Functional Programming for Logicians — Lecture 5

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

Malvin Gattinger

8 June 2018

```haskell
module L5 where


(!) :: 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)
```

DEL

Muddy Children

An early version of this puzzle are the three ladies on a train:

> *"Three ladies, A, B, C in a railway carriage all have dirty faces and are all laughing. It suddenly flashes on A: why doesn't B realize C is laughing at her? — Heavens! I must be laughable. (Formally: if I, A, am not laughable, B will be arguing: if I, B, am not laughable, C has nothing to laugh at. Since B does not so argue, I, A, must be laughable.)" (Littlewood 1953)*

Isomorphic to this is the story about muddy children:

> *"Imagine n children playing together. The mother of these children has told them that if they get dirty there will be severe consequences. So, of course, each child wants to keep clean, but each would love to see the others get dirty. Now it happens during their play that some of the children, say k of them, get mud on their foreheads. Each can see the mud on others but not on his own forehead. So, of course, no one says a thing. Along comes the father, who says, "At least one of you has mud on your forehead," thus expressing a fact known to each of them before he spoke (if $k > 1$). The father then asks the following question, over and over: "Does any of you know whether you have mud on your own forehead?" Assuming that all the children are perceptive, intelligent, truthful, and that they answer simultaneously, what will happen?" (Fagin et. al 1995)*

Epistemic Logic

Syntax

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

Kripke Models

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

▸ W	set of worlds
▸ $R_i \subseteq W \times W$	indistinguishability relation
▸ $\mathsf{Val}\colon W \to \mathcal{P}(P)$	valuation function

Semantics

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

Dynamic Epistemic Logic: Action Models

Action Models

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

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

Dynamic Epistemic Logic: Action Models

Action Models

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

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

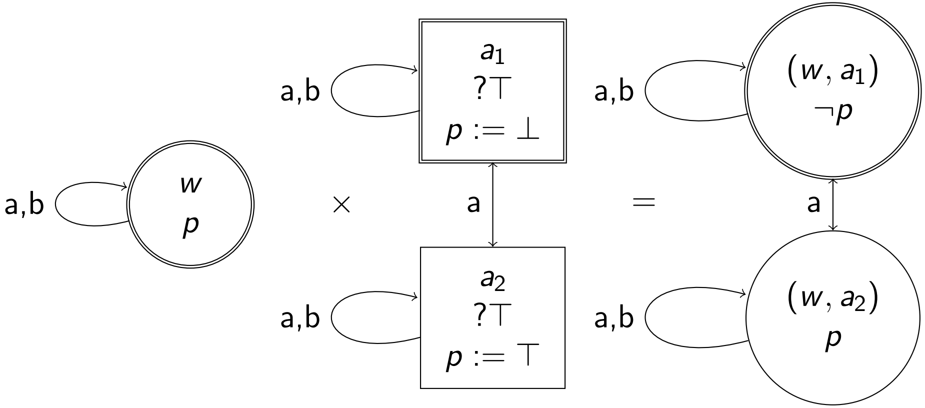
Product Update

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

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

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

DEL Example: Coin Flip hidden from a



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

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 Hardes Logic Puzzle Ever (Knights & Knaves)

But also:

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

Simple Explicit 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

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

```haskell
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

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

```
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.

```
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])]
```

```
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
```

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

```haskell
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
```

Making Announcements

```
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

- The set of possible worlds is explicitly constructed.
- Epistemic (equivalence) relations are spelled out.

$\Rightarrow$ Everything has to fit in memory. For large models (1000 worlds) it gets slow. Runtime in seconds for n Muddy Children, needing 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 for DEL

Symbolic Model Checking: General Idea

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

Symbolic Model Checking: General Idea

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

There exist efficient methods for many temporal logics like LTL and CTL [@Clarke1999] and also epistemic logics [@Su07:MCTLKO].

Today: How to do it for DEL.

Symbolic Model Checking: General Idea

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

There exist efficient methods for many temporal logics like LTL and CTL [@Clarke1999] and also epistemic logics [@Su07:MCTLKO].

Today: How to do it for DEL.

1. Represent $\mathcal{M} = (W, R_i, V)$ symbolically: $\mathcal{F} = (V, \theta, O_i)$.
2. Translate DEL to equivalent boolean formulas.
3. Use BDDs to speed up boolean operations.

Symbolic Model Checking: General Idea

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])]
```

Symbolic Model Checking for DEL

Knowledge Structures

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

V	*Vocabulary*	a set of propositional variables
θ	*State Law*	a boolean formula over V
$O_i \subseteq V$	*Observables*	propositions observable by i

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

Call $(\mathcal{F}, s)$ a scenario.

