NorMC: a Norm Compliance Temporal Logic Model Checker

Piotr Kaźmierczak, Truls Pedersen, Thomas Ågotnes
Bergen University College and University of Bergen
Norway

PAMT symposium, May 2014, Bergen

Outline

- Why model checking norms?
- Why Haskell?
- How is the model checker implemented?
- Proving that two people can pass each other on the street
- Live Demo

Why model checking norms?

- Normative systems are a framework for coordinating multi-agent systems that emerged recently in the literature.
- The starting point is a state-transition model of a multi-agent system, and the goal is to constrain the behavior of the agents in such a way that the global behavior of the system exhibits some desirable properties.
- Such a restriction is called a normative system. The desirable properties are typically represented using a (modal) logical formula; the objective (typically not satisfied in the initial system).

(Linear) Temporal Logic

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathsf{X}\varphi \mid \mathsf{F}\varphi \mid \mathsf{G}\varphi$$

Branching Time Temporal Logic

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathsf{AX}\varphi \mid \mathsf{AF}\varphi \mid \mathsf{AG}\varphi \mid$$
$$\mathsf{EX}\varphi \mid \mathsf{EF}\varphi \mid \mathsf{EG}\varphi \mid \mathsf{A}[\varphi\,\mathcal{U}\,\psi] \mid \mathsf{E}[\varphi\,\mathcal{U}\,\psi]$$

Norm Compliance CTL

It's like temporal logic, only different.

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathsf{AX}\varphi \mid \mathsf{AF}\varphi \mid \mathsf{AG}\varphi \mid$$
$$\mathsf{EX}\varphi \mid \mathsf{EF}\varphi \mid \mathsf{EG}\varphi \mid \mathsf{A}[\varphi\mathcal{U}\psi] \mid \mathsf{E}[\varphi\mathcal{U}\psi]$$

Norm Compliance CTL

It's like temporal logic, only different.

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathsf{AX}\varphi \mid \mathsf{AF}\varphi \mid \mathsf{AG}\varphi \mid \mathsf{EX}\varphi \mid \mathsf{EF}\varphi \mid \mathsf{EG}\varphi \mid \mathsf{A}[\varphi\mathcal{U}\psi] \mid \mathsf{E}[\varphi\mathcal{U}\psi] \mid \langle P\rangle\varphi$$

Norm Compliance CTL

- NCCTL extends the branching-time temporal logic *Computation-Tree Logic* (CTL) with a unary modality $\langle P \rangle$, where P is a *coalition predicate*, i.e., a possible coalition of groups of agents (*coalitions*). The meaning of the expression $\langle P \rangle$ is that if any coalition that satisfies P complies with the normative system, then ϕ will hold

