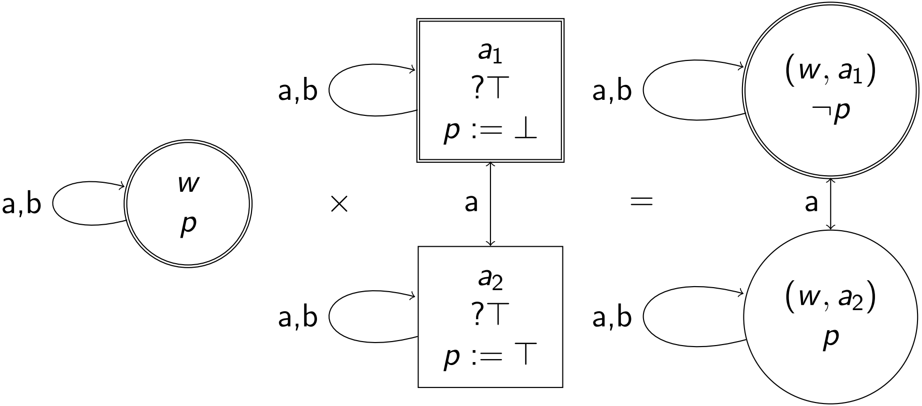
$\mathcal{M} \times \mathcal{A} := (W^{\mathsf{new}}, \mathcal{R}_i^{\mathsf{new}}, \mathsf{Val}^{\mathsf{new}})$ where

- $W^{\mathsf{new}} := \{(w, a) \in W \times A \mid \mathcal{M}, w \vDash \mathsf{pre}(a)\}$
- $\mathcal{R}_i^{\mathsf{new}} := \{((w, a), (v, b)) \mid \mathcal{R}_i wv \text{ and } R_i ab\}$
- $\mathsf{Val}^{\mathsf{new}}((w, a)) := \{p \in V \mid \mathcal{M}, w \vDash \mathsf{post}_a(p)\}$

$\mathcal{M}, v \vDash [\mathcal{A}, a]\varphi$ iff $\mathcal{M}, w \vDash \mathsf{pre}(a)$ implies $(\mathcal{M} \times \mathcal{A}, (w, a)) \vDash \varphi$

(Baltag, Moss & Solecki, 1998) and (van Benthem, van Eijck & Kooi, 2006)

DEL Example: Coin Flip hidden from a



$$\mathcal{M}, w \vDash K_a p \land K_b p \land [\mathcal{A}, a_1](K_b \neg p \land \neg K_a \neg p)$$

Two Perspectives: Dynamic / Temporal

- Dynamic Epistemic Logic: events are *model changing* operations
- Temporal Logics: time is a *relation inside the model*

DEL Applications

Fun puzzles:

- Russian Cards
- Muddy Children
- Sum and Product
- Drinking Logicians
- The Hardest Logic Puzzle Ever (Knights & Knaves)

But also more serious things:

- Epistemic Planning
- Protocol Verification
- Theory of Mind: Sally and Anne

Model Checking

Model Checking – The Task

Given a model and a formula, does it hold in the model?

$$\mathcal{M}, w \models \varphi \quad \text{or} \quad \mathcal{M}, w \not\models \varphi$$

???

In the case of DEL, φ might contain dynamic operators!

Agents and Formulas

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid K_i\varphi$$

```
type Prop = Int

type Ag = String

data Form = P Prop | Neg Form | Con Form Form | K Ag Form
  deriving (Eq,Ord,Show)

dis :: Form -> Form -> Form
dis f g = Neg (Con (Neg f) (Neg g))
```

Models

$$M = (W, R, V)$$

```
type World = Int

type Relations = [(Ag, [[World]])]

type Valuation = [(World, [Prop])]

data Model = Mo { worlds :: [World]
                , rel :: Relations
                , val :: Valuation }
  deriving (Eq,Ord,Show)
```

Note: We assume equivalence relations and encode them as `[[World]]`.

Semantics

Idea: translate the semantics definition of '$\vDash$' to a *recursive* function.

$$\begin{array}{lcl}
\mathcal{M}, w \vDash p & :\iff & p \in V(w) \\
\mathcal{M}, w \vDash \neg\varphi & :\iff & \text{not } \mathcal{M}, w \vDash \varphi \\
\mathcal{M}, w \vDash \varphi \wedge \psi & :\iff & \mathcal{M}, w \vDash \varphi \text{ and } \mathcal{M}, w \vDash \psi \\
\mathcal{M}, w \vDash K_i\varphi & :\iff & \mathcal{M}, w' \vDash \varphi \text{ for all } w' \text{ such that } R_i w w'
\end{array}$$

```
isTrue :: (Model,World) -> Form -> Bool
isTrue (m,w) (P p)     = p `elem` (val m ! w)
isTrue (m,w) (Neg f)   = not (isTrue (m,w) f)
isTrue (m,w) (Con f g) = isTrue (m,w) f && isTrue (m,w) g
isTrue (m,w) (K i f)   =
  and [ isTrue (m,w') f | w' <- (rel m ! i) ? w ]
```

Muddy Children in Haskell

8 worlds, 3 agents, 3 atomic propositions.

```haskell
muddy :: Model
muddy = Mo
  [0,1,2,3,4,5,6,7]
  [("1",[[0,4],[2,6],[3,7],[1,5]])
  ,("2",[[0,2],[4,6],[5,7],[1,3]])
  ,("3",[[0,1],[4,5],[6,7],[2,3]])]
  [(0,[])
  ,(1,[3])
  ,(2,[2])
  ,(3,[2, 3])
  ,(4,[1])
  ,(5,[1, 3])
  ,(6,[1, 2])
  ,(7,[1, 2, 3])]
```

Muddy Children in Haskell: examples

```
L5> isTrue (muddy,6) (Con (P 1) (P 2))
True
L5> isTrue (muddy,6) (K "1" (P 1))
False
L5> isTrue (muddy,6) (K "1" (P 2))
True
L5> isTrue (muddy,6) (K "3" (Con (P 1) (P 2)))
True
L5> isTrue (muddy,6) (K "3" (Neg (K "2" (P 2))))
True
```

Muddy Children in Haskell: `father`

$$p_1 \vee (p_2 \vee p_3)$$

```
father :: Form
father = dis (P 1) (dis (P 2) (P 3))
```

```
λ> map (\w->(w,isTrue (muddy, w) father)) (worlds muddy)
[(0,False),(1,True),(2,True),(3,True)
,(4,True),(5,True),(6,True),(7,True)]
```