Symbolic Model Checking for DEL

Knowledge Structures

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

V	*Vocabulary*	a set of propositional variables
θ	*State Law*	a boolean formula over V
$O_i \subseteq V$	*Observables*	propositions observable by i

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

Call $(\mathcal{F}, s)$ a scenario.

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

Ludwig Wittgenstein

New Semantics for DEL on Knowledge Structures

Easy:

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

I know something iff it follows from my observations:

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

Updates restrict the set of states:

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

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

Knowledge Structures

Example

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

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

Some facts:

- $\mathcal{F}, \varnothing \vDash \neg p \land K_1 \neg p \land \neg K_2 \neg p$
- $\mathcal{F}, \{p\} \vDash p \land K_1 p \land \neg K_2 p$
- $\mathcal{F}, \{p\} \vDash [!p] K_2 p$
  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 Scenario   = (KnowStruct,KnState)

eval :: Scenario -> Form -> Bool
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)   =
  if eval scn f then eval scn g else True
eval (kns@(KnS _ _ obs),s) (K i form) =
  all (\s' -> eval (kns,s') form) theres where
    oi       = apply obs i
    theres   = filter sameO (statesOf kns)
    sameO s' = (restrict s' oi) `seteq` (restrict s oi)
```

From Knowledge Structures to Kripke Models

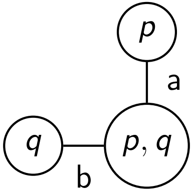
Theorem: For every knowledge structure $\mathcal{F}$ there is an equivalent S5 Kripke Model $\mathcal{M}$ such that $\mathcal{F}, s \vDash \varphi$ iff $\mathcal{M}, w_s \vDash \varphi$.

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

Example: The knowledge structure

$$\mathcal{F} = (V = \{p, q\}, \theta = p \vee q, O_a = \{p\}, O_b = \{q\})$$

is equivalent to this Kripke model:



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 :: KnowScene -> (PointedModelS5, StateMap)
knsToKripkeWithG (kns@(KnS ps _ obs),curs) =
  if curs `elem` statesOf kns
     then ((KrMS5 worlds rel val, cur) , g)
     else error "knsToKripke failed: Invalid state."
  where
    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 = map (\s -> (restrictState s (obs ! i), lav ! s)) (sta
    worlds = map snd lav
    cur    = lav ! curs
```

From Kripke Models to Knowledge Structures

This direction is non-trivial.

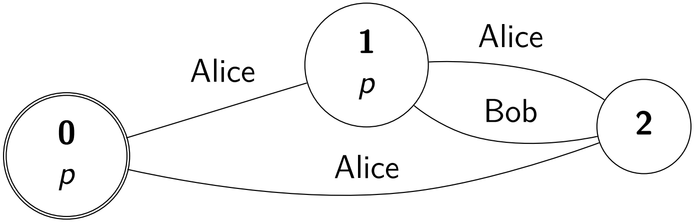
Theorem: For every S5 Kripke Model $\mathcal{M}$ there is an equivalent knowledge structure $\mathcal{F}$ such that $\mathcal{M}, w \vDash \varphi$ iff $\mathcal{F}, s_w \vDash \varphi$.

From Kripke Models to Knowledge Structures

This direction is non-trivial.

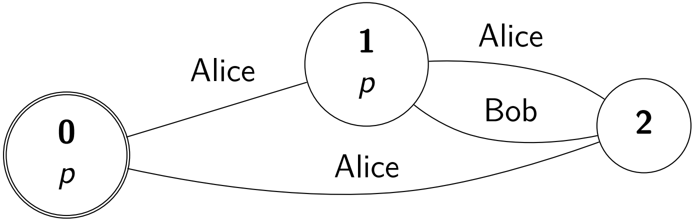
Theorem: For every S5 Kripke Model $\mathcal{M}$ there is an equivalent knowledge structure $\mathcal{F}$ such that $\mathcal{M}, w \vDash \varphi$ iff $\mathcal{F}, s_w \vDash \varphi$.