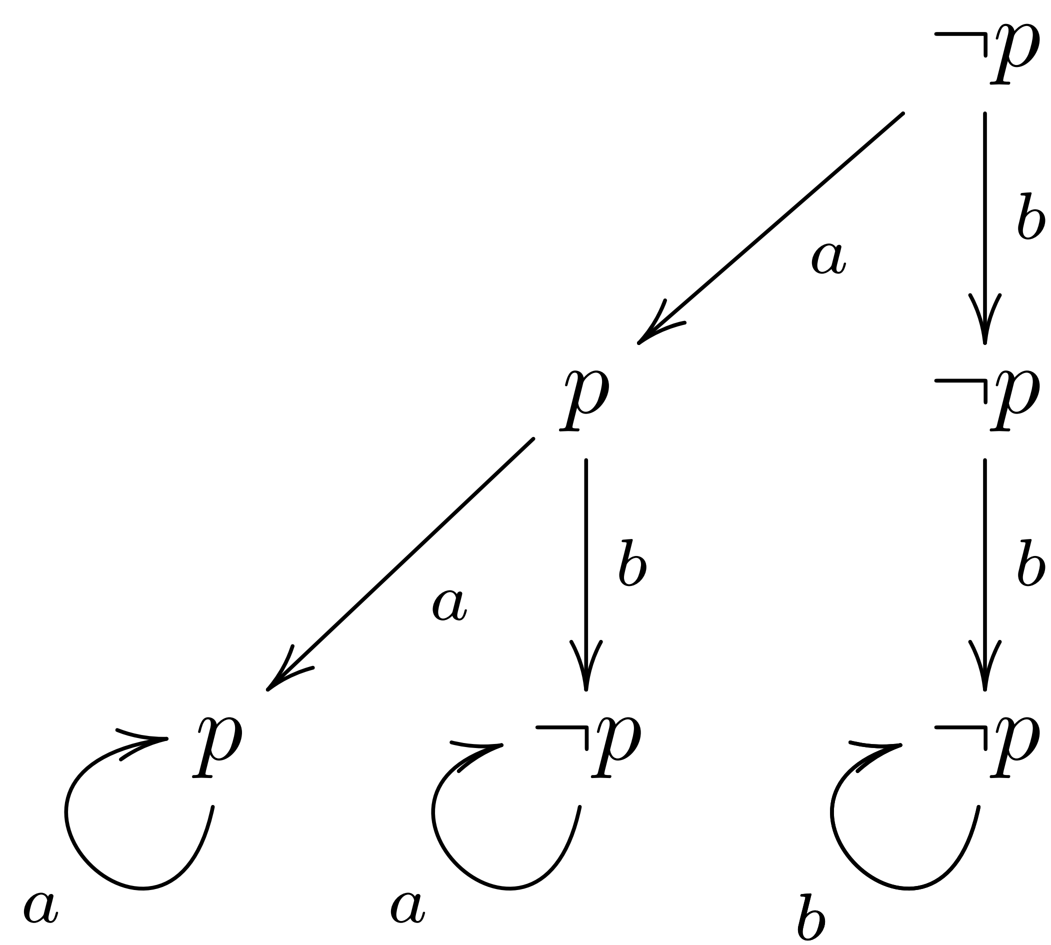
Norm Compliance CTL

Examples of NCCTL expressions include the following, which are evaluated in the context of a model and a normative system:

- $[supseteq(C)]\ \phi$: if any superset of C complies, ϕ will hold (C is *sufficient*);
- $[\neg geq(k)]\neg\phi$: at least k agents have to comply for ϕ to hold (the normative system is k-*necessary*)

