Model Checking of Robot Gathering

Ha Thi Thu Doan, François Bonnet, and Kazuhiro Ogata

JAIST, Nomi, Japan
Osaka University, Osaka, Japan

OPODIS, December 18, 2018

Outline

- **Motivation**
- **Gathering Problems on Ring**
- **Algorithm for Robot Gathering**
- **Formal Specification**
- **Model Checking**
- **Counterexamples**
- **Conclusion**

Motivation

Distributed mobile robot algorithms → Prove correctness

Challenges

- ***Informal proofs:***
  Difficult to study all cases -> Easy to make mistakes
- ***New area of research:***

The approach to the formal verification of these algorithms

Motivation

Distributed mobile robot algorithms

Prove correctness

Challenges

- ***Informal proofs:***

  Difficult to study all cases -> Easy to make mistakes

- ***New area of research:***

Model Checking of Robot Gathering

Computational Model – Ring

- **Robot**
  - *Anonymous*: identical
  - *No sense of direction,*
  - *Oblivious*: no memory of the past actions.
- **Anonymous ring**
  - Neither nodes nor links are labeled

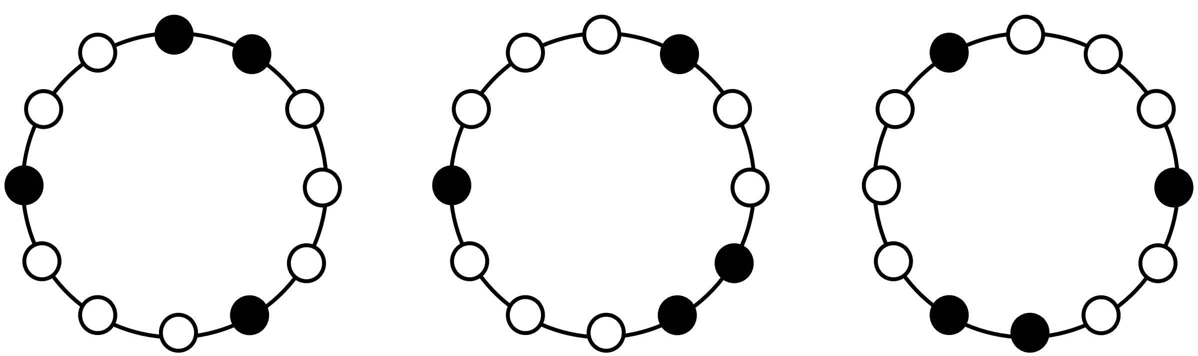
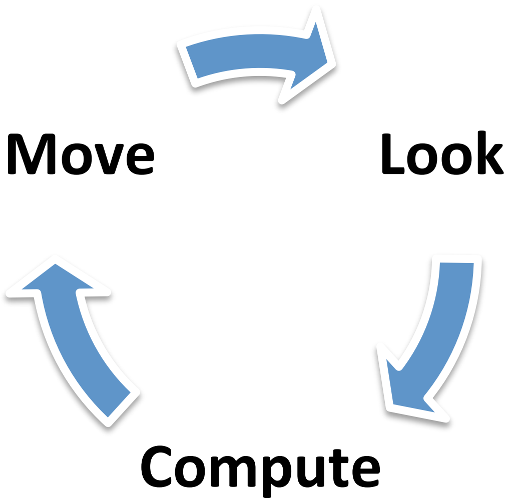


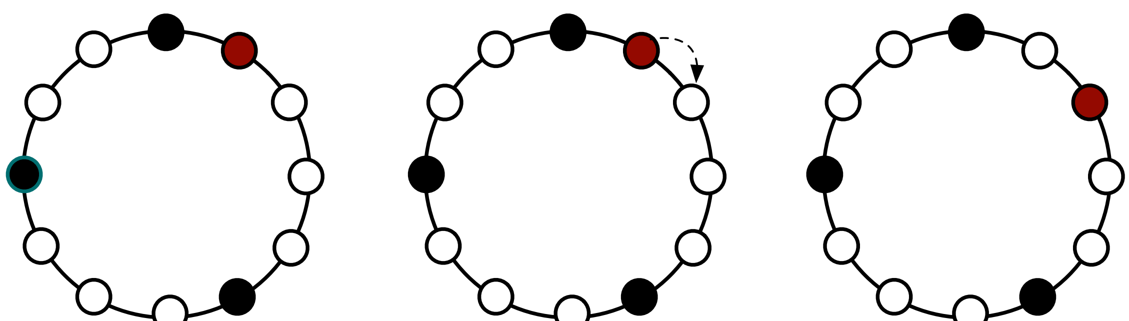
Fig 1. These configurations are considered the same

Computational Model – Ring

- **Look-Compute-Move Phase Behavior**
  i. *Look*: Take a snapshot
  ii. *Compute:* Compute a move
  iii. *Move* (instantaneous): Execute the computed move.

