Abstract Model Repair

George Chatzieleftheriou

Dept. of Informatics

Aristotle University of Thessaloniki, Greece

Borzoo Bonakdarpour

School of Computer Science,

University of Waterloo, Canada

Scott. A. Smolka

Dept. of Computer Science, Stony Brook University, USA

Panagiotis Katsaros

Dept. of Informatics

Aristotle University of Thessaloniki, Greece

The Model Repair problem

- Given a model M and a property ϕ, where M does not satisfy ϕ, obtain a new model M' such that M' satisfies ϕ.
  - Moreover, the changes made to M to derive M' should be minimal with respect to all such M'.

Motivation

- Algorithms for model repair strongly depend on the size of the model
- Inapplicable to models with large state space
  - State space explosion problem
- Success of abstraction-based model checking
- Objective
  - Find a way to use abstraction to make repair process applicable to large models

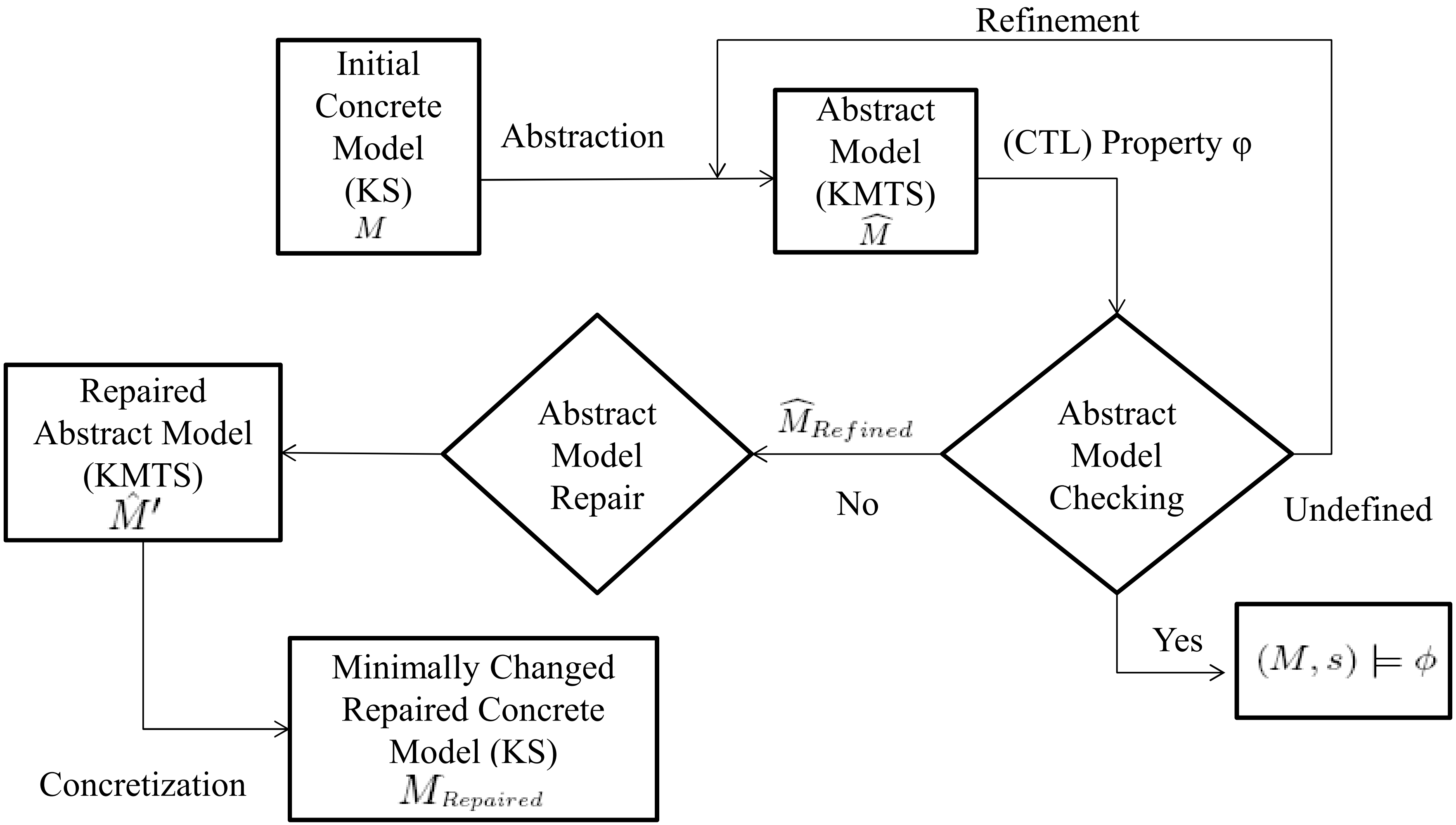
Main Contributions (1)

- Abstract Model Repair (AMR) framework
  - Kripke Structures (KSs) for concrete models
  - Kripke Modal Transition Systems (KMTSs) for abstract models
  - Computation Tree Logic (CTL) for property specification language
  - Use of abstraction and refinement

Main Contributions (2)

- Metric space on KSs to handle minimality of changes constraint
- Abstract Model Repair algorithm
- Application to an Automatic Door Opener system [Baier, Katoen MC book]

Abstract Model Repair Framework


Refinement
Initial Concrete Model (KS) M
Abstraction
Abstract Model (KMTS) $\widehat{M}$
(CTL) Property φ
Repaired Abstract Model (KMTS) $\hat{M}'$
Abstract Model Repair
$\widehat{M}_{Refined}$
No
Abstract Model Checking
Undefined
Yes
$(M, s) \models \phi$
Minimally Changed Repaired Concrete Model (KS) $M_{Repaired}$
Concretization

