Verification of FIFO Automata by Abstraction Refinement

Alexander Heußner[1] Tristan Le Gall[2] Grégoire Sutre[3]

[1] Université Libre de Bruxelles, Brussels, Belgium

[2] CEA, LIST, DILS/LMeASI, Gif-sur-Yvette, France

[3] Univ. Bordeaux & CNRS, LaBRI, UMR 5800, Talence, France

VACSIM Meeting, LaBRI, March 2012

For more details: `http://hal.archives-ouvertes.fr/hal-00380517/`

FIFO Systems

- communicating processes / threads / peers / …
  - complex interactions due to concurrency
  - need for algorithmic tool support

- processes are modeled as finite state automata
- communication is asynchronous, via channels
  - first-in first-out,
  - reliable (no loss, no insertion !),
  - and unbounded

- operational semantics given by an infinite transition system

FIFO Systems

- communicating processes / threads / peers / …
  - complex interactions due to concurrency
  - need for algorithmic tool support

- processes are modeled as finite state automata
- communication is asynchronous, via channels
  - first-in first-out,
  - reliable (no loss, no insertion !),
  - and unbounded

for example:
TCP-connections
as base of Berkeley
Socket API based
distributed systems

- operational semantics given by an infinite transition system

FIFO Systems

- communicating processes / threads / peers / …
  - complex interactions due to concurrency
  - need for algorithmic tool support

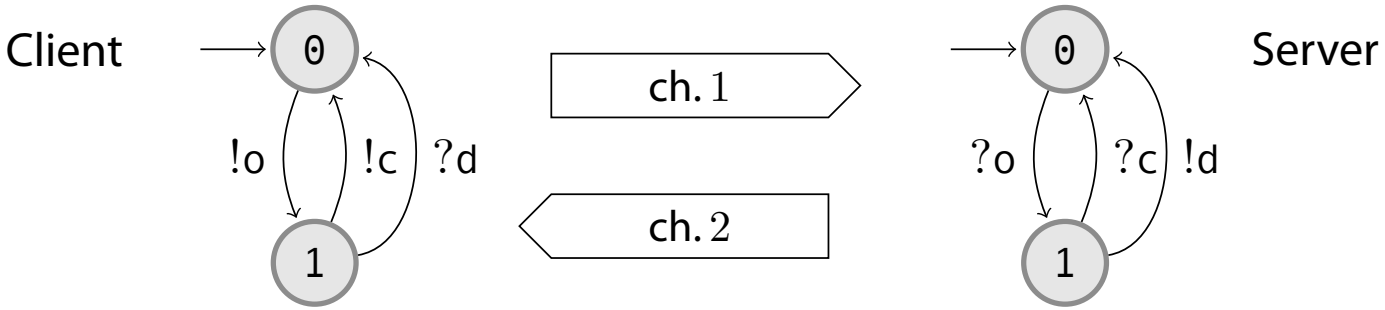
- processes are modeled as finite state automata
- communication is asynchronous, via channels
  - first-in first-out,
  - reliable (no loss, no insertion !),
  - and unbounded

for example:
TCP-connections
as base of Berkeley
Socket API based
distributed systems

- operational semantics given by an infinite transition system

FIFO Systems

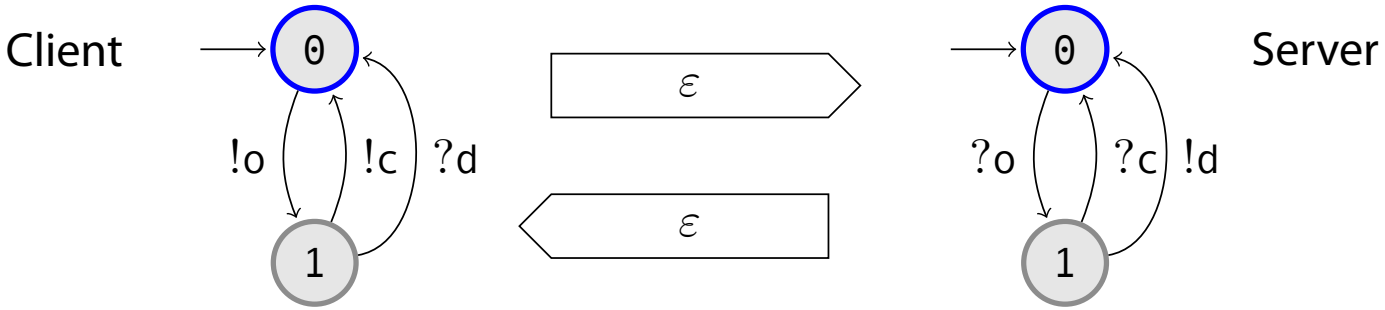
Example: Connection / Disconnection Protocol