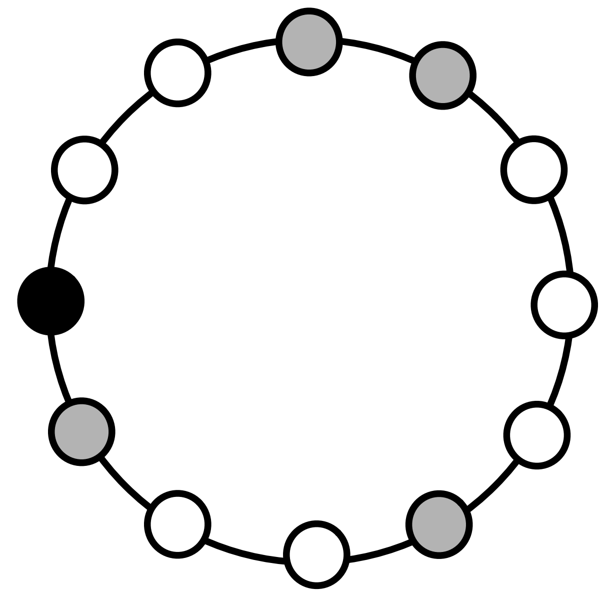


- **Asynchronous scheduler**
  - Phases are asynchronous.
  - Pending move



Computational Model – Ring

- Multiple robots can be located at the same node
- ***Multiplicity detection*** capacity:
  - ✓ empty,
  - ✓ single robot or
  - ✓ more than one (but not the exact number).



Gathering Problem

Robots, initially located at *different* locations, have to gather at the ***same*** location (not determined in advance) and *remain* in it

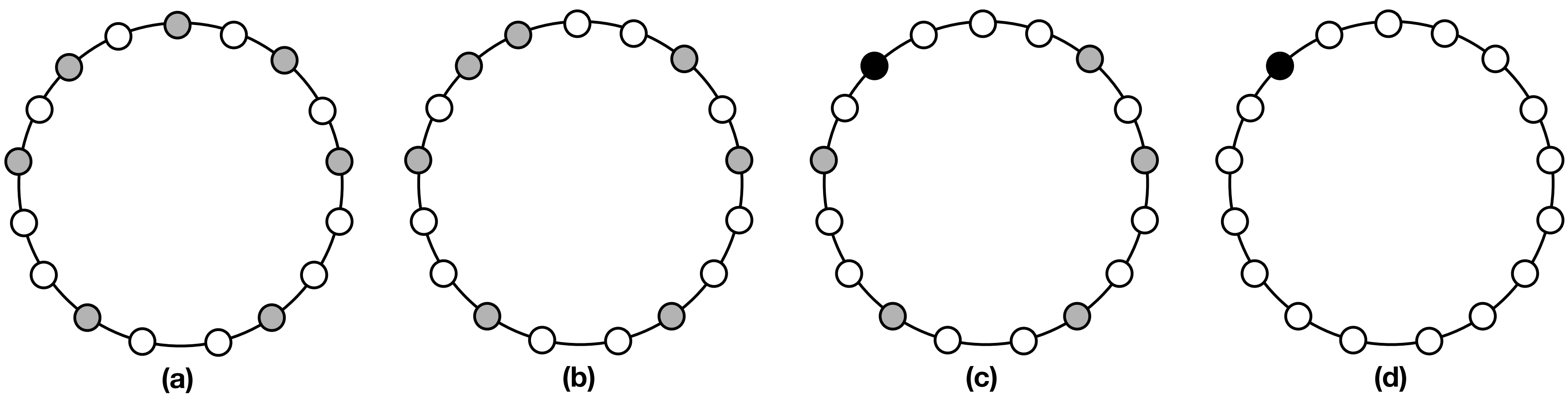
An Algorithm for Robot Gathering

Four phases:

- **MULTIPLICITY-CREATION:** creates either *one* or *two symmetric* multiplicities
- **COLLECT:** moves all but *four robots*
- **MULTIPLICITY-CONVERGENCE:** makes the two multiplicities to *merge into one*
- **CONVERGENCE:** allows the remaining single robots to *join the unique multiplicity*