Making Announcements

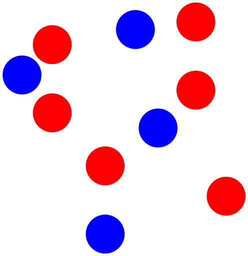
Exercise for you!

```
announce :: Model -> Form -> Model
announce oldModel f = Mo newWorlds newRel newVal where
  newWorlds = undefined
  newRel    = undefined
  newVal    = undefined

muddy2 :: Model
muddy2 = announce muddy father
```

Limits of explicit model checking

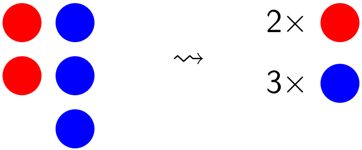
For large models ($\sim$ 1000 worlds) the explicit approach becomes really slow.
Example: runtime in seconds for n Muddy Children (i.e. 2^n worlds)::

n	DEMO-S5
3	0.000
6	0.012
8	0.273
10	8.424
11	46.530
12	228.055
13	1215.474



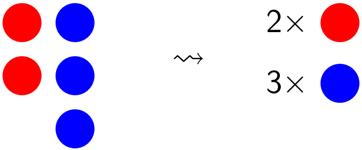
Symbolic Model Checking S5 PAL

Symbolic Model Checking: General Idea



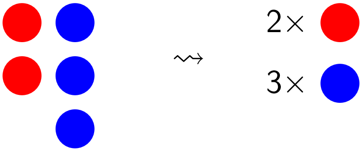
- Can we represent models in a more compact way?

Symbolic Model Checking: General Idea



- Can we represent models in a more compact way?
  ... such that we can still interpret all formulas?

Symbolic Model Checking: General Idea

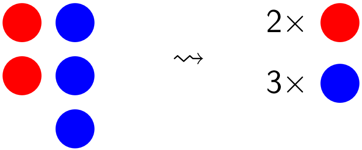


- Can we represent models in a more compact way?
  ... such that we can still interpret all formulas?

Yes! There is symbolic model checking for many temporal logics like LTL and CTL (Clarke et al. 1999) and also epistemic logics (Su et al. 2007).

Here: How to do it for DEL.

Symbolic Model Checking: General Idea



- Can we represent models in a more compact way?
  . . . such that we can still interpret all formulas?

Yes! There is symbolic model checking for many temporal logics like LTL and CTL (Clarke et al. 1999) and also epistemic logics (Su et al. 2007).

Here: How to do it for DEL.

1. Represent models symbolically with *knowledge structures*.
2. Translate EL and PAL to locally equivalent *boolean formulas*.
3. Use *Binary Decision Diagrams* to speed it up.

Symbolic Model Checking: General Idea in Haskell

Instead of listing all possible worlds explicitly . . .

```
KrM
  [0,1,2,3]
  [ ("Alice",[[0,1],[2,3]])
  , ("Bob"  ,[[0,2],[1,3]]) ]
  [ (0,[(P 1,False),(P 2,False)])
  , (1,[(P 1,False),(P 2,True )])
  , (2,[(P 1,True ),(P 2,False)])
  , (3,[(P 1,True ),(P 2,True )]) ]
```

. . . we list atomic propositions and who can observe them:

```
KnS [P 1,P 2] (boolBddOf Top) [ ("Alice",[P 1]), ("Bob",[P 2])]
```

Knowledge Structures

Knowledge Structures

$\mathcal{F} = (V, \theta, O_1, \ldots, O_n)$ where

V	*Vocabulary*	set of atoms
$\theta \in \mathcal{L}_B(V)$	*State Law*	boolean formula V
$O_i \subseteq V$	*Observables*	atoms seen by i

Knowledge Structures

Knowledge Structures

$\mathcal{F} = (V, \theta, O_1, \dots, O_n)$ where

- V *Vocabulary* set of atoms
- $\theta \in \mathcal{L}_B(V)$ *State Law* boolean formula V
- $O_i \subseteq V$ *Observables* atoms seen by i

The set of states is $\{s \subseteq V \mid s \vDash \theta\}$.
We call $(\mathcal{F}, s)$ where s is a state a scenario.

Knowledge Structures

Knowledge Structures

$\mathcal{F} = (V, \theta, O_1, \dots, O_n)$ where

▶ V	*Vocabulary*	set of atoms
▶ $\theta \in \mathcal{L}_B(V)$	*State Law*	boolean formula V
▶ $O_i \subseteq V$	*Observables*	atoms seen by i

$2\times$ 

$3\times$ 

The set of states is $\{s \subseteq V \mid s \vDash \theta\}$.

We call $(\mathcal{F}, s)$ where s is a state a scenario.

Knowledge Structures

Knowledge Structures

$\mathcal{F} = (V, \theta, O_1, \dots, O_n)$ where

- V — *Vocabulary* — set of atoms
- $\theta \in \mathcal{L}_B(V)$ — *State Law* — boolean formula V
- $O_i \subseteq V$ — *Observables* — atoms seen by i

$2\times$

$3\times$

The set of states is $\{s \subseteq V \mid s \vDash \theta\}$.
We call $(\mathcal{F}, s)$ where s is a state a scenario.

> *The world is everything that is the case.*
> *Die Welt ist alles, was der Fall ist.*

Ludwig Wittgenstein

Symbolic Semantics on Knowledge Structures

Easy:

- $(\mathcal{F}, s) \models p$ iff $p \in s$.
- $(\mathcal{F}, s) \models \neg\varphi$ iff not $(\mathcal{F}, s) \models \varphi$
- $(\mathcal{F}, s) \models \varphi \wedge \psi$ iff $(\mathcal{F}, s) \models \varphi$ and $(\mathcal{F}, s) \models \psi$

I know something iff it follows from my observations:

- $(\mathcal{F}, s) \models K_i\varphi$ iff for all $s' \vDash \theta$, if $s \cap O_i = s' \cap O_i$, then $(\mathcal{F}, s') \models \varphi$.

