The Symbolic Approach to Dynamic Epistemic Model Checking

Malvin Gattinger

2015-11-06, Tsinghua University
`https://w4eg.de/malvin/tsinghua`

Outline

Recap: Explicit DEL Model Checking

Explicit DEL Model Checking: Implementation

```
data EpistM state = Mo
              [state]
              [Agent]
              [(state,[Prp])]
              [(Agent,Erel state)]
              [state]  deriving (Eq,Show)

isTrueAt :: EpistM state -> state -> Form state -> Bool
isTrueAt _ _ Top = True
isTrueAt _ w (Info x) = w == x
isTrueAt(Mo _ _ val _ _) w (Prp p) = elem p (apply val w)
isTrueAt m w (Ng f)    = not (isTrueAt m w f)
isTrueAt m w (Conj fs) = all (isTrueAt m w) fs
isTrueAt m w (Disj fs) = any (isTrueAt m w) fs
isTrueAt m w (Kn ag f) =
  all (\v -> isTrueAt m v f) (bl (rel ag m) w)
```

Explicit DEL Model Checking: Example

```
initM 3 = Mo
  [[True,True,True],[True,True,False],[True,False,True]
  ,[True,False,False],[False,True,True],[False,True,False]
  ,[False,False,True],[False,False,False]]
  [Ag 1,Ag 2,Ag 3]
  []
  [(Ag 1,[[[True,True,True],[False,True,True]]
         ,[[True,True,False],[False,True,False]]
         ,[[True,False,True],[False,False,True]]
         ,[[True,False,False],[False,False,False]]])
  ,(Ag 2,...),(Ag 3,...)]
  [[False,True,True]]
```

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:

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 (Clarke, Grumberg, and Peled 1999) and also epistemic logics (Su, Sattar, and Luo 2007).

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 (Clarke, Grumberg, and Peled 1999) and also epistemic logics (Su, Sattar, and Luo 2007).
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 $(\mathcal{F}, s) \vDash \varphi$
- $(\mathcal{F}, s) \vDash \varphi \wedge \psi$ 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', 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

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

Knowledge Structures

Example

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

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

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

Implementation of Knowledge Structures and Semantics

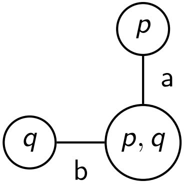
```haskell
data KnowStruct = KnS [Prp] Bdd [(Agent,[Prp])]
type KnState    = [Prp]
type Scenario   = (KnowStruct,KnState)

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 $W := \{s \subseteq V \mid s \vDash \theta\}$, $V = \mathrm{id}$ 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 $\rightarrow$ Kripke

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

```
knsToKripke :: Scenario -> PointedModel
knsToKripke (kns@(KnS ps _ obs),c) = (KrM ws rel val, w) wh
  lav = zip
    (statesOf kns)
    [0..(length (statesOf kns)-1)]
  val = map ( \(s,n) -> (n,state2kripkeass s) ) lav
  state2kripkeass s = map (\p -> (p, p `elem` s)) ps
  rel = [(i,rf i) | i <- map fst obs]
  rf i = map
    (map snd)
    (groupBy (\(x,_) (y,_) -> x==y) (sort $ pairs i))
  pairs i = map
    (\s -> (restrict s (apply obs i), apply lav s))
    (statesOf kns)
  ws = map fst val
  w  = apply lav c
```

From Kripke Models to Knowledge Structures

This direction is non-trivial.

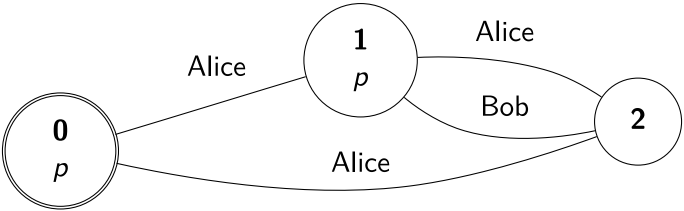
Theorem: For every S5 Kripke Model $\mathcal{M}$ there is an equivalent knowledge structure $\mathcal{F}$ such that $\mathcal{M}, w \vDash \varphi$ iff $\mathcal{F}, s_w \vDash \varphi$.

From Kripke Models to Knowledge Structures

This direction is non-trivial.

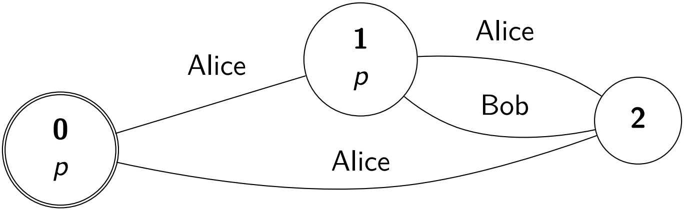
Theorem: For every S5 Kripke Model $\mathcal{M}$ there is an equivalent knowledge structure $\mathcal{F}$ such that $\mathcal{M}, w \vDash \varphi$ iff $\mathcal{F}, s_w \vDash \varphi$.

Proof. Problematic cases look like this:



From Kripke Models to Knowledge Structures

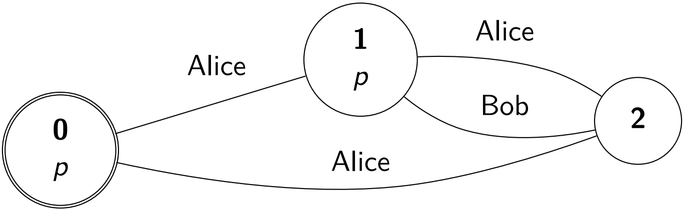
Proof. (continued)



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: $\{p, p_2\}$

□

Implementation: Kripke → KNS