KMTSs as abstract models of KSs

- Must-transitions (under-approximation)
  - Concrete transitions exist from all the corresponding concrete states
- May-transitions (over-approximation)
  - At least one concrete transition exists from one of the corresponding concrete states
- Preservation theorem

$$[(\hat{M},\hat{s}) \models \varphi] \neq \perp \Rightarrow [(M,s) \models \varphi] = [(\hat{M},\hat{s}) \models \phi]$$

3-valued CTL MC over KMTSs

- Result of 3-valued MC ∊ {True, False, *Undefined*}
- May-transitions are used to check the truth of universal CTL properties
- Must-transitions are used to check the truth of existential CTL properties
- Vice versa for the falsity of CTL properties

Refinement

- What happens when answer of 3-valued model checking is undefined?
  - *Refinement* of the abstract KMTS to acquire a more precise but larger abstract model
  - How?
    - Identify the *failure* state
    - Eliminate the cause of failure for this state
      - May-transition
      - An atomic proposition whose value is unknown at this state

Metric Space on KSs

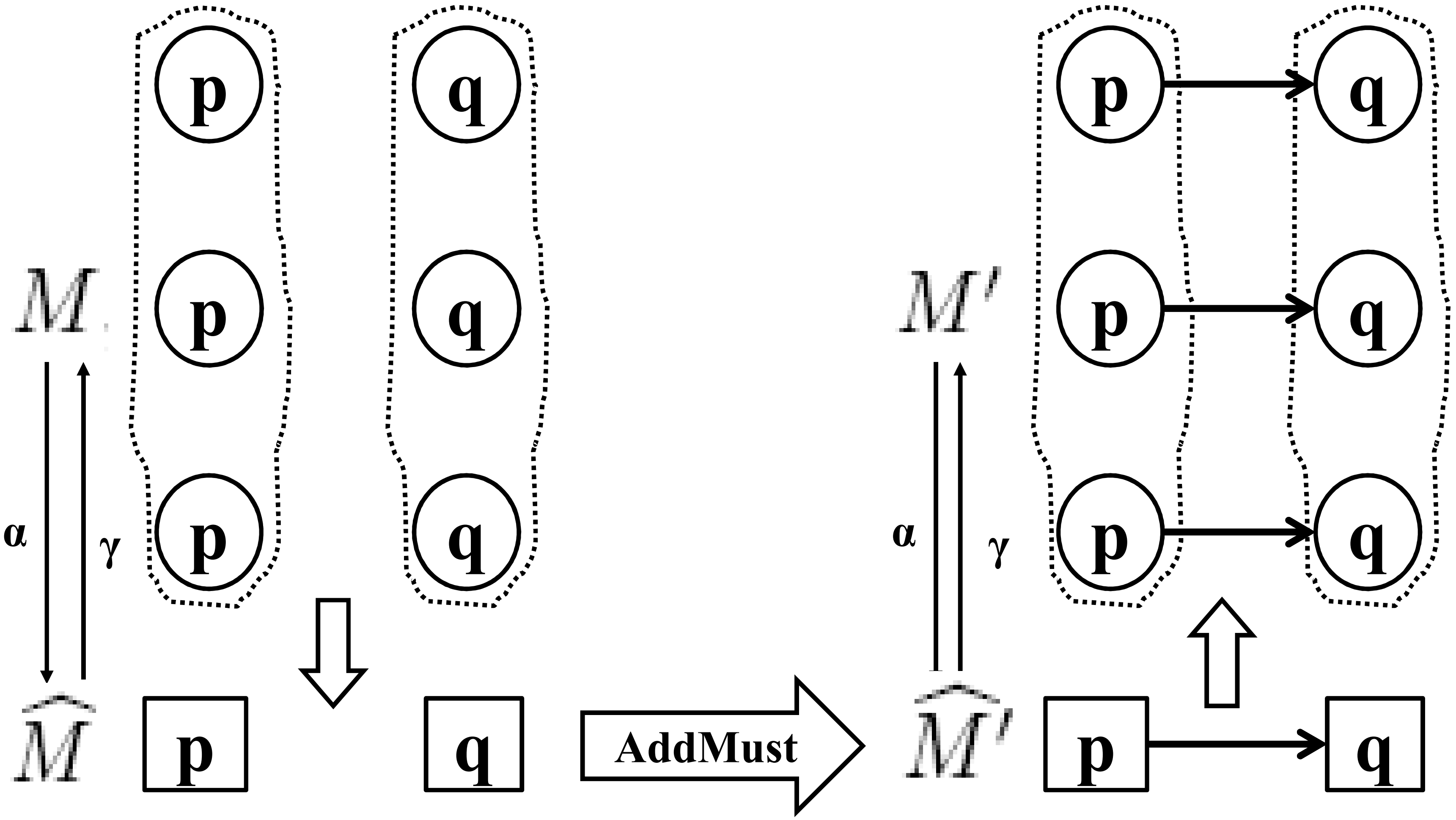
- Based on:
  - symmetric difference between the state space of the KSs
  - symmetric difference between the transition relation of the KSs
  - number of common states with altered labeling

$$d(M,M') = |S \Delta S'| + |R \Delta R'| + \frac{|G(L \restriction_{S \cap S'}) \Delta G(L' \restriction_{S \cap S'})|}{2}$$