Updates restrict the set of states:

- $(\mathcal{F}, s) \models [!\psi]\varphi$ iff $(\mathcal{F}, s) \models \psi$ implies $(\mathcal{F}^\psi, s) \models \varphi$ where $\|\psi\|_\mathcal{F}$ will be defined later and

$$\mathcal{F}^\psi := (V, \theta \wedge \|\psi\|_\mathcal{F}, O_1, \cdots, O_n)$$

Knowledge Structures

Example

$$\mathcal{F} = (V = \{p\}, \theta = \top, O_1 = \{p\}, O_2 = \varnothing)$$

States: $\varnothing$, $\{p\}$

Some facts:

- $\mathcal{F}, \varnothing \vDash \neg p \wedge K_1 \neg p \wedge \neg K_2 \neg p$
- $\mathcal{F}, \{p\} \vDash p \wedge K_1 p \wedge \neg K_2 p$
- $\mathcal{F}, \{p\} \vDash [!p] K_2 p$
  because $\mathcal{F}^p = (V = \{p\}, \theta = p, O_1 = \{p\}, O_2 = \varnothing)$

Implementation of Knowledge Structures and Semantics

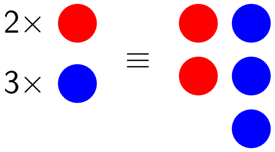
```
data KnowStruct = KnS [Prp] Bdd [(Agent,[Prp])]
type KnState    = [Prp]
type KnowScene  = (KnowStruct,KnState)

eval :: KnowScene -> Form -> Bool
eval _        Top              = True
eval (_,s)    (PrpF p)         = p `elem` s
eval (kns,s) (Neg form)        = not $ eval (kns,s) form
eval (kns,s) (Conj forms)      = all (eval (kns,s)) forms
eval scn     (Impl f g)        = not (eval scn f) || eval scn g
eval (kns@(KnS _ _ obs),s) (K i form) = all (\s' -> eval (kns,s') form) theres where
  theres = filter (\s' -> seteq (s' `intersect` oi) (s `intersect` oi)) (statesOf kns)
  oi = obs ! i
eval scn (PubAnnounce form1 form2) =
  not (eval scn form1) || eval (update scn form1) form2
```

Only parts of the code will be here on the slides. See https://github.com/jrclogic/SMCDEL/blob/master/src/SMCDEL/Symbolic/S5.hs

From Knowledge Structures to Kripke Models (easy direction)

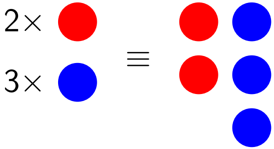
Theorem

For every knowledge structure there is an equivalent S5 Kripke Model.



From Knowledge Structures to Kripke Models (easy direction)

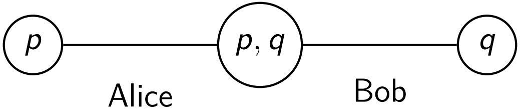
Theorem
For every knowledge structure there is an equivalent S5 Kripke Model.



Example

$$\mathcal{F} = (V = \{p, q\},\ \theta = p \vee q,\ O_{\text{Alice}} = \{p\},\ O_{\text{Bob}} = \{q\})$$

is equivalent to



Implementation: KNS $\to$ Kripke

Let $W := \{s \subseteq V \mid s \vDash \theta\}$, $V = \text{id}$ and $R_i st$ iff $s \cap O_i = t \cap O_i$.

```
knsToKripkeWithG :: KnowStruct -> (KripkeModelS5, StateMap)
knsToKripkeWithG kns@(KnS ps _ obs) =
  (KrMS5 worlds rel val, g) where
    g w    = statesOf kns !! w
    lav    = zip (statesOf kns) [0..(length (statesOf kns)-1)]
    val    = map ( \(s,n) -> (n,state2kripkeass s) ) lav where
      state2kripkeass s = map (\p -> (p, p `elem` s)) ps
    rel    = [(i,rfor i) | i <- map fst obs]
    rfor i = map (map snd) (groupBy ( \ (x,_) (y,_) -> x==y ) (sort pairs))
     where  pairs = map (\s -> (s `intersect` (obs ! i), lav ! s))
                        (statesOf kns)
    worlds = map snd lav
```

See https://github.com/jrclogic/SMCDEL/blob/master/src/SMCDEL/Translations/S5.hs

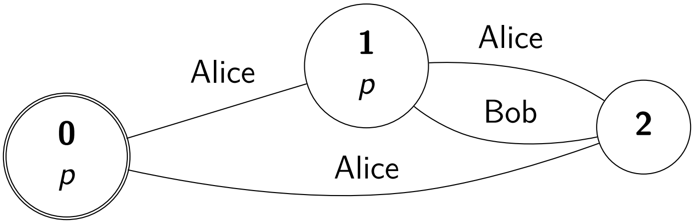
From Kripke Models to Knowledge Structures (tricky direction)

Theorem: For every S5 Kripke Model $\mathcal{M}$ there is an equivalent knowledge structure $\mathcal{F}$ such that $\mathcal{M}, w \vDash \varphi$ iff $\mathcal{F}, s_w \vDash \varphi$.

From Kripke Models to Knowledge Structures (tricky direction)

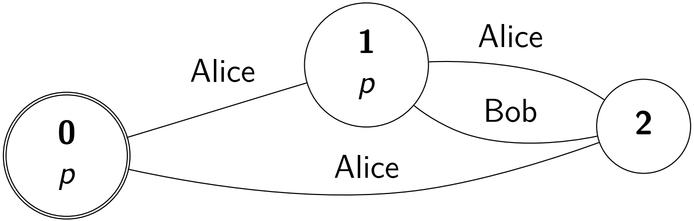
Theorem: For every S5 Kripke Model $\mathcal{M}$ there is an equivalent knowledge structure $\mathcal{F}$ such that $\mathcal{M}, w \vDash \varphi$ iff $\mathcal{F}, s_w \vDash \varphi$.

Proof. Problematic cases look like this:



From Kripke Models to Knowledge Structures

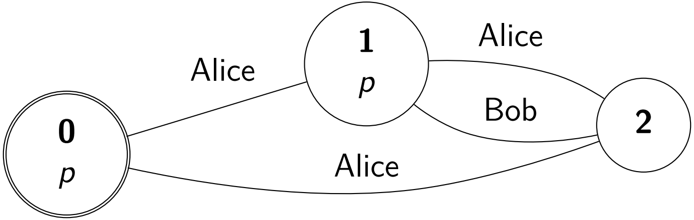
Proof. (continued)



Trick: Add propositions to distinguish all equivalence classes.

From Kripke Models to Knowledge Structures

Proof. (continued)



is equivalent to

$$\left(\; V = \{p, p_2\},\;\; \theta = p_2 \rightarrow p,\;\; O_{\mathrm{Alice}} = \varnothing,\;\; O_{\mathrm{Bob}} = \{p_2\} \;\right)$$

actual state: $\{p, p_2\}$

$\square$

Implementation: Kripke $\to$ KNS

```haskell
kripkeToKnsWithG :: KripkeModelS5 -> (KnowStruct, StateMap)
kripkeToKnsWithG m@(KrMS5 worlds rel val) = (KnS ps law obs, g) where
  v          = vocabOf m
  ags        = map fst rel
  newpstart  = fromEnum $ freshp v -- start counting new propositions here
  amount i   = ceiling (logBase 2 (fromIntegral $ length (rel ! i)) :: Float) -- = |O_i|
  newpstep   = maximum [ amount i | i <- ags ]
  newps i    = map (\k -> P (newpstart + (newpstep * inum) +k)) [0..(amount i - 1)] -- O_i
    where (Just inum) = elemIndex i (map fst rel)
  copyrel i = zip (rel ! i) (powerset (newps i)) -- label equiv.classes with P(O_i)
  gag i w    = snd $ head $ filter (\(ws,_) -> w `elem` ws) (copyrel i)
  g w        = filter (apply (val ! w)) v ++ concat [ gag i w | i <- ags ]
  ps         = v ++ concat [ newps i | i <- ags ]
  law        = disSet [ booloutof (g w) ps | w <- worlds ]
  obs        = [ (i,newps i) | i<- ags ]
```

So what, Kripke Models and knowledge structures are the same?!