```
kripkeToKns :: PointedModel -> Scenario
kripkeToKns (KrM worlds rel val, cur) = (KnS ps law obs, curs) where
  v          = map fst $ apply val cur
  ags        = map fst rel
  newpstart = fromEnum $ freshp v -- start counting new propositions
  amount i   =
    ceiling (logBase 2 (fromIntegral $ length (apply rel i)) -- |O_i|
  newpstep   = maximum [ amount i | i <- ags ]
  numberof i = fromJust $ elemIndex i (map fst rel)
  newps i    = map -- O_i
    (\k -> P (newpstart + (newpstep * numberof i) +k))
    [0..(amount i - 1)]
  copyrel i = zip -- label equiv.classes with P(O_i)
    (apply rel i)
    (powerset (newps i))
  gag i w    = snd $ head $ filter (\(ws,_) -> elem w ws) (copyrel i)
  g w        = filter (apply (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
$\neg\varphi$	$\neg\|\varphi\|_{\mathcal{F}}$
$\varphi_1 \wedge \varphi_2$	$\|\varphi_1\|_{\mathcal{F}} \wedge \|\varphi_2\|_{\mathcal{F}}$
$K_i\varphi$	$\forall(V \setminus O_i)(\theta \to \|\varphi\|_{\mathcal{F}})$
$[!\varphi]\psi$	$\|\varphi\|_{\mathcal{F}} \to \|\psi\|_{\mathcal{F}^\varphi}$

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

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

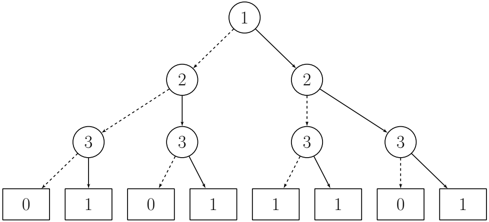
Why care about boolean formulas?

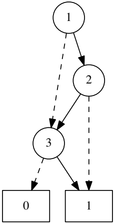
Binary Decision Diagrams

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?

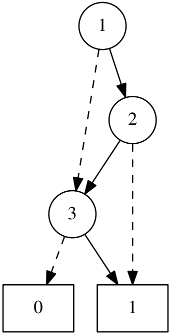
$$p_3 \vee \neg(p_1 \rightarrow p_2) \quad ??? \quad \neg(p_1 \wedge \neg p_2) \rightarrow p_3$$

BDD Magic

How long do you need to compare these two formulas?

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

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 $\mathsf{BDD}(\neg\varphi)$, $\mathsf{BDD}(\varphi \wedge \psi)$, $\mathsf{BDD}(\varphi \to \psi)$ etc.

NooBDD: A very naive BDD Implementation

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

```
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* (Lv, Su, and Xu 2013) via binding, see `https://github.com/m4lvin/HasCacBDD`.

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 modelcheck $\mathcal{F}, s \vDash \varphi$

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

```
evalViaBdd :: 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

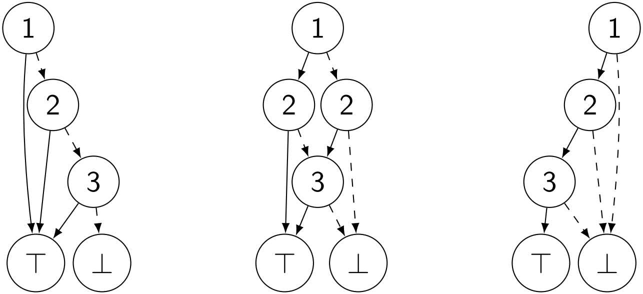
Initial knowledge structure:

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

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:



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.*

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."

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."

There are 102 such "safe announcements" which (Ditmarsch 2003) had to find and check by hand.
With symbolic model checking this takes 4 seconds.

Sum and Product

The puzzle from (Freudenthal 1969):

> *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

The puzzle from (Freudenthal 1969):

> *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).*

Solved in 2 seconds.