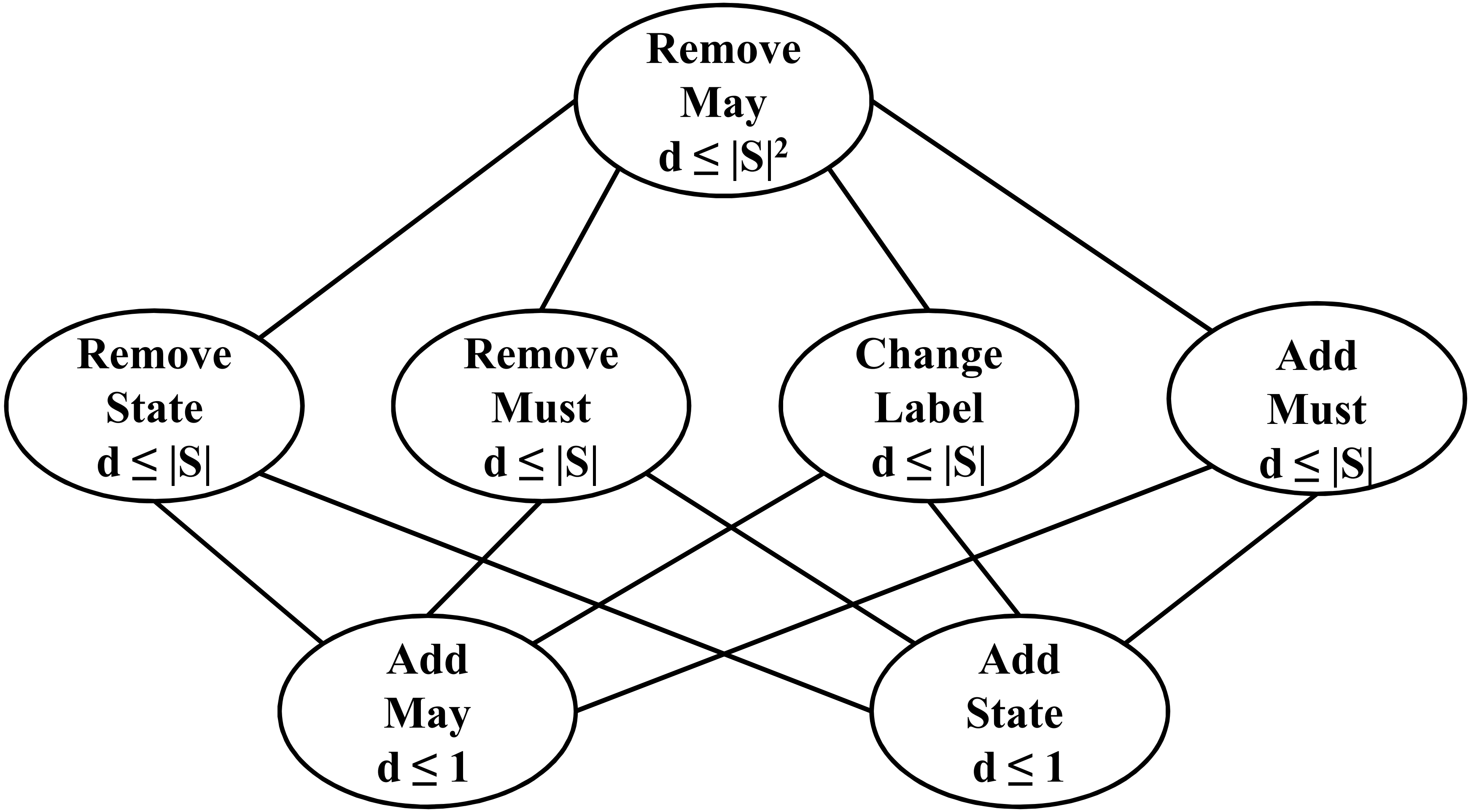
Basic Repair Operations

- **AddMust**: Adding a must-transition
- **AddMay**: Adding a may-transition
- **RemoveMust**: Removing an existing must-transition
- **RemoveMay**: Removing an existing may-transition
- **ChangeLabel**: Changing the labeling of a KMTS state
- **AddState**: Adding a new KMTS state
- **RemoveState**: Removing a disconnected KMTS state