- protocol originally "borrowed" from [Jard/Raynal '86]

FIFO Systems

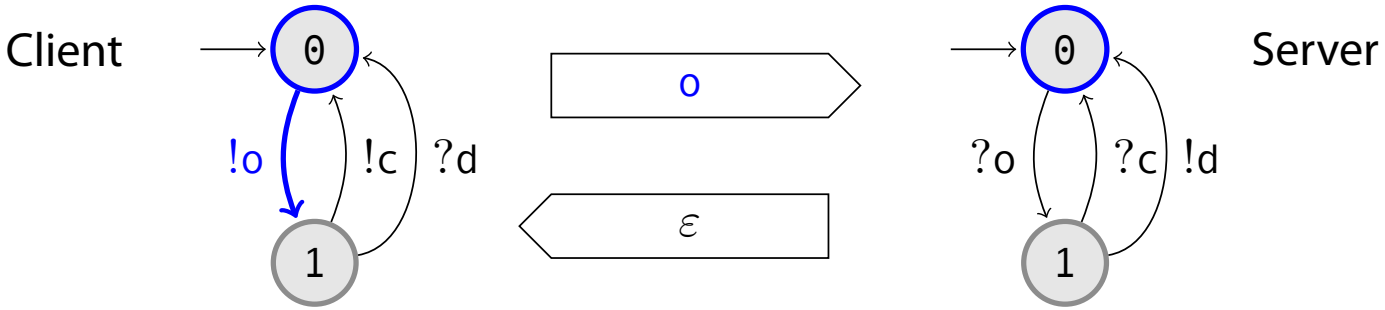
Example: Connection / Disconnection Protocol



$$\langle (0, 0), (\varepsilon, \varepsilon) \rangle$$

FIFO Systems

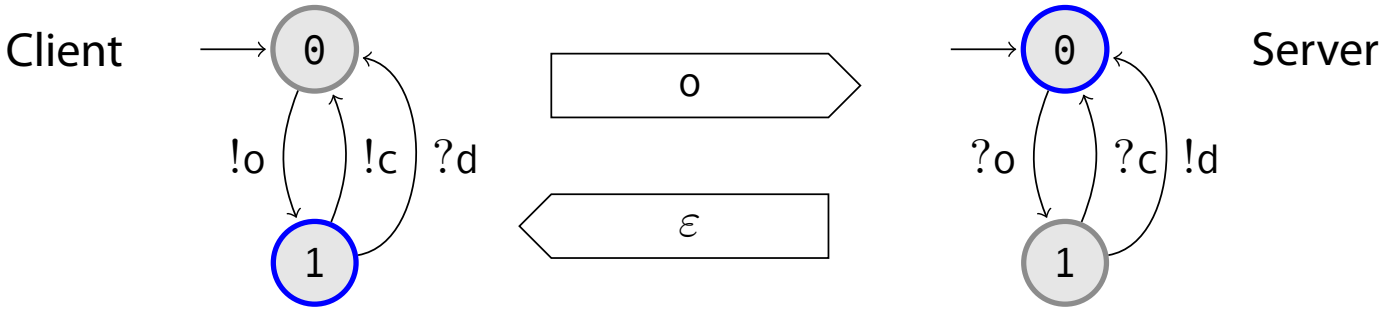
Example: Connection / Disconnection Protocol