Everything is boolean!

Definition

Given a knowledge structure $\mathcal{F} = (V, \theta, O)$,
we define a *local translation from epistemic to boolean formulas*:

- $\|p\|_{\mathcal{F}} := p$
- $\|\neg\varphi\|_{\mathcal{F}} := \neg\|\varphi\|_{\mathcal{F}}$
- $\|\varphi_1 \wedge \varphi_2\|_{\mathcal{F}} := \|\varphi_1\|_{\mathcal{F}} \wedge \|\varphi_2\|_{\mathcal{F}}$ 🤔
- $\|K_i\varphi\|_{\mathcal{F}} :=$

Everything is boolean!

Definition

Given a knowledge structure $\mathcal{F} = (V, \theta, O)$,
we define a *local translation from epistemic to boolean formulas*:

- $\|p\|_{\mathcal{F}} := p$
- $\|\neg\varphi\|_{\mathcal{F}} := \neg\|\varphi\|_{\mathcal{F}}$
- $\|\varphi_1 \wedge \varphi_2\|_{\mathcal{F}} := \|\varphi_1\|_{\mathcal{F}} \wedge \|\varphi_2\|_{\mathcal{F}}$
- $\|K_i\varphi\|_{\mathcal{F}} :=$ 🤔

where boolean quantification is defined by substitution:
$\forall p\varphi := [p/\top]\varphi \wedge [p/\bot]\varphi$

Example: $\forall p(p \vee q) = (\top \vee q) \wedge (\bot \vee q) \equiv \top \wedge q \equiv q$

Everything is boolean!

Definition

Given a knowledge structure $\mathcal{F} = (V, \theta, O)$,
we define a *local translation from epistemic to boolean formulas*:

- $\|p\|_{\mathcal{F}} := p$
- $\|\neg\varphi\|_{\mathcal{F}} := \neg\|\varphi\|_{\mathcal{F}}$
- $\|\varphi_1 \wedge \varphi_2\|_{\mathcal{F}} := \|\varphi_1\|_{\mathcal{F}} \wedge \|\varphi_2\|_{\mathcal{F}}$
- $\|K_i\varphi\|_{\mathcal{F}} := \forall(V \setminus O_i)(\theta \rightarrow \|\varphi\|_{\mathcal{F}})$

where boolean quantification is defined by substitution:
$\forall p\varphi := [p/\top]\varphi \wedge [p/\bot]\varphi$

Example: $\forall p(p \vee q) = (\top \vee q) \wedge (\bot \vee q) \equiv \top \wedge q \equiv q$

Announcements on Knowledge Structures

To announce a formula, **add its boolean equivalent to the state law**.

Consider a knowledge structure $\mathcal{F} = (V, \theta, O)$.

We define:

$\mathcal{F}, s \vDash [!\varphi]\psi \qquad \text{iff} \qquad \mathcal{F}, s \vDash \varphi$ implies $(V, \theta \land \|\varphi\|_{\mathcal{F}}, O), s \vDash \psi$.

Reducing DEL model checking to Boolean evaluation

Theorem

For all scenarios $(\mathcal{F}, s)$ and all formulas φ:

$$\mathcal{F}, s \vDash \varphi \iff s \vDash \|\varphi\|_{\mathcal{F}}$$

Reducing DEL model checking to Boolean evaluation

Theorem

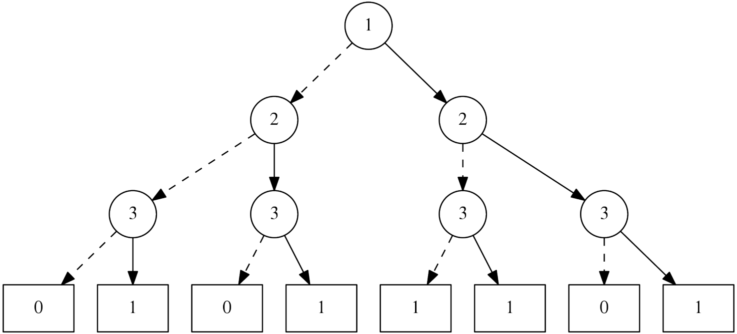
For all scenarios $(\mathcal{F}, s)$ and all formulas φ:

$$\mathcal{F}, s \vDash \varphi \iff s \vDash \|\varphi\|_{\mathcal{F}}$$

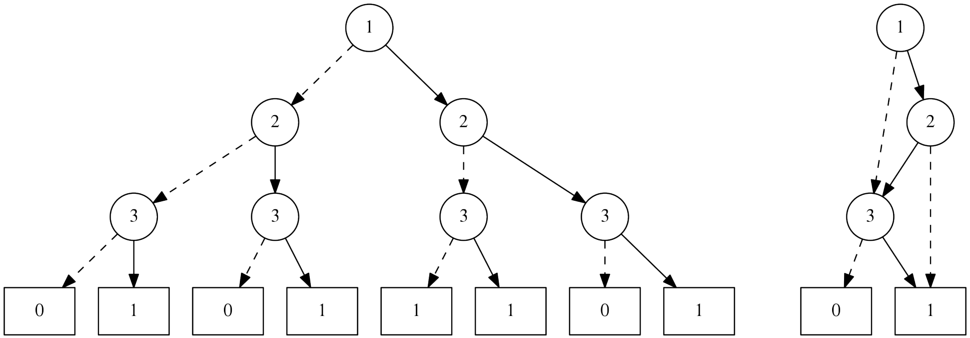
But why is it useful to go from DEL to boolean formulas?

Binary Decision Diagrams

BDD Example



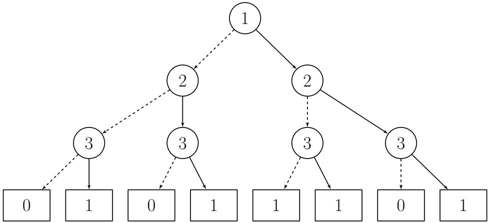
BDD Example

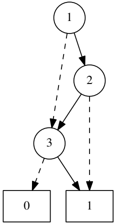


Truth Tables are dead, long live trees

Definition: A Binary Decision Diagram for the variables V is a directed acyclic graph where non-terminal nodes are from V with two outgoing edges and terminal nodes are $\top$ or $\bot$.