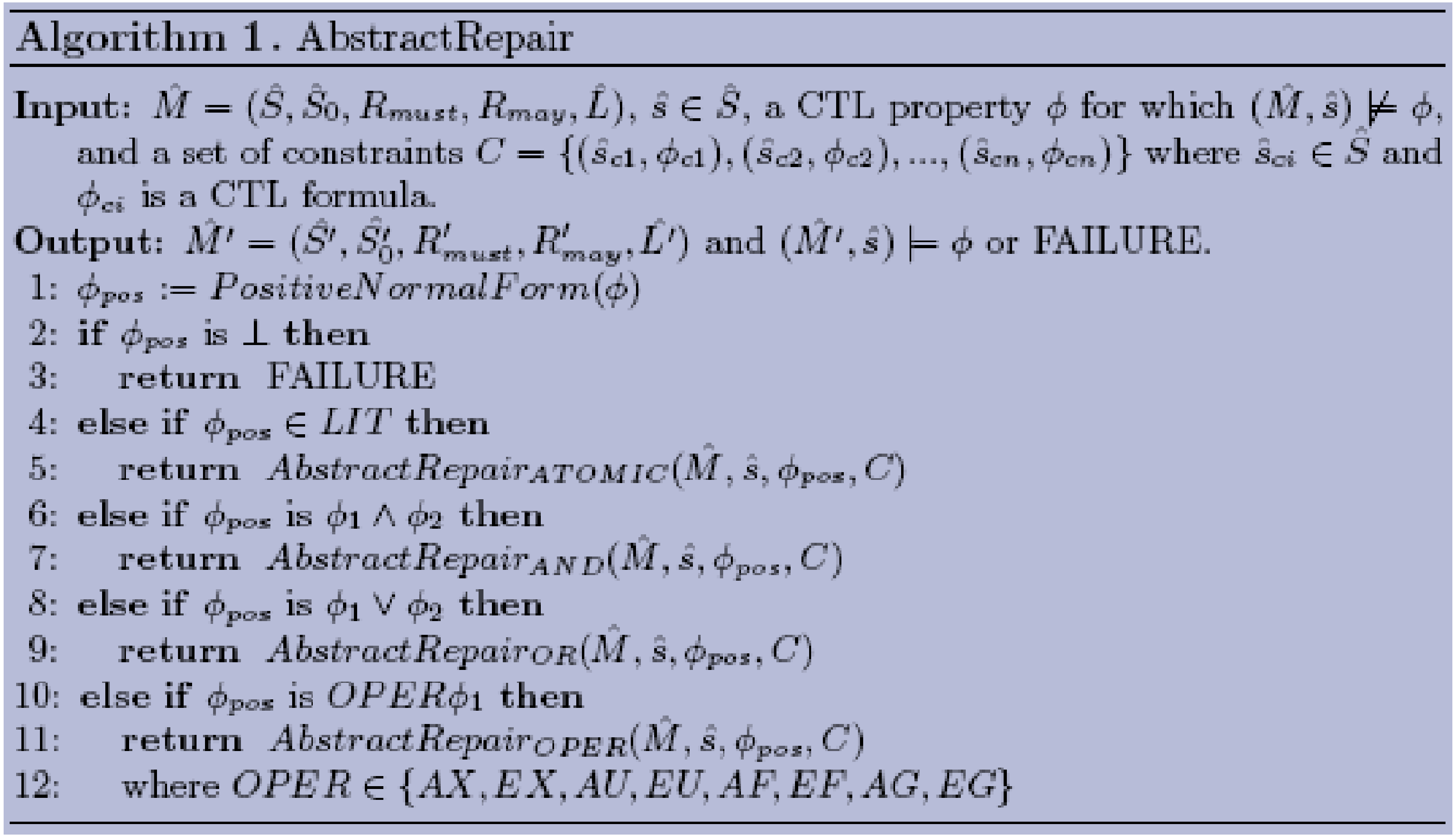
AddMust


M
α
γ
$\widehat{M}$
p
p
p
p
q
q
q
q
AddMust
M'
α
γ
$\widehat{M}'$
p
p
p
p
q
q
q
q

Minimality of Changes Ordering of Basic Repair Operations



Abstract Model Repair Algorithm



Algorithm 1. AbstractRepair

Input: $\hat{M} = (\hat{S}, \hat{S}_0, R_{must}, R_{may}, \hat{L})$, $\hat{s} \in \hat{S}$, a CTL property ϕ for which $(\hat{M}, \hat{s}) \not\models \phi$, and a set of constraints $C = \{(\hat{s}_{c1}, \phi_{c1}), (\hat{s}_{c2}, \phi_{c2}), ..., (\hat{s}_{cn}, \phi_{cn})\}$ where $\hat{s}_{ci} \in \hat{S}$ and ϕ_{ci} is a CTL formula.

Output: $\hat{M}' = (\hat{S}', \hat{S}_0', R_{must}', R_{may}', \hat{L}')$ and $(\hat{M}', \hat{s}) \models \phi$ or FAILURE.

```
$\phi_{pos} := PositiveNormalForm(\phi)$
if $\phi_{pos}$ is $\bot$ then
    return FAILURE
else if $\phi_{pos} \in LIT$ then
    return $AbstractRepair_{ATOMIC}(\hat{M}, \hat{s}, \phi_{pos}, C)$
else if $\phi_{pos}$ is $\phi_1 \wedge \phi_2$ then
    return $AbstractRepair_{AND}(\hat{M}, \hat{s}, \phi_{pos}, C)$
else if $\phi_{pos}$ is $\phi_1 \vee \phi_2$ then
    return $AbstractRepair_{OR}(\hat{M}, \hat{s}, \phi_{pos}, C)$
else if $\phi_{pos}$ is $OPER\phi_1$ then
    return $AbstractRepair_{OPER}(\hat{M}, \hat{s}, \phi_{pos}, C)$
    where $OPER \in \{AX, EX, AU, EU, AF, EF, AG, EG\}$
```

AbstractRepair$_{EX}$

- If $\phi = EX\phi_1$
  1. Adding a must-transition to a state which satisfies ϕ_1
  2. Changing label to an immediate must-successor of the input state in order to make it satisfy ϕ_1
  3. Adding a new state and repeat steps 2 and 1.