$$\langle (0,0),(\varepsilon,\varepsilon)\rangle \xrightarrow{!o}$$

FIFO Systems

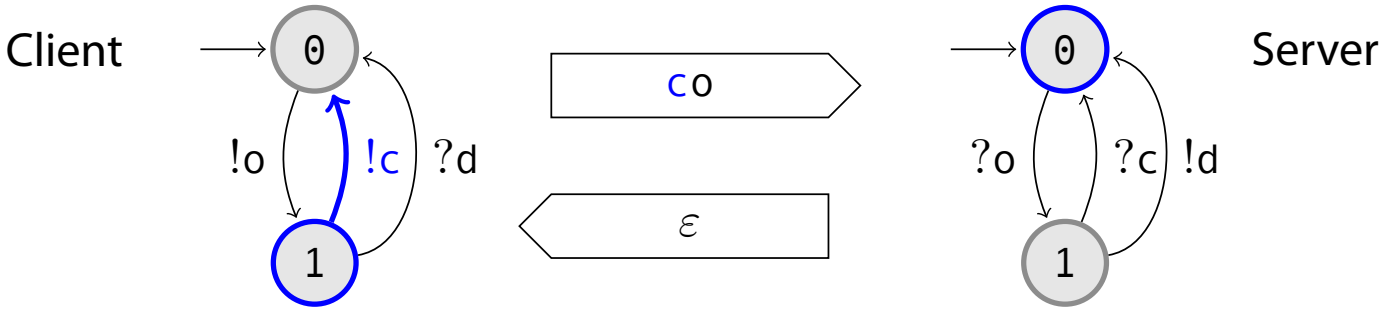
Example: Connection / Disconnection Protocol



$$\langle (0,0),(\varepsilon,\varepsilon)\rangle \xrightarrow{!o} \langle (1,0),(o,\varepsilon)\rangle$$

FIFO Systems

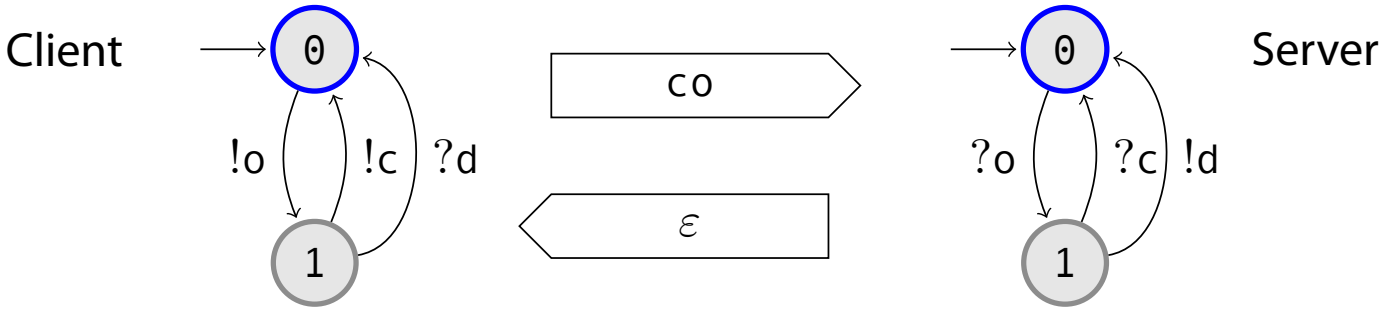
Example: Connection / Disconnection Protocol



$$\langle (0,0),(\varepsilon,\varepsilon)\rangle \xrightarrow{!o} \langle (1,0),(o,\varepsilon)\rangle \xrightarrow{!c}$$

FIFO Systems

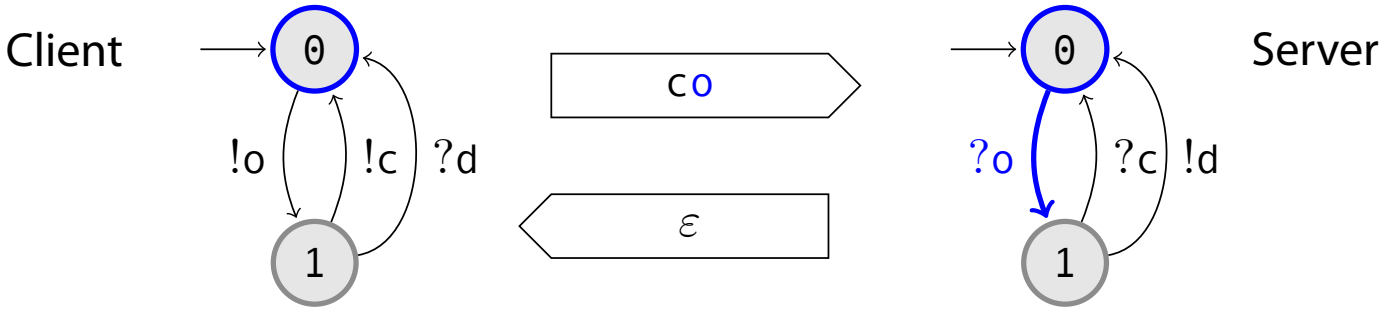
Example: Connection / Disconnection Protocol