- All boolean functions can be represented like this.
- Ordered: Variables in a given order, maximally once.
- Reduced: No redundancy, identify isomorphic subgraphs.
- By "BDD" we always mean an ordered and reduced BDD.





Read the classic Bryant 1986 for more details!

BDD Magic from 1986

How long do you need to compare two formulas?

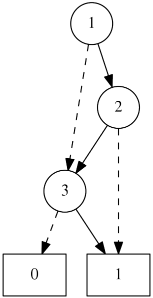
$$p_3 \vee \neg(p_1 \to p_2) \quad ??? \quad \neg(p_1 \wedge \neg p_2) \to p_3$$

BDD Magic from 1986

How long do you need to compare two formulas?

$$p_3 \vee \neg(p_1 \to p_2) \quad ??? \quad \neg(p_1 \wedge \neg p_2) \to p_3$$

On the right ~~are~~ is their BDD~~s~~.

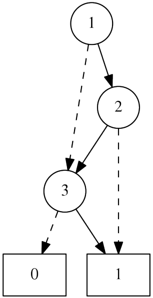


BDD Magic from 1986

How long do you need to compare two formulas?

$$p_3 \lor \neg(p_1 \to p_2) \quad ??? \quad \neg(p_1 \land \neg p_2) \to p_3$$

On the right ~~are~~ is their BDD~~s~~.



This was not an accident, BDDs are *canonical*.

Theorem

$\varphi \equiv \psi \quad \iff \quad \text{BDD}(\varphi) = \text{BDD}(\psi)$

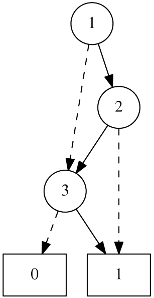
Equivalence checks are free and we can quickly get $\text{BDD}(\neg\varphi)$, $\text{BDD}(\varphi \land \psi)$, etc.

BDD Magic from 1986

How long do you need to compare two formulas?

$$p_3 \vee \neg(p_1 \rightarrow p_2) \quad ??? \quad \neg(p_1 \wedge \neg p_2) \rightarrow p_3$$

On the right ~~are~~ is their BDD~~s~~.



This was not an accident, BDDs are *canonical*.

Theorem

$$\varphi \equiv \psi \quad \iff \quad \mathrm{BDD}(\varphi) = \mathrm{BDD}(\psi)$$

Equivalence checks are free and we can quickly get $\mathrm{BDD}(\neg\varphi)$, $\mathrm{BDD}(\varphi \wedge \psi)$, etc.

Randal E. Bryant (1986): *Graph-Based Algorithms for Boolean Function Manipulation* https://doi.org/bnrh63

Implementation: Translation to BDDs

```haskell
import Data.HasCacBDD -- (var,neg,conSet,forallSet,...)

bddOf :: KnowStruct -> Form -> Bdd
bddOf _   Top              = top
bddOf _   (PrpF (P n))     = var n
bddOf kns (Neg form)       = neg $ bddOf kns form
bddOf kns (Conj forms)     = conSet $ map (bddOf kns) forms
bddOf kns (Impl f g)       = imp (bddOf kns f) (bddOf kns g)
bddOf kns@(KnS allprops lawbdd obs) (K i form) =
  forallSet otherps (imp lawbdd (bddOf kns form)) where
    otherps = map (\(P n) -> n) $ allprops \\ obs ! i
bddOf kns (PubAnnounce form1 form2) =
  imp (bddOf kns form1) (bddOf (update kns form1) form2)
```

See https://github.com/jrclogic/SMCDEL/blob/master/src/SMCDEL/Symbolic/S5.hs

Putting it all together

To model check whether $\mathcal{F}, s \vDash \varphi$ ...

1. Translate φ to a BDD with respect to $\mathcal{F}$.
2. Restrict the BDD to s.
3. Return the resulting constant.

```
evalViaBdd :: KnowScene -> Form -> Bool
evalViaBdd (kns,s) f =
  evaluateFun (bddOf kns f) (\n -> P n `elem` s)
```

Examples and Benchmarks

Symbolic Muddy Children

Initial knowledge structure:

$$\mathcal{F} = (\{p_1, p_2, p_3\}, \top, O_1 = \{p_2, p_3\}, O_2 = \{p_1, p_3\}, O_3 = \{p_1, p_2\})$$

After the third announcement the children know their own state:

$$\varphi = [!(p_1 \vee p_2 \vee p_3)][!\bigwedge_i \neg(K_i p_i \vee K_i \neg p_i)][!\bigwedge_i \neg(K_i p_i \vee K_i \neg p_i)](\bigwedge_i (K_i p_i))$$

Symbolic Example: Muddy Children I

$$\mathcal{F}_0 = \left(V = \{p_1, p_2, p_3\},\ \theta_0 = \top,\ \begin{array}{l} O_1 = \{p_2, p_3\} \\ O_2 = \{p_1, p_3\} \\ O_3 = \{p_1, p_2\} \end{array} \right)$$

Symbolic Example: Muddy Children I

$$\mathcal{F}_0 = \left(V = \{p_1, p_2, p_3\},\ \theta_0 = \top,\ \begin{array}{l} O_1 = \{p_2, p_3\} \\ O_2 = \{p_1, p_3\} \\ O_3 = \{p_1, p_2\} \end{array} \right)$$

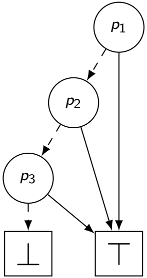
$\Downarrow$ "At least one of you is muddy."

$$\mathcal{F}_1 = \left(V = \{p_1, p_2, p_3\},\quad \theta_1 = (p_1 \vee p_2 \vee p_3),\quad \begin{array}{l} O_1 = \{p_2, p_3\} \\ O_2 = \{p_1, p_3\} \\ O_3 = \{p_1, p_2\} \end{array} \right)$$

Symbolic Example: Muddy Children II

In the actual implementation, we use a BDD for $\theta_1 = p_1 \lor p_2 \lor p_3$, not a formula:

$$\mathcal{F}_1 = \left(V = \{p_1, p_2, p_3\}, \quad \theta_1 = \right.$$

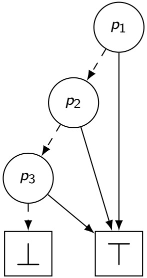


$$\left. , \quad \begin{array}{l} O_1 = \{p_2, p_3\} \\ O_2 = \{p_1, p_3\} \\ O_3 = \{p_1, p_2\} \end{array} \right)$$

Symbolic Example: Muddy Children II

In the actual implementation, we use a BDD for $\theta_1 = p_1 \vee p_2 \vee p_3$, not a formula:

$$\mathcal{F}_1 = \left(V = \{p_1, p_2, p_3\}, \quad \theta_1 = \right.$$



$$\left. , \quad \begin{array}{l} O_1 = \{p_2, p_3\} \\ O_2 = \{p_1, p_3\} \\ O_3 = \{p_1, p_2\} \end{array} \right)$$

"Do you know if you are muddy?" ... Nobody reacts.