Constraints

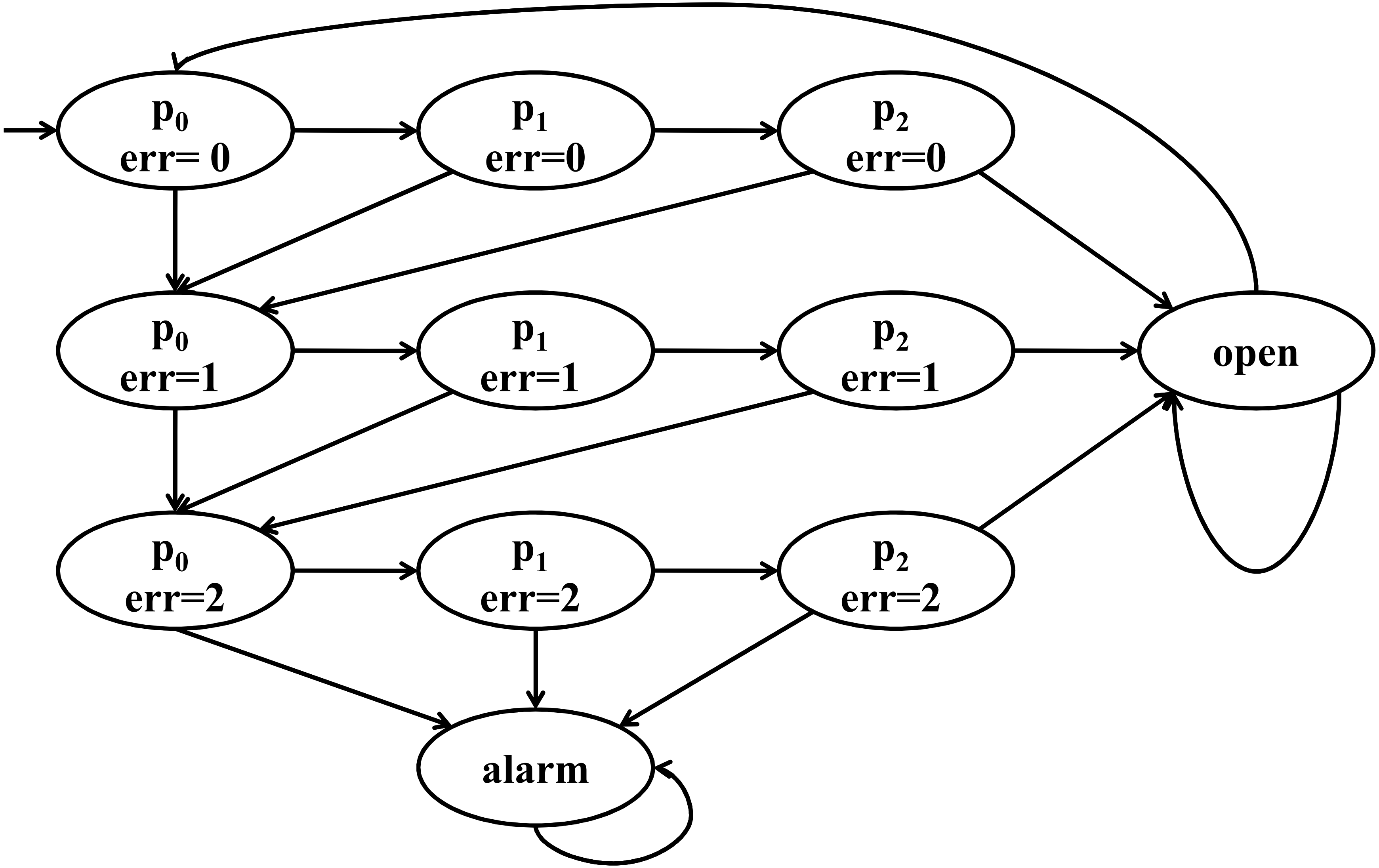
- Constraints are used to handle conjunctive formulas
  - If $\phi = \phi_1$ AND ϕ_2 , ϕ_2 is used as a constraint property when our algorithm tries to find a repaired model for ϕ_1.
  - Conjunctive formulas cannot be handled without the use of constraints

Properties of AMR algorithm

- Well-defined
  - All possible cases are handled
  - Each algorithm step is deterministically defined
    - Even concrete model repair algorithms lack this feature
- Sound
  - If it returns a KMTS, then this KMTS satisfies the input CTL formula ϕ.
- **Does not depend on the size of the concrete model!**