Proof. Problematic cases look like this:



From Kripke Models to Knowledge Structures

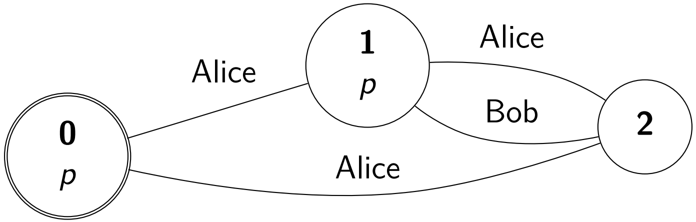
Proof. (continued)



Trick: Add propositions to distinguish all equivalence classes.

From Kripke Models to Knowledge Structures

Proof. (continued)



is equivalent to

$$(\ V = \{p, p_2\},\ \theta = p_2 \rightarrow p,\ O_{\text{Alice}} = \varnothing,\ O_{\text{Bob}} = \{p_2\}\ )$$

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

$\square$

Implementation: Kripke → KNS

```haskell
kripkeToKnsWithG :: PointedModelS5 -> (KnowScene, StateMap)
kripkeToKnsWithG (KrMS5 worlds rel val, cur) = ((KnS ps law obs, curs), g) where
  v         = map fst (val ! cur)
  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,_) -> elem w 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 ]
  curs      = sort (g cur)
```

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

Everything is boolean!

Definition: Fix a knowledge structure $\mathcal{F} = (V, \theta, O_1, \cdots, O_n)$. We translate everything to boolean formulas $\| \cdot \|_{\mathcal{F}}$:

p	p
$\neg\varphi$	$\neg\|\varphi\|_{\mathcal{F}}$
$\varphi_1 \wedge \varphi_2$	$\|\varphi_1\|_{\mathcal{F}} \wedge \|\varphi_2\|_{\mathcal{F}}$
$K_i\varphi$	$\forall(V \setminus O_i)(\theta \rightarrow \|\varphi\|_{\mathcal{F}})$
$[!\varphi]\psi$	$\|\varphi\|_{\mathcal{F}} \rightarrow \|\psi\|_{\mathcal{F}^\varphi}$

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

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

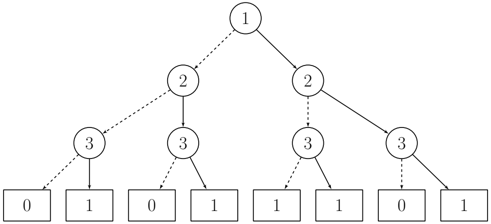
Why care about boolean formulas?

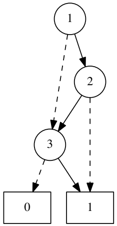
Binary Decision Diagrams

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

How long do you need to compare these two formulas?

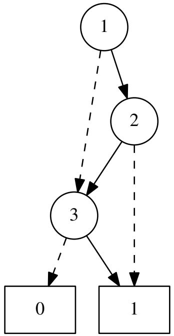
$$p_3 \vee \neg(p_1 \rightarrow p_2) \quad ??? \quad \neg(p_1 \wedge \neg p_2) \rightarrow p_3$$

BDD Magic

How long do you need to compare these two formulas?

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

Here ~~are~~ is their BDD~~s~~:



BDD Magic

This was not an accident, BDDs are canonical.

Theorem:

$$\varphi \equiv \psi \quad \Rightarrow \quad \mathsf{BDD}(\varphi) = \mathsf{BDD}(\psi)$$

Equivalence checks are free and we have fast algorithms to compute BDD($\neg\varphi$), BDD($\varphi \wedge \psi$), BDD($\varphi \to \psi$) etc.

NooBDD: A very naive BDD Implementation

See https://github.com/m4lvin/NooBDD.

```haskell
data Bdd = Top | Bot | Node Int Bdd Bdd
```

(Has)CacBDD

If you worry about speed then use C++, they say. Hence to speed up boolean operations, we use *CacBDD* [@Su13:CacBDD] via binding, see https://github.com/m4lvin/HasCacBDD.

Alternatively, SMCDEL can also use CUDD.