This is an announcement of $\bigwedge_{i \in I}(\neg(K_i p_i \vee K_i \neg p_i))$.

Symbolic Example: Muddy Children III

Wanted: Boolean equivalent of $\bigwedge_{i \in I}(\neg(K_i p_i \vee K_i \neg p_i))$.

$$
\begin{aligned}
\|K_1 p_1\|_{\mathcal{F}_1} &\equiv \forall (V \setminus O_1)(\theta_1 \to \|p_1\|_{\mathcal{F}_1}) \\
&\equiv \forall p_1((p_1 \vee p_2 \vee p_3) \to p_1) \\
&\equiv ((\top \vee p_2 \vee p_3) \to \top) \wedge ((\bot \vee p_2 \vee p_3) \to \bot) \\
&\equiv \neg(p_2 \vee p_3)
\end{aligned}
$$

Symbolic Example: Muddy Children III

Wanted: Boolean equivalent of $\bigwedge_{i \in I}(\neg(K_i p_i \vee K_i \neg p_i))$.

$$\begin{aligned}
\|K_1 p_1\|_{\mathcal{F}_1} &\equiv \forall(V \setminus O_1)(\theta_1 \rightarrow \|p_1\|_{\mathcal{F}_1}) \\
&\equiv \forall p_1((p_1 \vee p_2 \vee p_3) \rightarrow p_1) \\
&\equiv ((\top \vee p_2 \vee p_3) \rightarrow \top) \wedge ((\bot \vee p_2 \vee p_3) \rightarrow \bot) \\
&\equiv \neg(p_2 \vee p_3)
\end{aligned}$$

$$\begin{aligned}
\|K_1 \neg p_1\|_{\mathcal{F}_1} &\equiv \forall(V \setminus O_1)(\theta_1 \rightarrow \|\neg p_1\|_{\mathcal{F}_1}) \\
&\equiv \forall p_1((p_1 \vee p_2 \vee p_3) \rightarrow \neg p_1) \\
&\equiv ((\top \vee p_2 \vee p_3) \rightarrow \neg\top) \wedge ((\bot \vee p_2 \vee p_3) \rightarrow \neg\bot) \\
&\equiv \bot
\end{aligned}$$

and analogous for $K_2 p_2$, $K_2 \neg p_2$, $K_3 p_3$ and $K_3 \neg p_3$ ...

Example: Symbolic Muddy Children III

... together we get:

$$\left\| \bigwedge_{i \in I} (\neg(K_i p_i \vee K_i \neg p_i)) \right\|_{\mathcal{F}_1} \equiv (p_2 \vee p_3) \wedge (p_1 \vee p_3) \wedge (p_1 \vee p_2)$$

"Nobody knows their own state."

is locally equivalent to

"At least two are muddy."

Example: Symbolic Muddy Children III

... together we get:

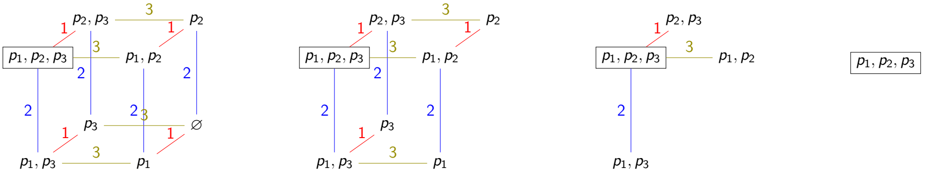
$$\left\| \bigwedge_{i \in I} (\neg(K_i p_i \vee K_i \neg p_i)) \right\|_{\mathcal{F}_1} \equiv (p_2 \vee p_3) \wedge (p_1 \vee p_3) \wedge (p_1 \vee p_2)$$

"Nobody knows their own state."

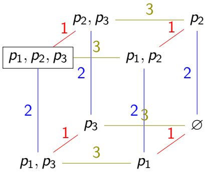
is locally equivalent to

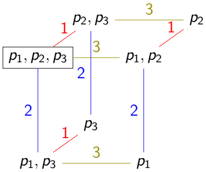
"At least two are muddy."

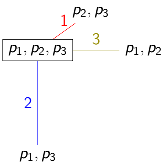
Explicit and Symbolic Muddy Children

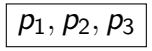


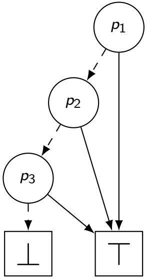
Explicit and Symbolic Muddy Children

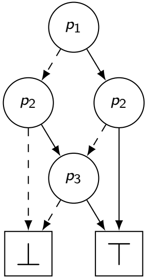


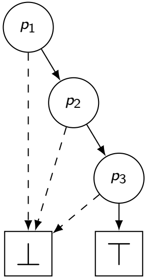




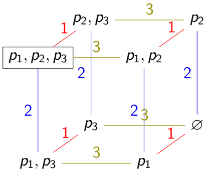


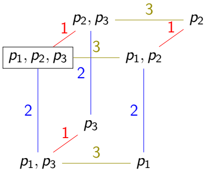


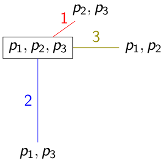


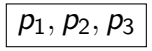


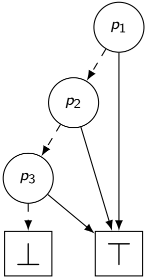
Explicit and Symbolic Muddy Children

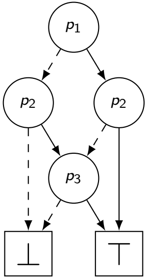


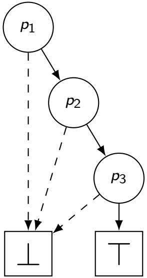












Note: $V = \{p_1, p_2, p_3\}$ and $O_1 = \{p_2, p_3\}$ etc. never change.