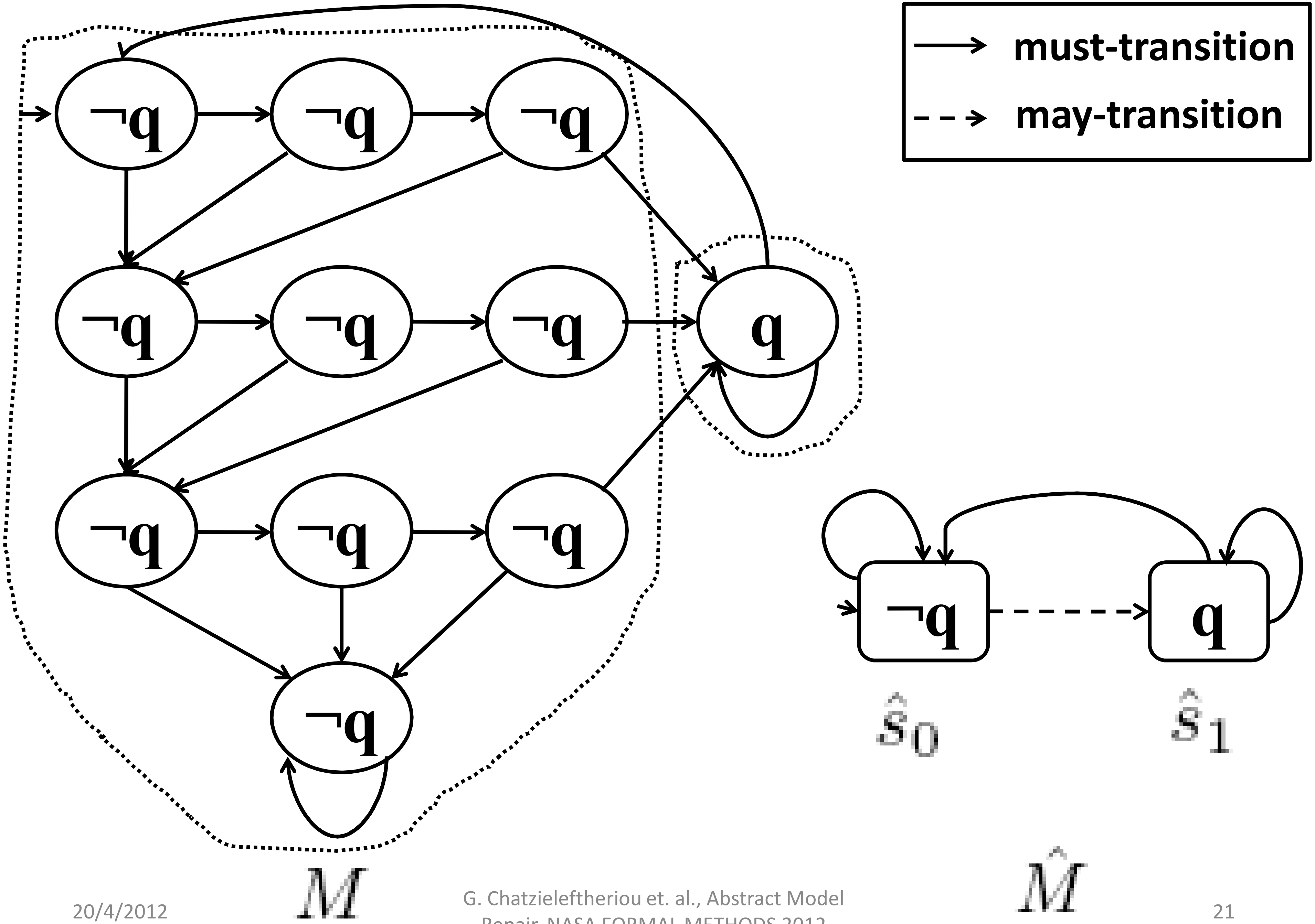
Application

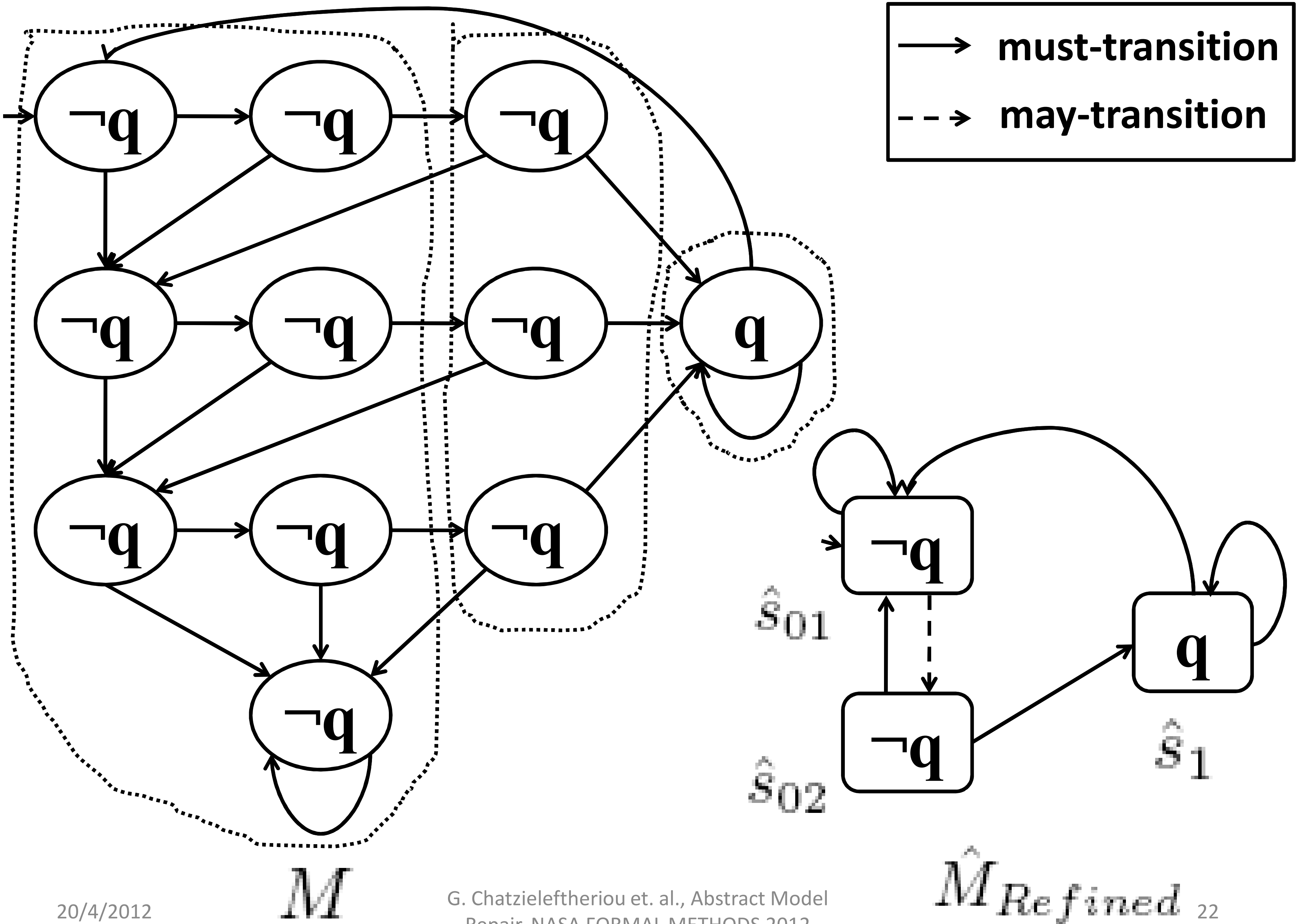
ADO System

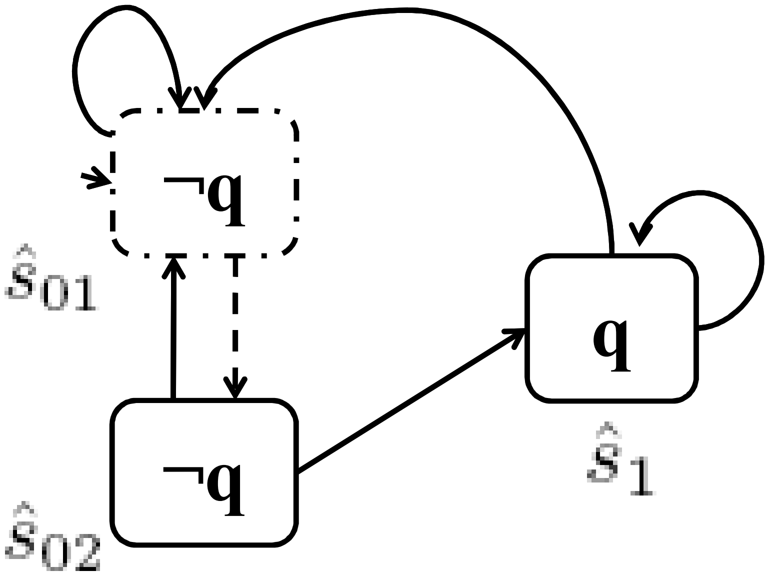

p_0 err= 0
p_1 err=0
p_2 err=0
p_0 err=1
p_1 err=1
p_2 err=1
open
p_0 err=2
p_1 err=2
p_2 err=2
alarm

Specification of the property

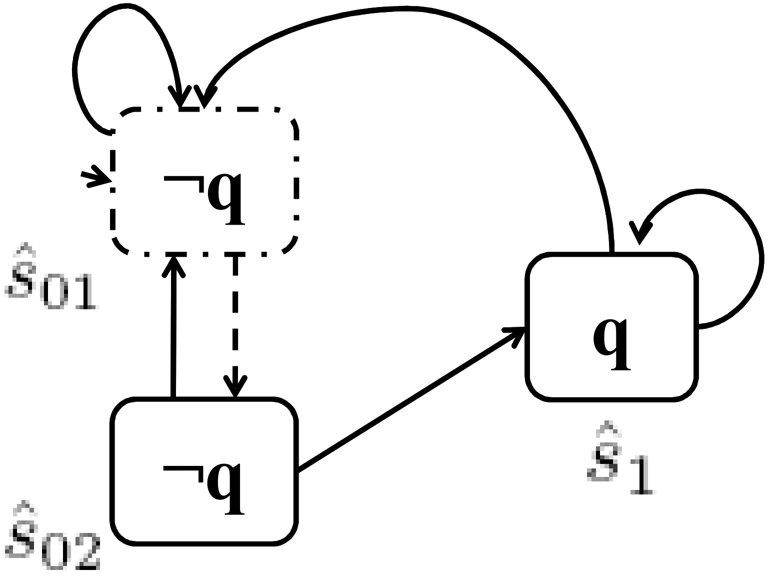
- In the given system, from all states there should be an option to open the door in