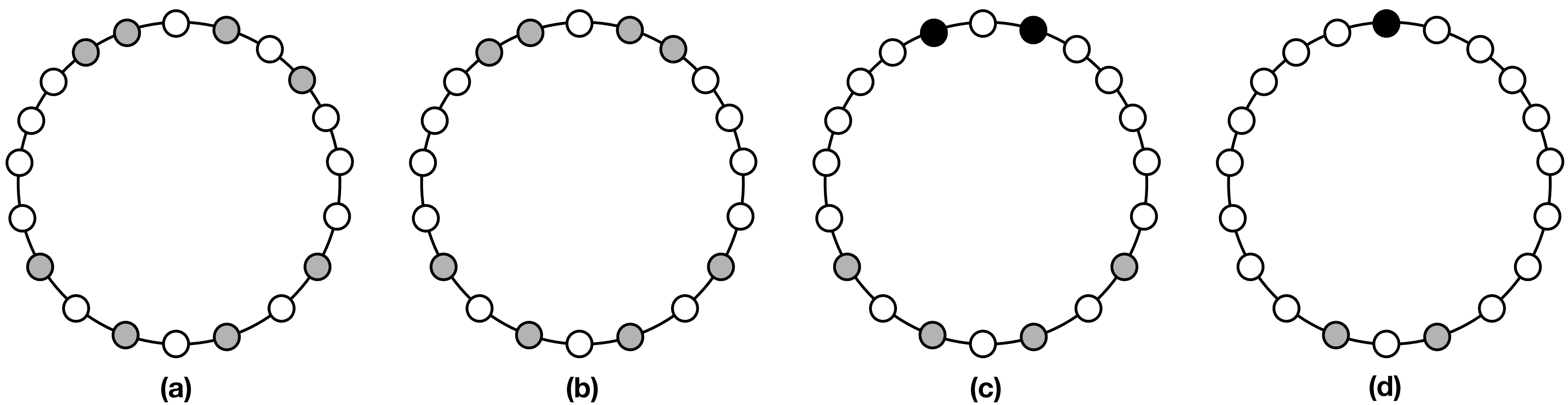
G. D'Angelo, G. Di Stefano, and A. Navarra. Gathering on rings under the look–compute– move model. *Distributed Computing*, 27:255–285, March 2014.

A first look to the algorithm



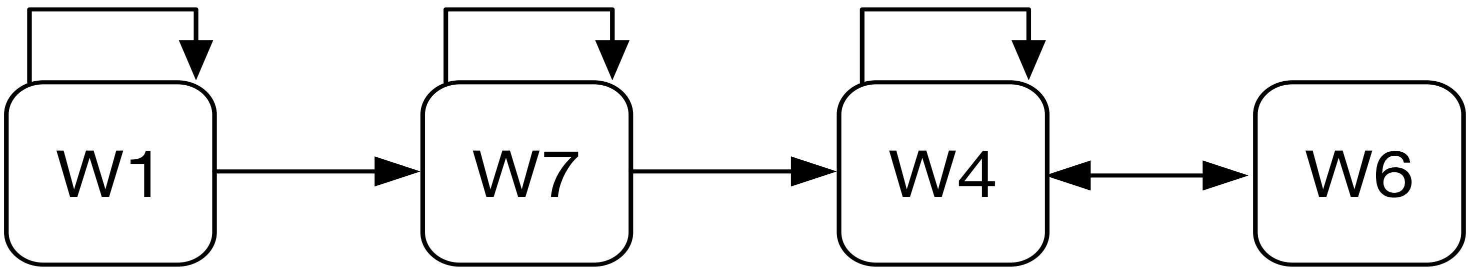
Two examples of possible executions of the gathering algorithm.
Black nodes represent multiplicities

A first look to the algorithm



Two examples of possible executions of the gathering algorithm.
Black nodes represent multiplicities

Classifying Configurations



Procedure: MULTIPLICITY-CREATION
Input: CT, $C = Q(r) = (q_0, q_1, \ldots, q_j)$

case CT $= W1$
  if $C = \overline{(C_j)}$ **then**
  move towards q_0;

case CT $= W2$
  if GATHER-FOUR-NODES(C) **then**
  REDUCTION (C);

Formal Specification

1. State Expression:

ops FC FC- nil : -> Pending [ctor] .

