Epistemic Model Checking with Haskell

Malvin Gattinger

2016-12-01, Peking University

Haskell in 10 Minutes

Simple Explicit Model Checking

Symbolic Model Checking

Binary Decision Diagrams

More Puzzles

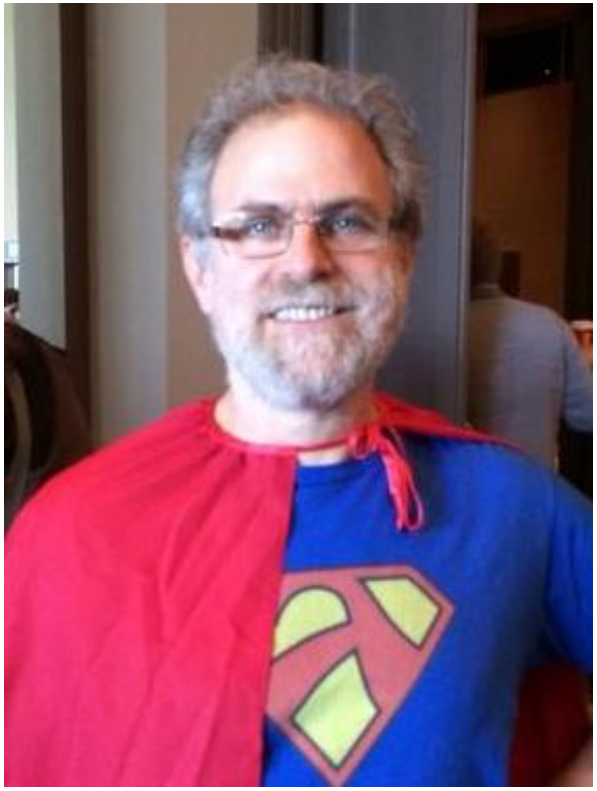
Even More

Haskell in 10 Minutes

Why Haskell?

"Most of you use languages that were invented, and you can tell, can't you. This is my invitation to you to use programming languages that are discovered."

Philip Wadler: Propositions as Types



Functional vs. Imperative

Imperative Programming (C++, Java, etc.)

- instructions, telling the computer *what to do*
- mutable *state*
- functions are subroutines

```
function add(x,y) {
  z = x + y;
  return z;
}
c = 5;
c = add (10,c);
print c;
```

Result: 15

Functional vs. Imperative II

Functional Programming (Haskell, OCaml, Lisp, Closure, . . .):

- definitions, telling the computer *what to calculate*
- nothing is mutable
- everything is a function with specific arguments and results

```
add (x,y) = x + y
c = 5
newc = add (10,c)
```

Haskell is Statically Typed

Read `::` as “is a” or “has the type”:

```haskell
c :: Int
c = 5

greeting :: String
greeting = "Hello"

add :: (Int,Int) -> Int
add (x,y) = x + y

add' :: Int -> Int -> Int
add' x y = x + y
```

Lists and Patterns

We can use lists of things of the same type.

```
somenumbers :: [Int]
somenumbers = [2,3,4,2,37]

myfunction :: Int -> Int
myfunction x = x + 5
```

```
λ> map myfunction somenumbers
[7,8,9,7,42]
```

List Pattern Matching and List Comprehension

A list can be empty `[]` or contain a first element `x:xs`.

```
addDouble :: [Int] -> Int
addDouble []     = 0
addDouble (x:xs) = 2 * x + addList xs

addDouble' :: [Int] -> Int
addDouble' l = sum [ 2 * x | x <- l ]
```

Compare the last line to set theory: $\{2 * x \mid x \in l\}$

Creating Types

```
data Animal = Cat | Dog

greet :: Animal -> String
greet Cat = "Meeow!"
greet Dog = "Woof!"

type Zoo = [Animal]

greetAll :: Zoo -> String
greetAll z = concatMap greet z

GHCi> greetAll [Cat,Dog,Cat]
"Meeow!Woof!Meeow!"
```

Simple Explicit Model Checking

Agents, Formulas

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

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

type Prop = Int

type Ag = String
```

Abbreviations like $\varphi \vee \psi := \neg(\neg\varphi \wedge \neg\psi)$:

```
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)
```

Semantics

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

```
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

```
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])]
```

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

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

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

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

More Features

- model update: `announce :: Model -> Form -> Model`
- generate large models: `muddyFor :: Int -> Model`
- draw models automatically

(show examples)

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

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:

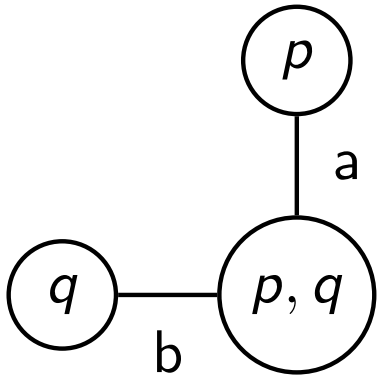
```
KnS [P 1,P 2]
    (boolBddOf Top)
    [ ("Alice",[P 1])
    , ("Bob"  ,[P 2])]
```

Symbolic Model Checking Epistemic Logic

Example: The knowledge structure

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

is equivalent to this Kripke model:



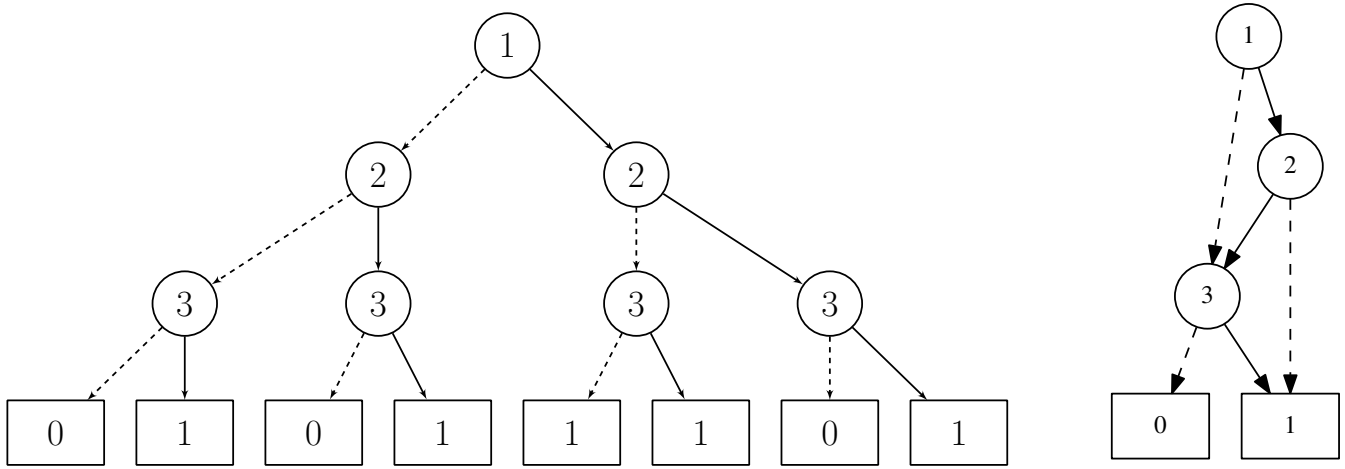
Motto: Describe instead of list! Use boolean operations!

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.



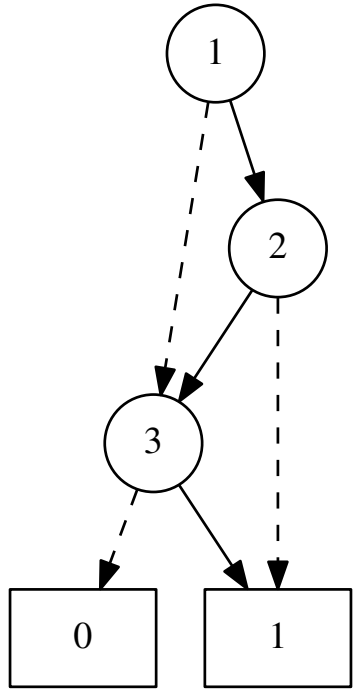
[Read the classic (Bryant 1986) for more details.]