Implementation: Translation to BDDs

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

bddOf :: KnowStruct -> Form -> Bdd
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 (Disj forms) = disSet $ 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 \\ apply obs i
bddOf kns (PubAnnounce form1 form2) =
  imp (bddOf kns form1) newform2 where
    newform2 = bddOf (pubAnnounce kns form1) form2
```

Putting it all together

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

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

```haskell
evalViaBdd :: Scenario -> Form -> Bool
evalViaBdd (kns@(KnS allprops _ _),s) f = bool where
  b      = restrictSet (bddOf kns f) facts
  facts = [ (n, P n `elem` s) | (P n) <- allprops ]
  bool  | b == top   = True
        | b == bot   = False
        | otherwise = error ("BDD leftover.")
```

Examples and Results

Symbolic Muddy Children

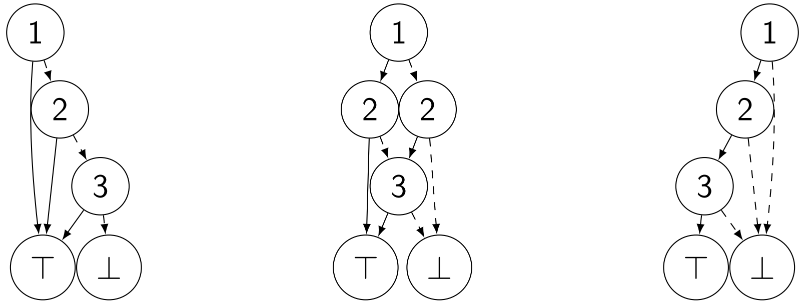
Initial knowledge structure:

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

After the third announcement the children know their own state:

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

Intermediate BDDs for the state law in muddy children:



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

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.

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

Bob: “Crow has card 7.”

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

Bob: “Crow has card 7.”

There are 102 such “safe announcements” which [@vDitm2003RC] 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 ]
```

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 Fenrong, Yanjing and Jan 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 Tsinghua University. 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 Fenrong, Yanjing and Jan 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 Tsinghua University. 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?

Howto use SMCDEL

The easy way: SMCDEL web

Link: https://w4eg.de/malvin/illc/smcdelweb

Input: A knowledge structure and formulas to be checked.

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

The hard way: `import SMCDEL`

This allows you to define abbreviations and generat larger models automatically without writing them by hand.

Transformers

Action Models and Product Update

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

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

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

- $W' = \{(w, a) \in W \times A \mid \mathcal{M}, w \vDash \text{pre}(a)\}$
- $R'_i(s, a)(t, b)$ 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 $\mathcal{M}, w \vDash \text{pre}(a)$ 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

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

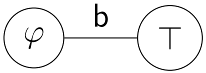
Event: $(\mathcal{X}, x)$ where $x \subseteq V^+$

Knowledge Transformers

Examples:

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

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



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.

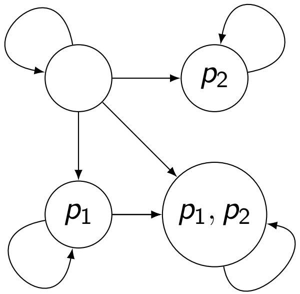
We assume $K\varphi \to \varphi$ but in general not $B\varphi \to \varphi$.

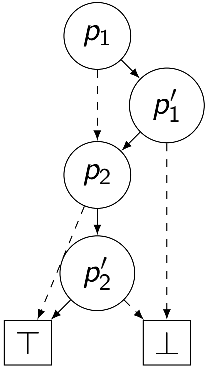
$\Rightarrow$ Kripke Models for Belief are not reflexive.

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.

[@GoroRyan02:BelRevBDD]





Non-S5 Knowledge Structures

For every agent we replace O_i with a BDD Ω_i.

Now translate $\Box_i \varphi$ to:

$$\forall \vec{p'} (\theta' \rightarrow (\Omega_i(\vec{p}, \vec{p'}) \rightarrow (\|\varphi\|_{\mathcal{F}})'))$$

Type Safe BDD manipulation

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

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.

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.

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`.

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' |

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

instance TagBdd Dubbel
```

. . .

Project Practicalities

Topic Choice

See https://malv.in/2018/funcproglog/topics.html for a list of topics.

You can also come up with your own.

Please *send us an email by Sunday evening* what you would like to work on.

Next Friday, you will give a short presentation on what you are doing. You may prepare a simple `.lhs` file, something to write on the board or up to three slides for this.

Report Grading Criteria

You will only get a pass/fail grade.

Additionally we will of course give feedback in text.

Your final report should:

- have a clear topic, and a concrete goal or research question
- be written in literate programming style and well-structured
- compile
- have zero warnings with `ghc -Wall`
- generate zero hints from `hlint`
- be between 5 and 25 pages (i.e. the length does not really matter)