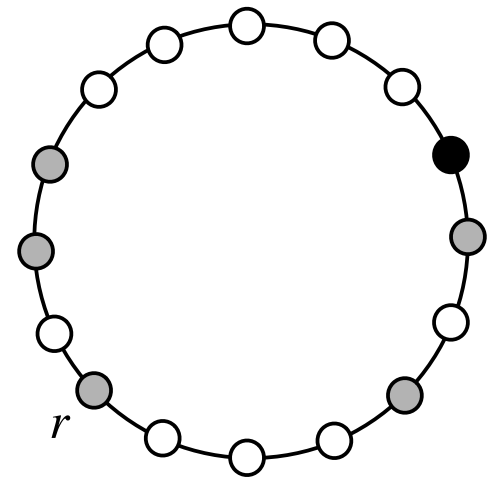
op <_,_> : Int Pending -> Node [ctor] .

op _ _ : Seq Seq -> Seq [ctor assoc] .

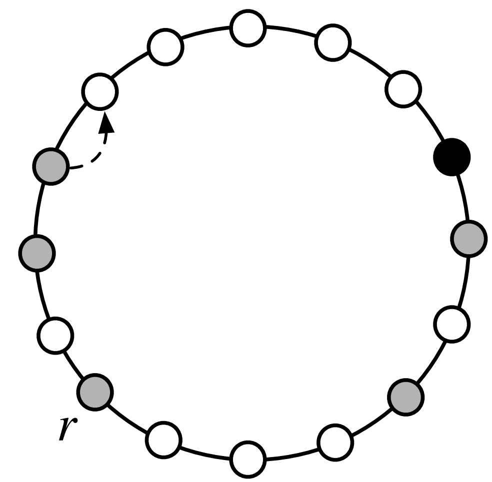
(a) {< 1, nil > < 0, nil > < 5, nil > < -1, nil > < 1, nil > < 3, nil >}

(b) {< 1, nil > < 0, nil > < 5, FC > < -1, nil > < 1, nil > < 3, nil >}

Maude specification language is used!



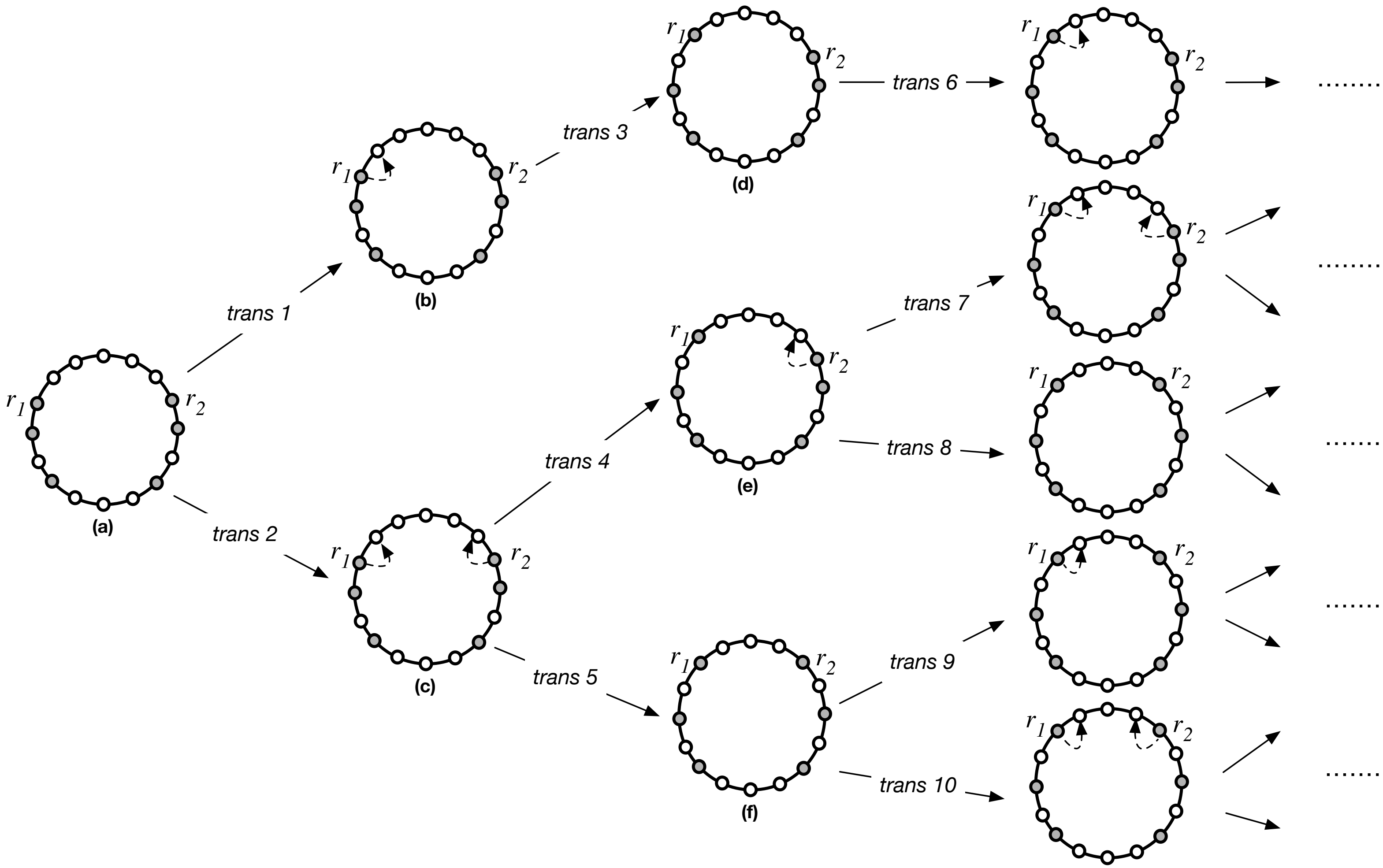
(a)