BDD Magic

How long do you need to compare these two formulas?

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

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



BDD Magic

This was not an accident, BDDs are canonical.

Theorem:

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

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

(Has)CacBDD

To speed up boolean operations, we use *CacBDD* via binding, see `https://github.com/m4lvin/HasCacBDD`.

Implementation: Translation to BDDs

```
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 $\mathcal{F}, s \models \varphi$

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.")
```

Symbolic Muddy Children

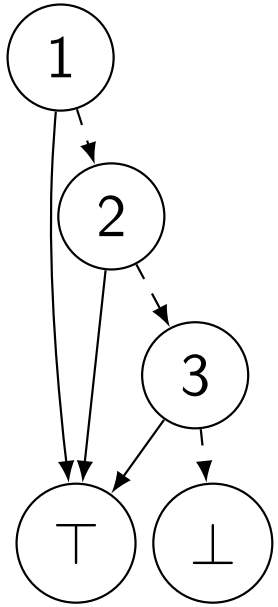
Initial knowledge structure:

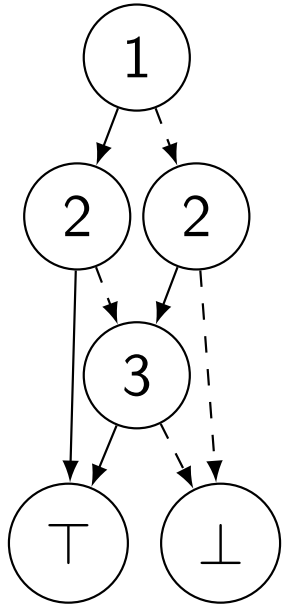
$$\mathcal{F} = (\{p_1, p_2, p_3\}, \top, O_1 = \{p_2, p_3\}, O_2 = \{p_1, p_3\}, O_3 = \{p_1, p_2\})$$

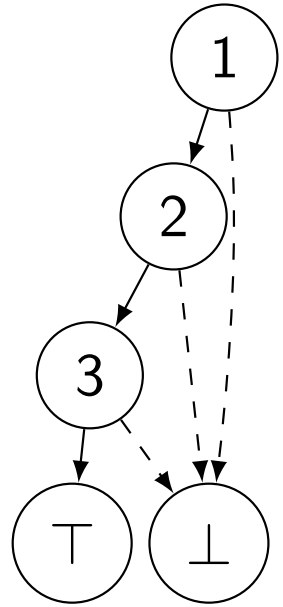
After the third announcement the children know their own state:

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

Intermediate BDDs for the state law:







Muddy Children as a Benchmark

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

More Puzzles

Russian Cards

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 (van Ditmarch 2003) found and checked by hand. With symbolic model checking we can finde them in 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).

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 one of them or the University paid. However, if one of them paid, they also respect the wish to stay anonymous. That is, they do not want to know who of them paid if it was one of them.

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

Even More

Further Topics

Haskell:

- Type variables
- Typeclasses and Polymorphism
- Monads

Epistemic Model Checking:

- S5 vs. Non-S5
- Computational Complexity
- Translations between frameworks:
  - Kripke Model $\leftrightarrow$ Knowledge Structure
  - DEL $\leftrightarrow$ ETL

References

Jan van Eijck, Kees Doets: *The Haskell Road to Logic, Maths and Programming*, 2004. `http://homepages.cwi.nl/~jve/HR`

Miran Lipovača: *Learn You a Haskell*, 2011.
`http://learnyouahaskell.com`

Philip Wadler: *Propositions as Types*, 2015.
`https://youtu.be/IOiZatlZtGU`

Johan van Benthem, Jan van Eijck, Malvin Gattinger, Kaile Su: *Symbolic Model Checking for Dynamic Epistemic Logic – S5 and Beyond*, Journal of Logic and Computation (JLC), to appear.
`https://is.gd/77Th6u`

Malvin Gattinger: *SMCDEL*, last update May 2016.
`https://github.com/jrclogic/smcdel`
Try it online: `https://w4eg.de/malvin/illc/smcdelweb`

Thank You!

https://w4eg.de/malvin

malvin@w4eg.de