Muddy Children

Runtime in seconds:

n	DEMO-S5	SMCDEL
3	0.000	0.000
6	0.012	0.002
8	0.273	0.004
10	8.424	0.008
11	46.530	0.011
12	228.055	0.015
13	1215.474	0.019
20		0.078
40		0.777
60		2.563
80		6.905

How to use SMCDEL

The easy way: SMCDEL web interface at https://w4eg.de/malvin/illc/smcdelweb

```
VARS 1,2,3
LAW  Top
OBS  alice: 2,3
     bob: 1,3
     carol: 1,2
VALID?
  [ ! (1|2|3) ]
  [ ! ( (~ (alice knows whether 1))
      & (~ (bob   knows whether 2))
      & (~ (carol knows whether 3)) ) ] (1 & 2 & 3)
```

The hard way: `import SMCDEL.Symbolic.S5` etc. Then define abbreviations and generate larger models using Haskell. See https://github.com/jrclogic/SMCDEL/tree/master/src/SMCDEL/Examples

Beyond S5 PAL

Action Models and Product Update

Action Model: $\mathcal{A} = (A, S_i, \mathrm{pre})$

A	set of actions
$S_i \subseteq A \times A$	indistinguishability relation
$\mathrm{pre} : A \to \mathcal{L}$	preconditions

Product Update:
$\mathcal{M} \otimes \mathcal{A} := (W', R', V')$ where

- $W' = \{(w, a) \in W \times A \mid \mathcal{M}, w \vDash \mathrm{pre}(a)\}$
- $R'_i(s, a)(t, b)$ iff $R_i st$ and $S_i ab$
- $V'(w, a) = V(w)$ no factual change

Semantics:
$\mathcal{M}, w \vDash [\mathcal{A}, a]\varphi$ iff $\mathcal{M}, w \vDash \mathrm{pre}(a)$ implies $\mathcal{M} \otimes \mathcal{A}, (w, a) \vDash \varphi$

Knowledge Transformers

Knowledge Transformer: $\mathcal{X} = (V^+, \mu, O_1^+, \ldots, O_n^+)$

V^+	*New Vocabulary*	new propositional variables
μ	*Event Law*	a formula over $V \cup V^+$
$O_i^+ \subseteq V^+$	*Observables*	what can i observe?

Transformation: Given $\mathcal{F} = (V, \theta, O_1, \ldots, O_n)$ and $\mathcal{X} = (V^+, \mu, O_1^+, \ldots, O_n^+)$, define

$$\mathcal{F} \otimes \mathcal{X} := (V \cup V^+, \theta \land ||\mu||_{\mathcal{F}}, O_1 \cup O_1^+, \ldots, O_n \cup O_n^+)$$

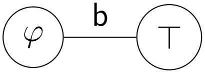
Event: $(\mathcal{X}, x)$ where $x \subseteq V^+$

Knowledge Transformers

Examples:

- public announcement: $\mathcal{X} = (\varnothing, \varphi, \varnothing, \varnothing)$
- (almost) private announcement of φ to a:

$$\mathcal{X} = (\{p\}, p \rightarrow \varphi, O_a = \{p\}, O_b = \varnothing)$$



Theorem: For every S5 action model $\mathcal{A}$ there is a transformer $\mathcal{X}$ (and vice versa) such that for every equivalent $\mathcal{M}$ and $\mathcal{F}$:

$$\mathcal{M} \otimes \mathcal{A}, (w, a) \vDash \varphi \iff \mathcal{F} \otimes \mathcal{X}, s \cup x \vDash \varphi$$

Non-S5: Belief as KD45

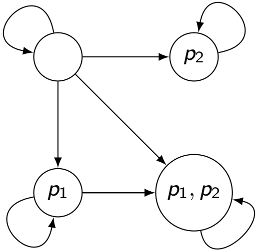
A crucial difference between Knowledge and Belief is Truth.

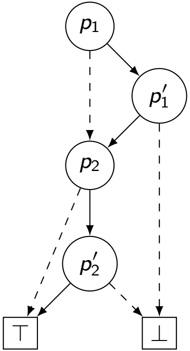
We assume $K\varphi \to \varphi$ but in general not $B\varphi \to \varphi$.

$\Rightarrow$ Kripke Models for Belief are not reflexive.

Non-S5 Arbitrary Relations with BDDs

We can replace O_i with a BDD Ω_i to describe any relation.
Trick: Use copy-propositions to describe reachable worlds.





NOTE: this examples has a mistake, see https://malv.in/phdthesis/gattinger-thesis-errata.pdf

For every agent we replace O_i with a BDD Ω_i.

Now translate $\Box_i \varphi$ to: $\forall \vec{p'} (\theta' \rightarrow (\Omega_i(\vec{p}, \vec{p'}) \rightarrow (\|\varphi\|_{\mathcal{F}})'))$

Summary

Summary

- Representation matters!
- Model Checking: decide whether $\mathcal{M}, w \vDash \varphi$.
- Binary Decision Diagrams: a data structure for boolean ~~formulas~~ functions.
- Symbolic structures can encode Kripke models for faster model checking.

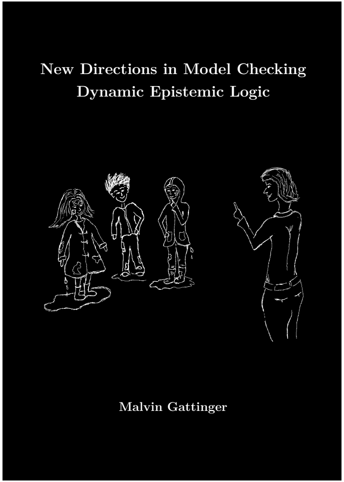
Summary

- Representation matters!
- Model Checking: decide whether $\mathcal{M}, w \vDash \varphi$.
- Binary Decision Diagrams: a data structure for boolean ~~formulas~~ functions.
- Symbolic structures can encode Kripke models for faster model checking.

- Further topics:
  - non-equivalence relations: K instead of S5
  - beyond public announcements: action models
  - alternative: "succinct" models using mental programs
  - Epistemic Planning (see Bolander, Schwarzentruber, etc.)
  - ...

References

- https://github.com/jrclogic/SMCDEL
  (Literate Haskell documentation in `SMCDEL.pdf`.)
- *Symbolic Model Checking for Dynamic Epistemic Logic — S5 and Beyond*,
  Journal of Logic and Computation, 2017.
  https://doi.org/10.1093/logcom/exx038
- *New Directions in Model Checking Dynamic Epistemic Logic*, PhD thesis, Amsterdam, 2018.
  https://malv.in/phdthesis/
- *Towards Symbolic and Succinct Perspective Shifts*,
  Epistemic Planning workshop at ICAPS 2020.
  https://doi.org/10.5281/zenodo.4767546



BONUS CONTENT

Russian Cards

A puzzle:

> *Seven cards, enumerated from 1 to 7, are distributed between Alice, Bob and Carol. Alice and Bob both receive three cards and Carol one card. It is common knowledge which cards exist and how many cards each agent has. Everyone knows their own but not the others' cards. The goal of Alice and Bob now is to learn each others cards without Carol learning their cards.*
> *They are only allowed to communicate via public announcements.*

Russian Cards: Solution

Alice: "My set of cards is 123, 145, 167, 247 or 356."