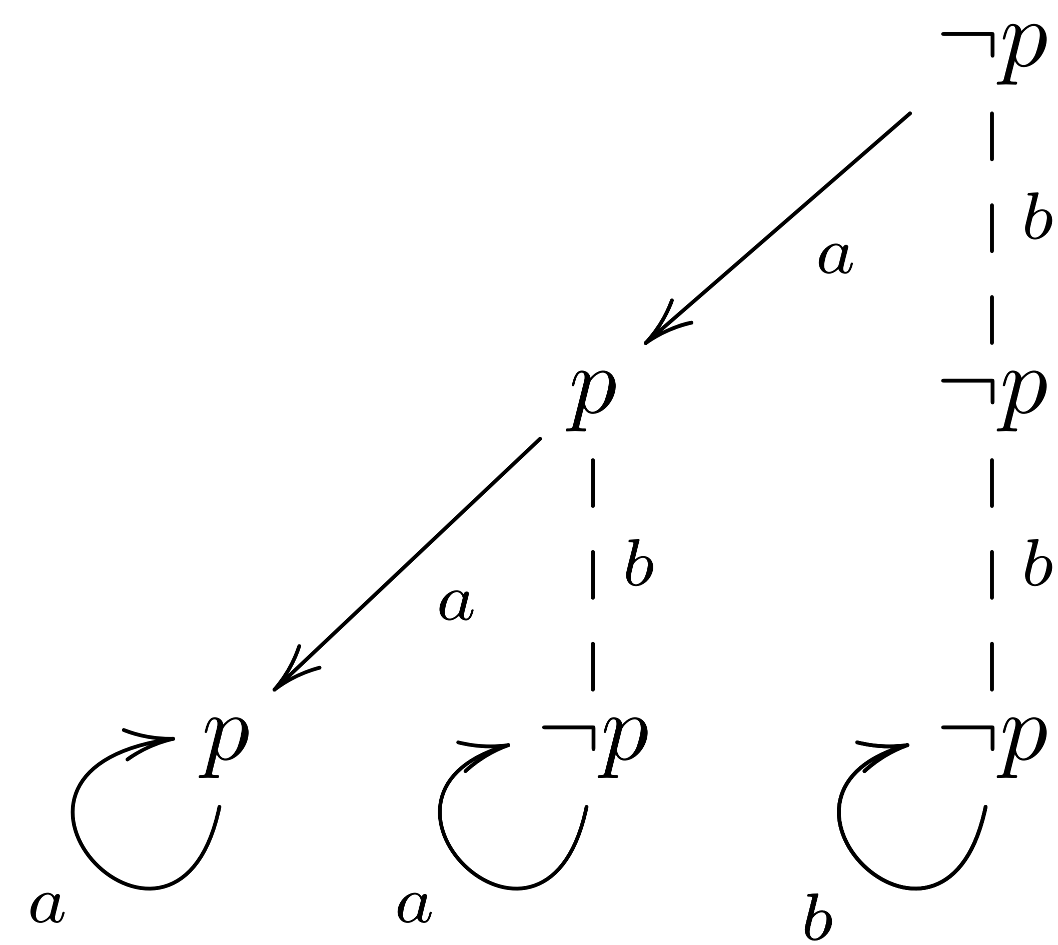
Implementing normative systems

An example agent-labeled Kripke structure, where $K \not\models \mathsf{A} \bigcirc p$



Implementing normative systems

An example agent-labeled Kripke structure, where $K\dagger(\eta \upharpoonright \{b\}) \models \mathsf{A} \bigcirc p$



Why do we need a new model checker?

- Adapting existing CTL model checkers for NCCTL is difficult;
- Verifying properties is difficult even for relatively small models.

Why Haskell?

- Haskell's syntax is close to mathematics;
- Haskell is especially convenient for handling discrete structures;
- We have the full power of the programming language to describe our models

Why Haskell?

- Because Haskell is free, cross-platform, well-documented and actively developed generic programming language.
- Because it is easy to extend the code of the model checker and adapt it to one's needs.

Implementation details

```haskell
data (Ord s, Eq p) => Kripke p s = Kripke {
  agents :: [Int],
  states :: [s],
  tr :: FODBR s s,
  owner :: (s, s) -> Int,
  valuation :: p -> [s]
  }
```

$$K = \langle S, s^0, R, A, \alpha, V \rangle$$

A data structure that represents agent-labeled Kripke models.

Implementation details

```
check :: (Ord s, Eq p) => (Kripke p s) -> (FODBR s s) -> (Formula p) -> [s]
check mod sys (Prop p)    = sort $ (valuation mod) p
check mod sys (Neg f)     = (states mod) `nubminus` (check mod sys f)
check mod sys (Disj f f') = (check mod sys f) `nubunion` (check mod sys f')
check mod sys (Conj f f') = (check mod sys f) `nubisect` (check mod sys f')
```

Model checking. Recursive definition of Sat.

Implementation details

```
check mod sys (EX f)      = find (backwards mod) (check mod sys f)
check mod sys (EF f)      = fix ff (check mod sys (EX f)) where
    ff ss = ss `nubunion` (find $ backwards mod) ss
check mod sys (EG f)      = fix ff (check mod sys f) where
    ff ss = ss `nubisect` (find $ backwards mod) ss
check mod sys (EU f f')   = fix ff (check mod sys f') where
    ff ss = ss `nubunion` ((find $ backwards mod) ss `nubisect`
                                                   (check mod sys f))
```

CTL model checking implementation.

Implementation details

$$K, \eta, s \models \langle P \rangle \varphi \text{ iff } \exists C \subseteq A\ (C \models_{cp} P \text{ and } K \dagger (\eta \upharpoonright C), \eta, s \models \varphi)$$

$$sat(\langle P \rangle \phi) = \bigcup_{C \models_{cp} P} \{s \in S : K \dagger (\eta \upharpoonright C), \eta, s \models \phi\}$$

```
check mod sys (CD c f) = foldl' nubunion [] $
                         map (\mod -> (check mod sys f)) $
                             map (ir mod sys) $ coasGivenCP mod c
```