(b)

Formal Specification

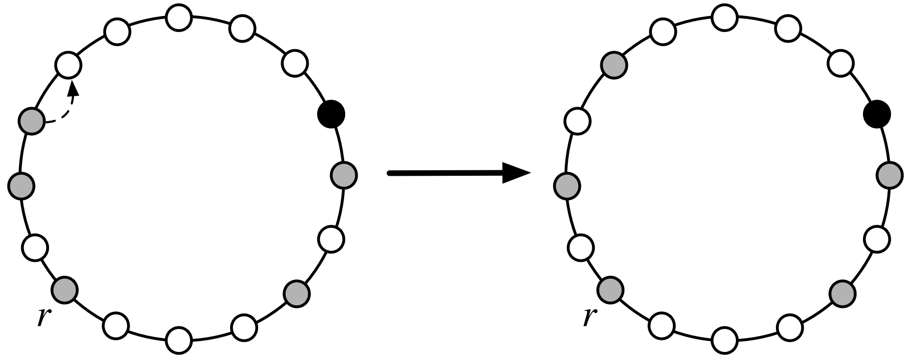
2. State Transitions:



Formal Specification

rl [FC2-pending] :
{S1 < I1, P > < I2, FC > S2} => {S1 < I1 + 1, P > < I2 - 1, nil > S2} .

{< 1, nil > < 0, nil > < 5, FC > < -1, nil > < 1, nil > < 3, nil >}



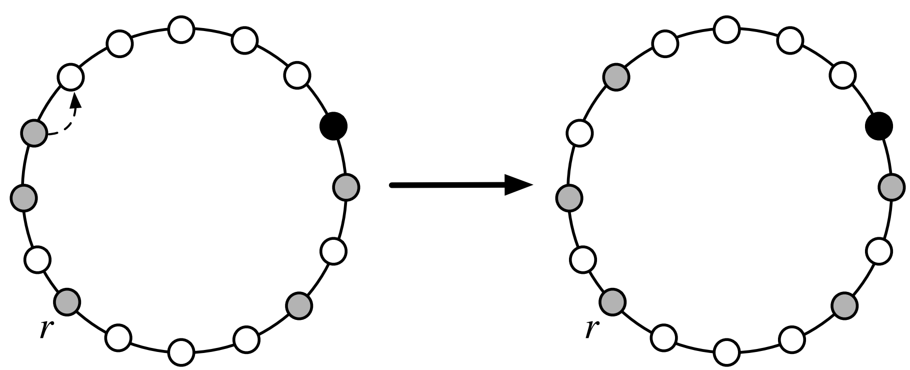
Formal Specification

rl [FC2-pending] :
{S1 < I1, P > < I2, FC > S2} => {S1 < I1 + 1, P > < I2 - 1, nil > S2} .

{< 1, nil > < 0, nil > < 5, FC > < -1, nil > < 1, nil > < 3, nil >}

$\Downarrow$

{< 1, nil > < 1, nil > < 4, FC > < -1, nil > < 1, nil > < 3, nil >}



Formal Specification

```
mod W1-MOVE is
  pr CONFIG-CALCULATION .

  crl [w1-fo] : {S1 < I, nil > S2} => {S1 < I, FC > S2}
      if  checkEqual({< I, nil > S2 S1}) and w1({S1 < I, nil > S2}) .
   …
endm

Mod MULTIPLICITY-CREATION is
  pr W1-MOVE .
  …
endm
```

Formal Specification

```
mod GATHERING is
  pr MULTIPLICITY-CREATION .
  pr COLLECT .
  pr MULTIPLICITY-CONVERGENCE .
  pr CONVERGENCE .
endm
```

Model Checking

- *Lemmas: Lemma 5, 6, and 7*

Lemma 5: *Phase* multiplicity–creation *terminates with either one or two symmetric multiplicities after a finite number of moves.*

Lemma 6: *Phase* collect *terminates after a finite number of moves by reaching a symmetric configuration with six nodes occupied by two multiplicities, two guards and two single robots adjacent to the multiplicities.*

Model Checking

- We took *exactly* the pseudo-code given in the paper

- Informal description of the algorithm (in plain sentence) may be correct, but it is difficult to formalize such informal description!