the next step (e.g. for emergency reasons) (invariant property).
- Specification of property in CTL
  - CTL property
    - **AGEXq, where q = (open == true)**


must-transition
may-transition
¬q
¬q
¬q
¬q
¬q
¬q
q
¬q
¬q
¬q
¬q
¬q
q
$\hat{s}_0$
$\hat{s}_1$
M
$\hat{M}$

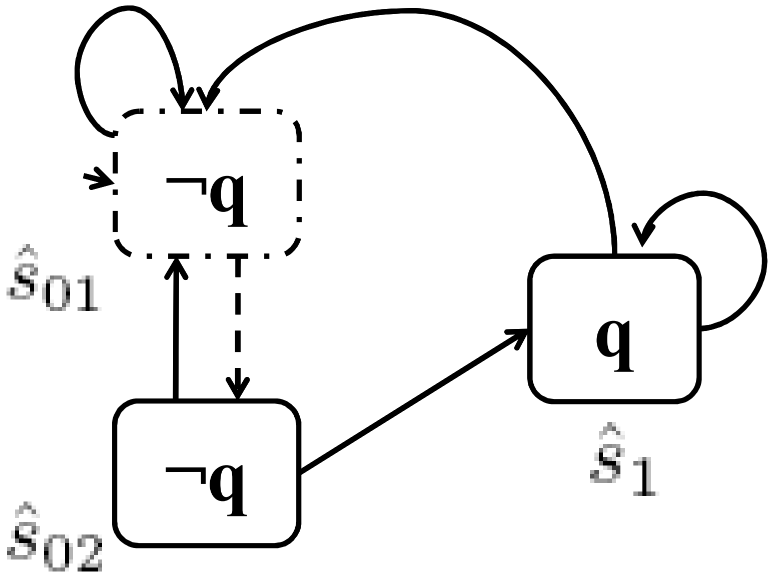

must-transition
may-transition
¬q
q
$\hat{s}_{01}$
$\hat{s}_{02}$
$\hat{s}_1$
M
$\hat{M}_{Refined}$