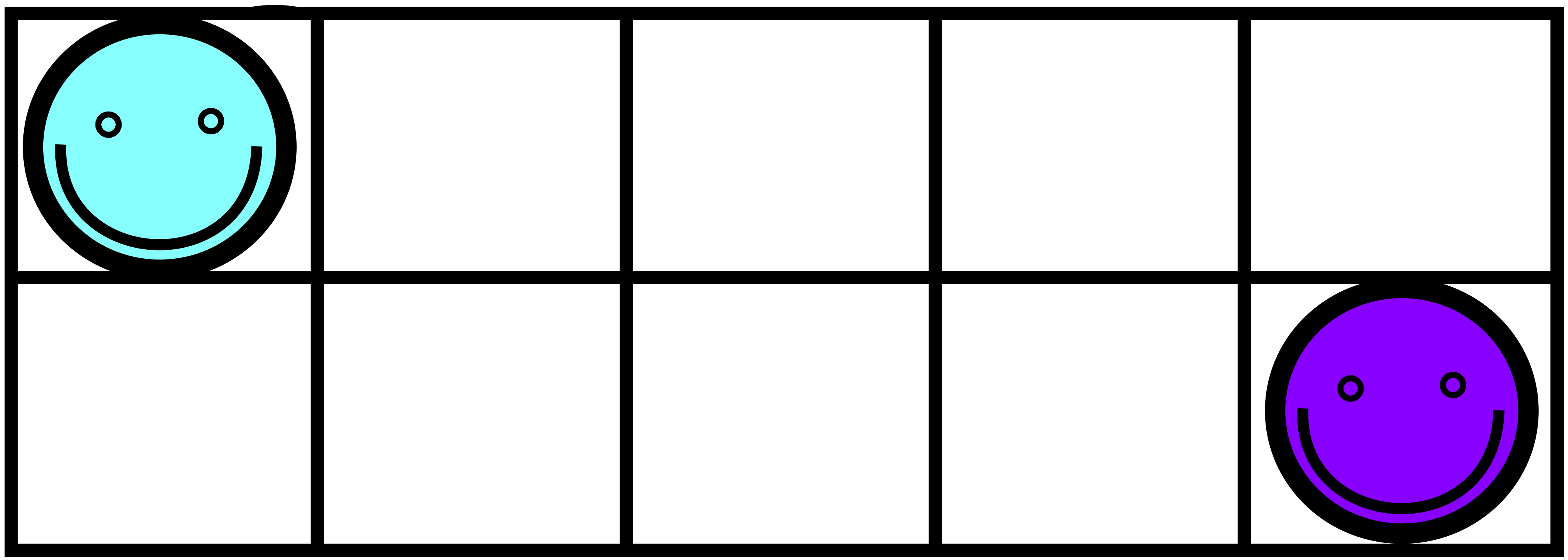
Model checking the ‘diamond’ coalition predicate.

Is NorMC actually useful?

- YES.
- The code is simple, yet able to handle relatively big models (tested for 62500 states/470596 transitions).
- We managed to find a bug in an example in a published paper.

A simple example

A simple example



A simple example

0	2	4	6	8
1	3	5	7	9

ManualSimpleExample.lhs

```
exampleModel :: Kripke Prop State
exampleModel = Kripke [0,1] statespace transition owner val
```

where:

```
data Property = N | S | E | W | T deriving Eq

type Prop  = (Property, Int)
type State = (Int, Int, Int)
```

ManualSimpleExample.lhs

```haskell
statespace :: [State]
statespace = [ (p0, p1, i) | p0 <- [0..9], p1 <- [0..9], p0 /= p1,
                                                           i <- [0,1] ]
```

$$statespace = \{(p_0, p_1, i) \mid p_0 \in \{0, \ldots, 9\} \wedge p_1 \in \{0, \ldots, 9\} \wedge p_0 \neq p_1 \wedge i \in \{0, 1\}\}$$

Definition of the state space.