LTL formulas

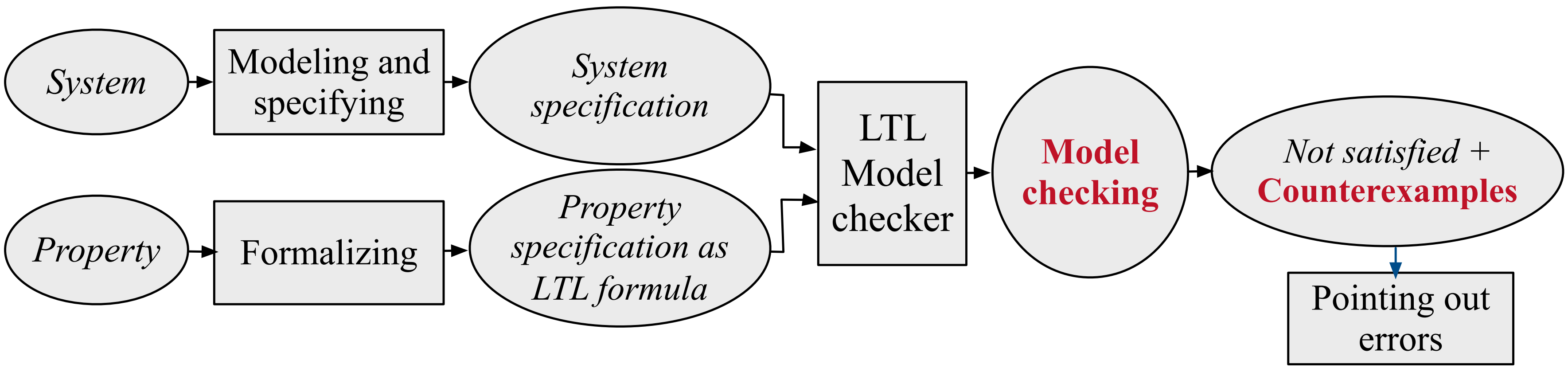
- Specifying Lemmas as Linear Temporal Logic (LTL) formulas

C |= endOfColl = checkOfColl(C) .

C |= coll = checkColl(C) .

lemma6 = $\square$(endOfColl $\rightarrow$ coll) $\wedge$ $\diamond$ endOfColl .

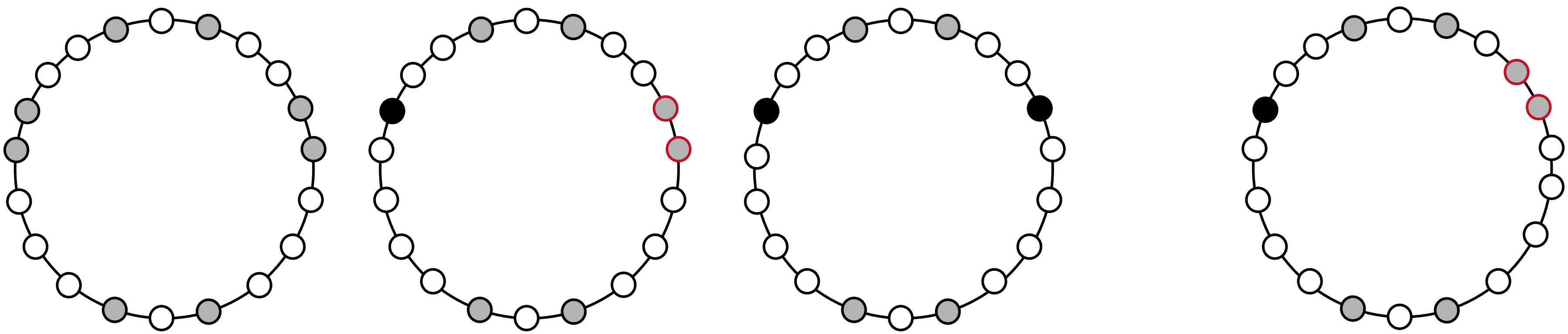
Model Checking



- The model checking: *counterexamples are found.*

Counterexamples

- **Design Errors**



(a) (b) (c) (d)