Bob: "Crow has card 7."

Russian Cards: Solution

Alice: “My set of cards is 123, 145, 167, 247 or 356.”

Bob: “Crow has card 7.”

There are 102 such “safe announcements” which van Ditmarsch et al. (2003) had to find and check by hand.

With symbolic model checking this takes 4 seconds.

Sum and Product

The puzzle from Freudenthal 1969 (translated from Dutch):

A says to S and P: I chose two numbers x, y *such that* $1 < x < y$ *and* $x + y \leq 100$. *I will tell* $s = x + y$ *to S alone, and* $p = xy$ *to P alone. These messages will stay secret. But you should try to calculate the pair* (x, y).

He does as announced. Now follows this conversation:

1. P says: I do not know it.

2. S says: I knew that.

3. P says: Now I know it.

4. S says: No I also know it.

Determine the pair (x, y).

Sum and Product: Encoding numbers

```
pairs :: [(Int, Int)] -- possible pairs 1<x<y, x+y<=100
pairs = [(x,y) | x<-[2..100], y<-[2..100], x<y, x+y<=100]

xProps, yProps, sProps, pProps :: [Prp]
xProps = [(P  1)..(P  7)] -- 7 propositions to label [2..100]
yProps = [(P  8)..(P 14)]
sProps = [(P 15)..(P 21)]
pProps = [(P 22)..(P (21+amount))]
  where amount = ceiling (logBase 2 (50*50) :: Double)

xIs, yIs, sIs, pIs :: Int -> Form
xIs n = booloutofForm (powerset xProps !! n) xProps
yIs n = booloutofForm (powerset yProps !! n) yProps
sIs n = booloutofForm (powerset sProps !! n) sProps
pIs n = booloutofForm (powerset pProps !! n) pProps

xyAre :: (Int,Int) -> Form
xyAre (n,m) = Conj [ xIs n, yIs m ]
```

Sum and Product: Benchmark

BDDs don't like products:

```
Benchmark bench-sumandproduct: RUNNING...
Benchmarking the complete run.
*** Running DEMO_S5 ***
Mo [(4,13)] [Ag 0,Ag 1] [] [ (Ag 0,[[(4,13)]])
                              ,(Ag 1,[[(4,13)]])]  [(4,13)]
This took 0.964665s seconds.

*** Running SMCDEL ***
x = 4, y = 13, x+y = 17 and x*y = 52
This took 1.632393s seconds.
```

Dining Cryptographers

> *Suppose Jonathan, Patrick and Bo had a very fancy diner. The waiter comes in and tells them that it has already been paid.*
> *They want to find out if it was one of them or the ILLC. However, if one of them paid, they also respect the wish of that person to stay anonymous. That is, they do not want to know who of them paid if it was one of them.*

This puzzle was solved by David Chaum in his "Dining Cryptographers" protocol.

Dining Cryptographers

> *Suppose Jonathan, Patrick and Bo had a very fancy diner. The waiter comes in and tells them that it has already been paid.*
> *They want to find out if it was one of them or the ILLC. However, if one of them paid, they also respect the wish of that person to stay anonymous. That is, they do not want to know who of them paid if it was one of them.*

This puzzle was solved by David Chaum in his “Dining Cryptographers” protocol.

SMCDEL can check the case with 160 agents (and a lot of coins) in 10 seconds.

Digression: Comparing DEL and ETL

Scenarios and protocols like the Dining Dryptographers can be formalized in temporal logics (LTL,CTLK,...) and in DEL.

With SMCDEL we can now also check the DEL variant quickly.

This motivates many questions:

- When are two formalizations of the same protocol equivalent? [@vB2009merging, @ditmarsch2013connecting]
- Which formalizations are more intuitive?
- What is faster
  - for your computer to model check?
  - for you to write down formulas?

Type Safe BDD manipulation

(This is about Belief Structures.)

Note that φ and φ' etc. are formulas in different languages, but we can use the same type `Form` and `Bdd` in Haskell for it.

This will lead to disaster.

Type Safe BDD manipulation

(This is about Belief Structures.)

Note that φ and φ' etc. are formulas in different languages, but we can use the same type `Form` and `Bdd` in Haskell for it.

This will lead to disaster.

The following type `RelBDD` is in fact just a newtype of `Bdd`. Tags (aka labels) from the module Data.Tagged can be used to distinguish objects of the same type which should not be combined or mixed. Making these differences explicit at the type level can rule out certain mistakes already at compile time which otherwise might only be discovered at run time or not at all.

Type Safe BDD manipulation (continued)

The use case here is to distinguish BDDs for formulas over different vocabularies, i.e.~sets of atomic propositions. For example, the BDD of p_1 in the standard vocabulary V uses the variable 1, but in the vocabulary of $V \cup V'$ the proposition p_1 is mapped to variable 3 while p_1' is mapped to 4. This is implemented in the mv and cp functions above which we are now going to lift to BDDs.

If RelBDD and Bdd were synonyms (as it was the case in a previous version of this file) then it would be up to us to ensure that BDDs meant for different vocabularies would not be combined. Taking the conjunction of the BDD of p in V and the BDD of p_2 in $V \cup V'$ just makes no sense — one BDD first needs to be translated to the vocabulary of the other — but as long as the types match Haskell would happily generate the chaotic conjunction.

Type Safe BDD manipulation (continued continued)

To catch these problems at compile time we now distinguish Bdd and RelBDD@. In principle this could be done with a simple newtype, but looking ahead we will need even more different vocabularies (for factual change and symbolic bisimulations). It would become tedious to write the same instances of Functor, Applicative and Monad each time we add a new vocabulary. Fortunately, `Data.Tagged` already provides us with an instance of Functor for `Tagged t` for any type `t`.

Type Safe BDD manipulation (continued continued continued)

Also note that Dubbel is an empty type, isomorphic to ().

```
data Dubbel
type RelBDD = Tagged Dubbel Bdd

totalRelBdd, emptyRelBdd :: RelBDD
totalRelBdd = pure $ boolBddOf Top
emptyRelBdd = pure $ boolBddOf Bot

allsamebdd :: [Prp] -> RelBDD
allsamebdd ps = pure $ conSet [boolBddOf $ PrpF p `Equi` PrpF p' | (p,p') <- zi

class TagBdd a where
  tagBddEval :: [Prp] -> Tagged a Bdd -> Bool
  tagBddEval truths querybdd = evaluateFun (untag querybdd) (\n -> P n `elem` t

instance TagBdd Dubbel
```