ManualSimpleExample.lhs

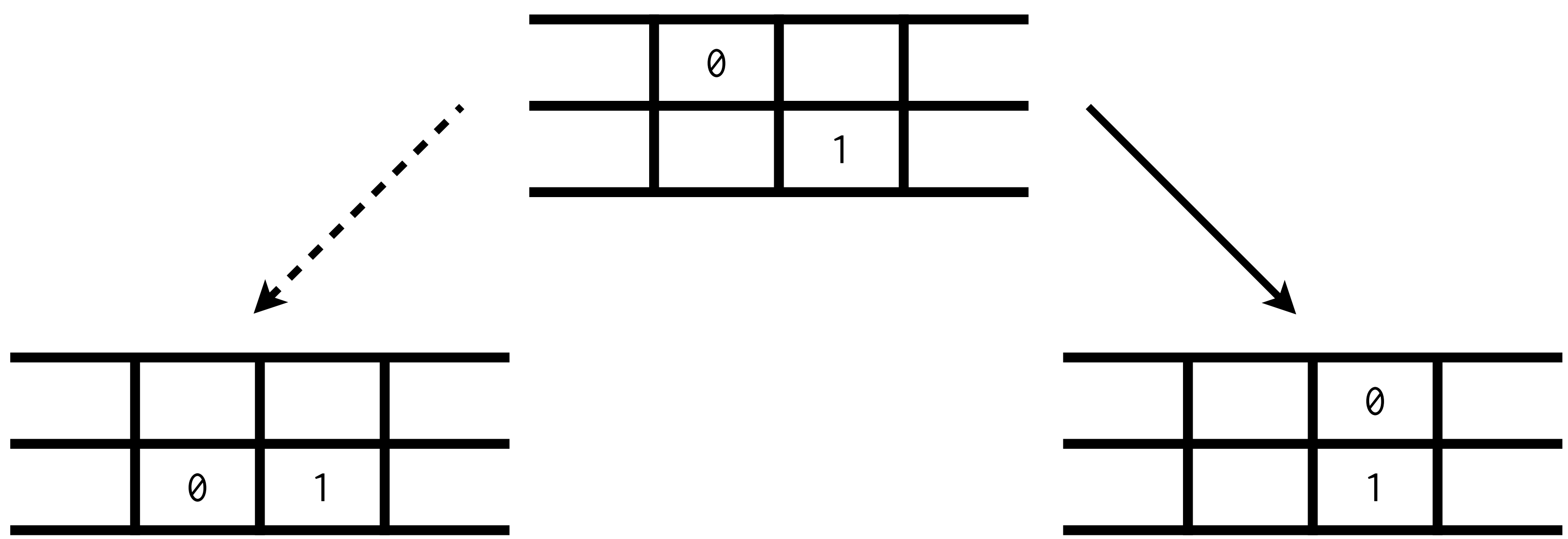
```
transition = build [((p0,p1,i), (p0',p1',1-i)) |
                    (p0,p1,i) <- statespace,
                    p0'       <- possibleSteps (i == 0) p0,
                    p1'       <- possibleSteps (i == 1) p1,
                    (p0',p1',1-i) `elem` statespace
                   ]

possibleSteps :: Bool -> Int -> [Int]
possibleSteps False p = [p]
possibleSteps True p = [p-2, p, p+2]++(if even p then [p+1] else [p-1])
```

Transition relation implementation.

ManualSimpleExample.lhs

```
illegal :: FODBR State State
illegal = build [ ((p0,p1,i),(p0',p1',1-i)) |
                  (p0,p1,i)   <- statespace,
                  (p0',p1',_) <- (find1 (fst transition) (p0,p1,i)),
                  p1' == p0' + 2 || if i == 0 then (p0' < 8 && p0 >= p0')
                                                else (p1' > 1 && p1 <= p1')]
```



Definition of the normative system.

Live demo

```
➜  NorMC git:(master) ghci ManualSimpleExample.lhs
GHCi, version 7.4.1: http://www.haskell.org/ghc/  :? for help
Loading package ghc-prim ... linking ... done.
Loading package integer-gmp ... linking ... done.
Loading package base ... linking ... done.
[1 of 3] Compiling FODBR            ( FODBR.hs, interpreted )
[2 of 3] Compiling NCCTL            ( NCCTL.hs, interpreted )
[3 of 3] Compiling ManualSimpleExample ( ManualSimpleExample.lhs,
interpreted )
Ok, modules loaded: ManualSimpleExample, NCCTL, FODBR.
*ManualSimpleExample> initF
Conj (Conj (Prop (T,0)) (Conj (Prop (N,0)) (Prop (W,0)))) (Conj (Prop (S,1))
(Prop (E,1)))
*ManualSimpleExample> goalF
Conj (Prop (E,0)) (Prop (W,1))
*ManualSimpleExample> check exampleModel illegal (Conj initF (af goalF))
[]
*ManualSimpleExample> check exampleModel illegal (CD (GEQ 0) (Conj initF (af
goalF)))
[(0,9,0)]
```

Thank you

The paper and all the source code is available on github:
http://pkazmierczak.github.com/NorMC