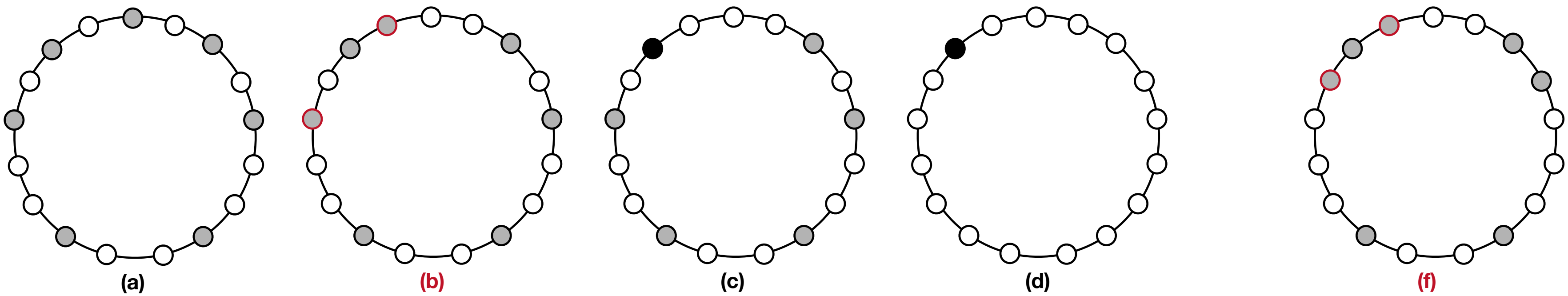
{< -1,nil > < 2,nil > < 1,nil > < 2,nil > < 0,nil > < 3,nil > < 1,nil > < 4,nil >}

{< -1,nil > < 2,nil > < 1,nil > < 2,nil > < 0,FC- > < 3,FC- > < 1,nil > < 4,nil >}

{< -1,nil > < 2,nil > < 1,nil > < 1,nil > < 0,nil > < 4,nil > < 1,nil > < 4,nil >}

Counterexamples

- **Design Errors**



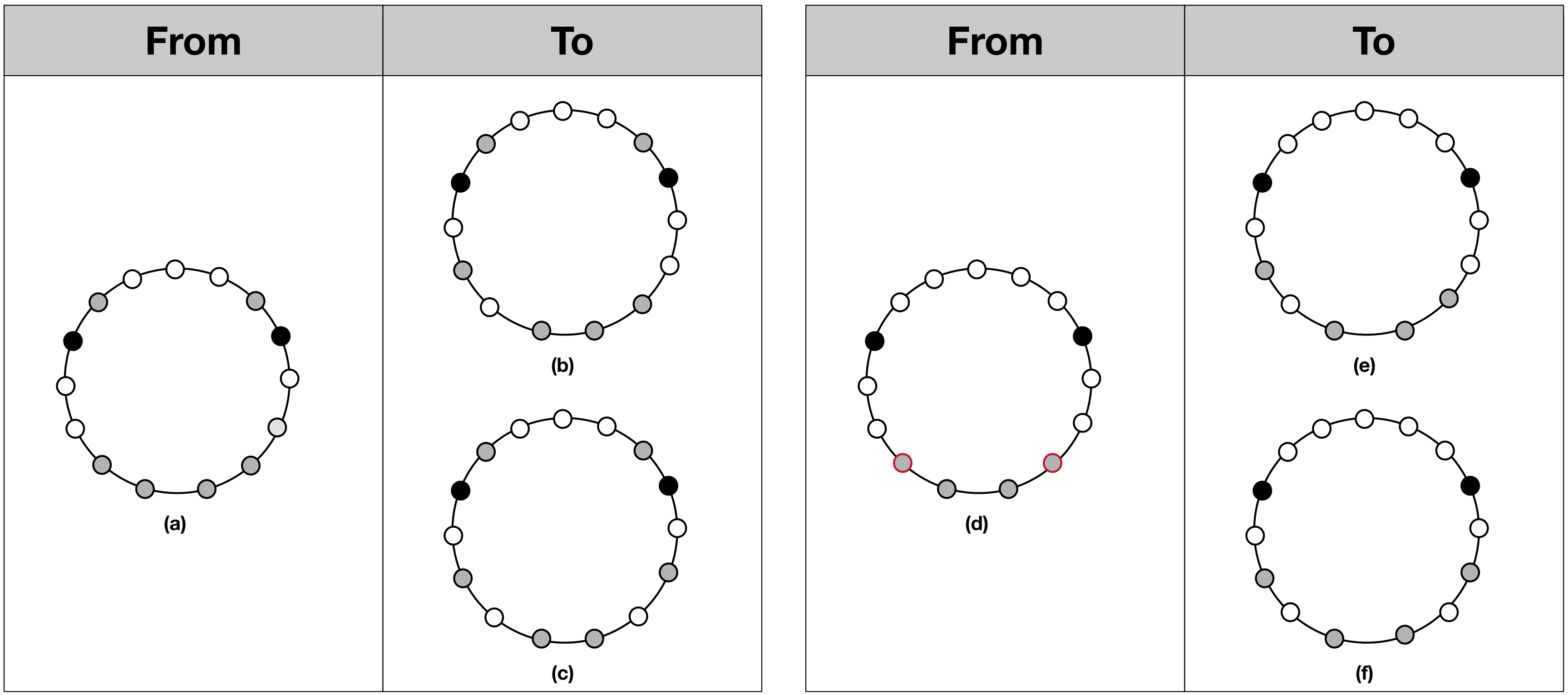
{< 2,nil > < 1,nil > < 0,nil > < 2,nil > < 1,nil > < 2,nil > < 2,nil >} .