Sum and Product: Encoding numbers

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

-- 7 propositions are enough to label [2..100]
xProps, yProps, sProps, pProps :: [Prp]
xProps = [(P  1)..(P  7)]
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
```

Dining Cryptographers

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 is solved by (Chaum 1988).

Dining Cryptographers

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 is solved by (Chaum 1988).
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? (Benthem et al. 2009, Ditmarsch, Hoek, and Ruan (2013))
- Which formalizations are more intuitive?
- What is faster
  - for your computer to model check?
  - for you to write down formulas?

Howto use SMCDEL yourself

The easy way: SMCDEL web

Link: `https://w4eg.de/malvin/illc/smcdelweb`
Input: A knowledge structure and formulas to be checked.

```
-- Three Muddy Children in SMCDEL
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
  [ ! ((~ (alice knows whether 1)) & (~ (bob knows whether
  (1 & 2 & 3)
```

The hard way: `import SMCDEL`

Install HasCacBDD, then SMCDEL.
This allows you to define abbreviations and generat larger models automatically without writing them by hand.

Action Models and Transformers

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:

$$\mathcal{X} = (V^+, \mu, O_1^+, \dots, O_n^+)$$

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

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

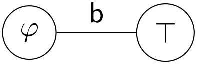
Event: $(\mathcal{X}, x)$ where $x \subseteq V^+$

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 $\mathcal{F}$:

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

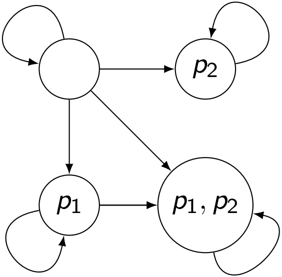
Non-S5

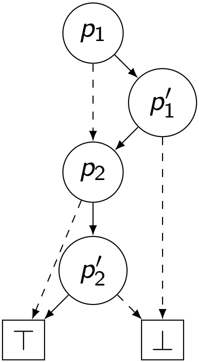
Belief as KD45

A crucial difference between Knowledge and Belief is Truth.
We assume $K\varphi \rightarrow \varphi$ but in general not $B\varphi \rightarrow \varphi$.
$\Rightarrow$ 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.
(Gorogiannis and Ryan 2002)





Non-S5 Knowledge Structures

For every agent we replace O_i with a BDD Ω_i.
Now translate $K_i\varphi$ to $\forall \vec{p'}(\theta' \rightarrow (\Omega_i(\vec{p}, \vec{p'}) \rightarrow (\|\varphi\|_{\mathcal{F}})'))$

Thank you!

References

Benthem, Johan van, Jelle Gerbrandy, Tomohiro Hoshi, and Eric Pacuit. 2009. "Merging Frameworks for Interaction." *Journal of Philosophical Logic* 38 (5). Springer: 491–526. doi:http://doi.org/10.1007/s10992-008-9099-x.

Bryant, Randal E. 1986. "Graph-Based Algorithms for Boolean Function Manipulation." *IEEE Transaction on Computers* C-35 (8): 677–91. doi:http://doi.org/10.1109/TC.1986.1676819.

Chaum, David. 1988. "The dining cryptographers problem: Unconditional sender and recipient untraceability." *Journal of Cryptology* 1 (1). Springer-Verlag: 65–75. doi:http://doi.org/10.1007/BF00206326.

Clarke, E. M., O. Grumberg, and D. A. Peled. 1999. *Model Checking*. Cambridge, Massachusetts, USA: The MIT Press.

Ditmarsch, Hans van. 2003. "The Russian Cards Problem." *Studia Logica* 75 (1). Springer: 31–62.

Ditmarsch, Hans van, Wiebe van der Hoek, and Ji Ruan. 2013. "Connecting Dynamic Epistemic and Temporal Epistemic Logics." *Logic Journal of IGPL* 21 (3). Oxford Univ Press: 380–403. doi:http://doi.org/10.1093/jigpal/jzr038.

Freudenthal, Hans. 1969. "Formulering van Het 'Som-En-Product'-Probleem." *Nieuw Archief Voor Wiskunde* 17: 152.

Gorogiannis, Nikos, and Mark D. Ryan. 2002. "Implementation of Belief Change Operators Using BDDs." *Studia Logica* 70 (1). Kluwer Academic Publishers: 131–56. doi:http://doi.org/10.1023/A:1014610426691.

Lv, Guanfeng, Kaile Su, and Yanyan Xu. 2013. "CacBDD: A BDD Package with Dynamic Cache Management." In *Proceedings of the 25th International Conference on Computer Aided Verification*, 229–34. CAV'13. Berlin, Heidelberg: Springer-Verlag. doi:http://doi.org/10.1007/978-3-642-39799-8_15.

Su, Kaile, Abdul Sattar, and Xiangyu Luo. 2007. "Model Checking Temporal Logics of Knowledge Via OBDDs." *The Computer Journal* 50 (4): 403–20. doi:http://doi.org/10.1093/comjnl/bxm009.