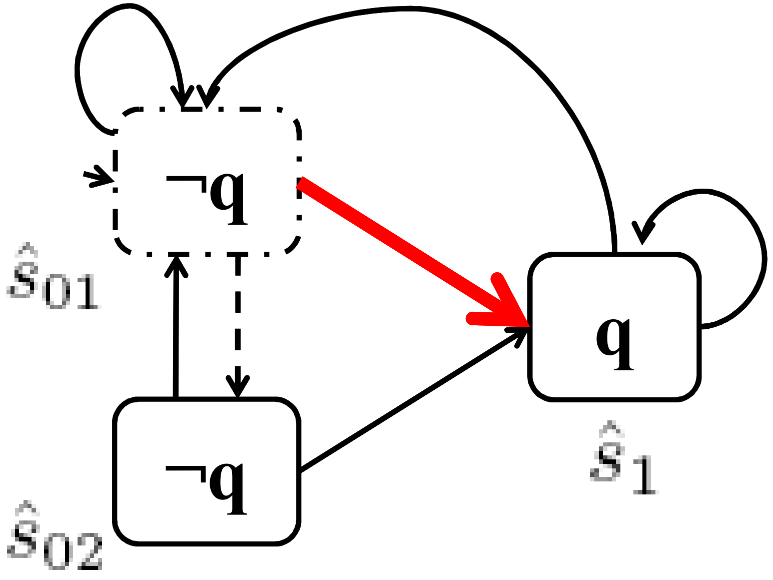
(Step 1) AbstractRepair



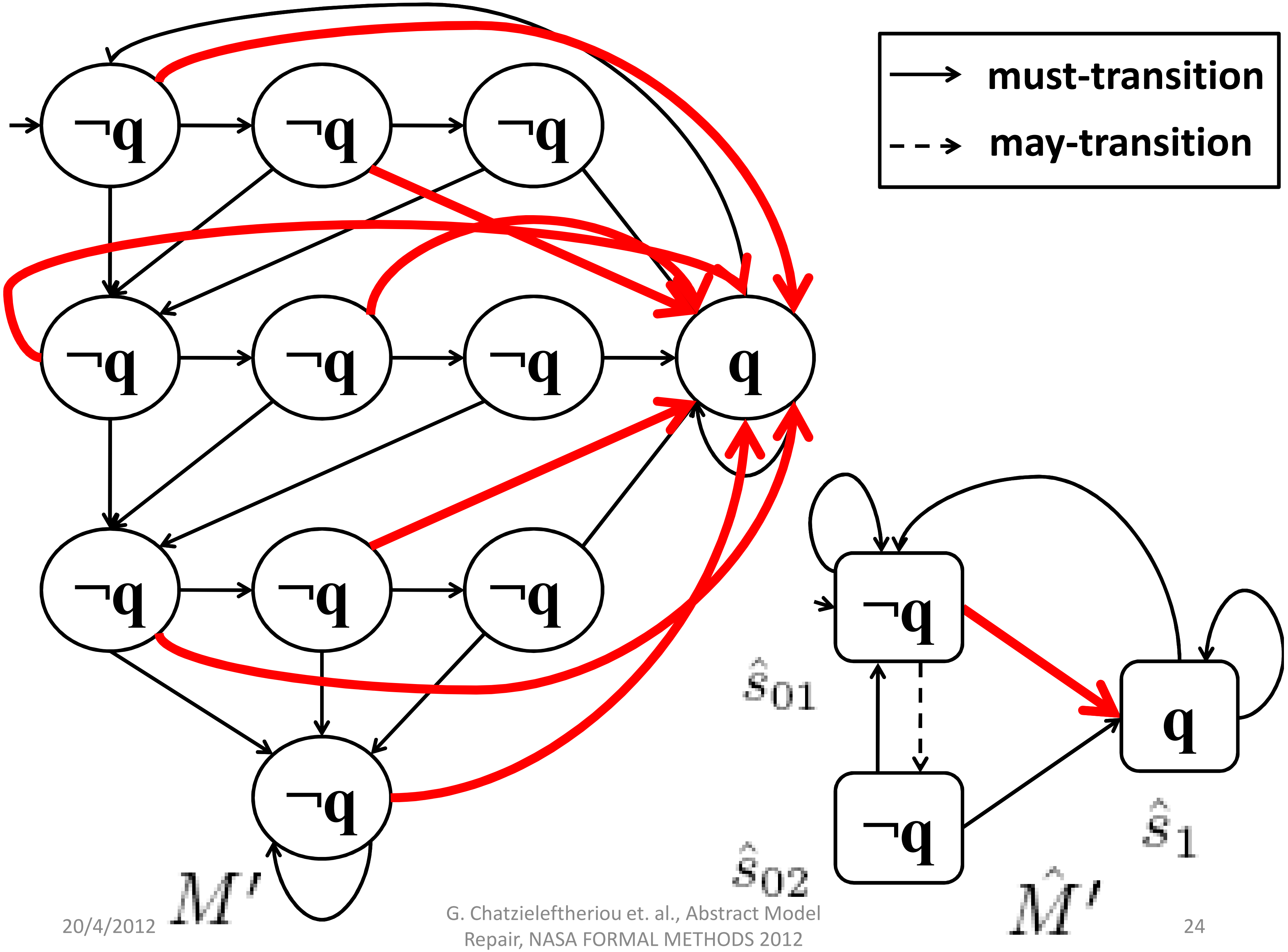
(Step 2) AbstractRepair$_{AG}$



(Step 3) AbstractRepair$_{EX}$



(Step 4) AddMust



→ must-transition

⇢ may-transition

Related Work

- Concrete model repair algorithms
  - State explosion problem in their approach
- Attempts to fight state space explosion
  - Restricted to ACTL
  - Extend CTL with new operators
- Abstract interpretation has been used in *program synthesis*

Summary

- Abstract Model Repair
  - Use of abstraction to fight the state explosion problem of Model Repair
  - Make repair applicable to large systems
- Metric space on Kripke Structures
  - Minimality of changes is taken into account during the repair process
- Sound algorithm for automating the process
- http://mathind.csd.auth.gr/abstract_repair