{< 2,nil > < 1,FC > < 0,nil > < 2,nil > < 1,nil > < 2,FC- > < 2, nil >}

{< 3,nil > < 0,nil > < 0,nil > < 2,nil > < 0,nil > < 3,nil > < 2,nil >}

Counterexamples

- **Omission of Special Cases**



{< 2,nil > < -1,nil > < 5,nil > < -1,nil > < 2,nil > < 1, nil > < 0,nil > < 1,nil >}

{< 2,nil > < -1,nil > < 5,nil > < -1,nil > < 2,nil > < 1,nil > < 0,nil > < 1,nil >}

Counterexamples

- **Some minor errors**

case CT $= W6$

$C' := (q_0 + 1, q_1 - 1, \ldots, q_j)$;

if $C' = \overline{C'}$ *and* q_0 *is odd* **then** move towards q_0; q0 + 1

;

else

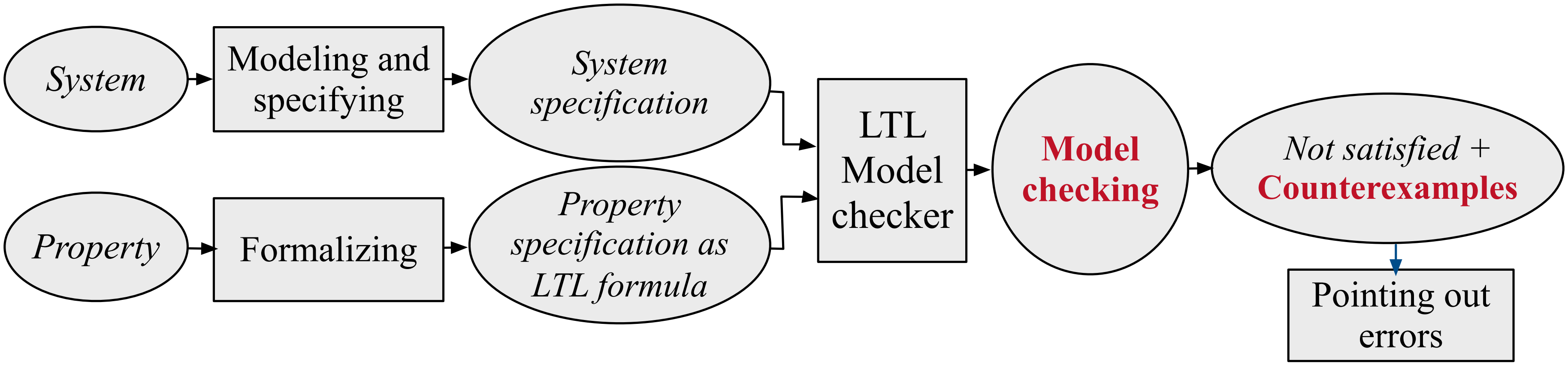
$C'' := (q_0, \ldots, q_{j-1} - 1, q_j + 1)$; qj + 1

if $C''_j = \overline{(C''_j)}$ *and* q_j *is odd* **then**

move towards q_j;

else

Conclusion



- The lemmas are *not satisfied!*

Conclusion

future directions:

- Fixing the mistakes that we found
- Come up with a way to make it concisely specify mobile robot algorithms
- Modeling other algorithms