$$\langle (0,0),(\varepsilon,\varepsilon)\rangle \xrightarrow{!o} \langle (1,0),(o,\varepsilon)\rangle \xrightarrow{!c} \langle (0,0),(oc,\varepsilon)\rangle$$

FIFO Systems

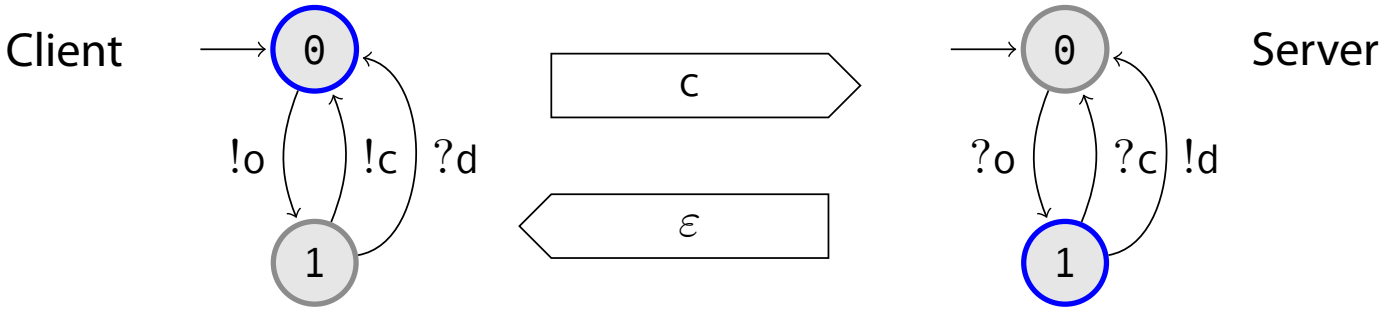
Example: Connection / Disconnection Protocol



$$\langle (0,0),(\varepsilon,\varepsilon)\rangle \xrightarrow{!o\ !c} \langle (0,0),(oc,\varepsilon)\rangle \xrightarrow{?o}$$

FIFO Systems

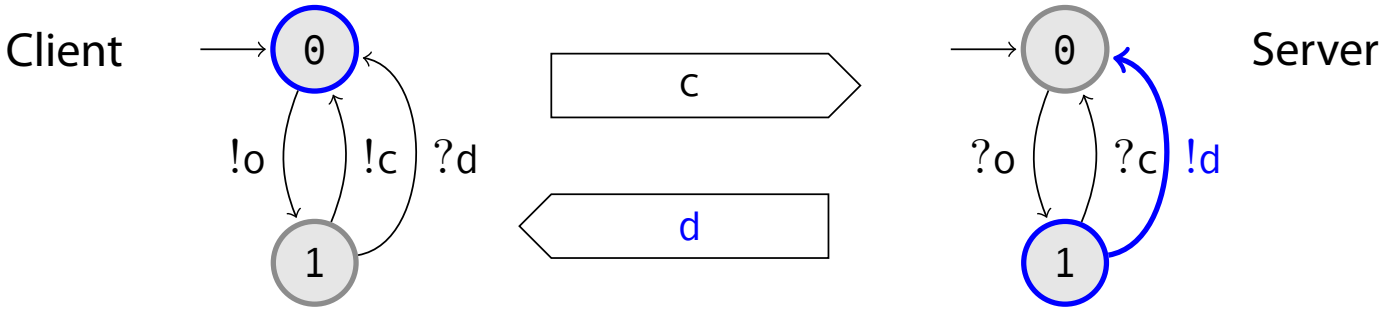
Example: Connection / Disconnection Protocol



$$\langle (0,0),(\varepsilon,\varepsilon)\rangle \xrightarrow{!o\ !c} \langle (0,0),(oc,\varepsilon)\rangle \xrightarrow{?o} \langle (0,1),(c,\varepsilon)\rangle$$

FIFO Systems

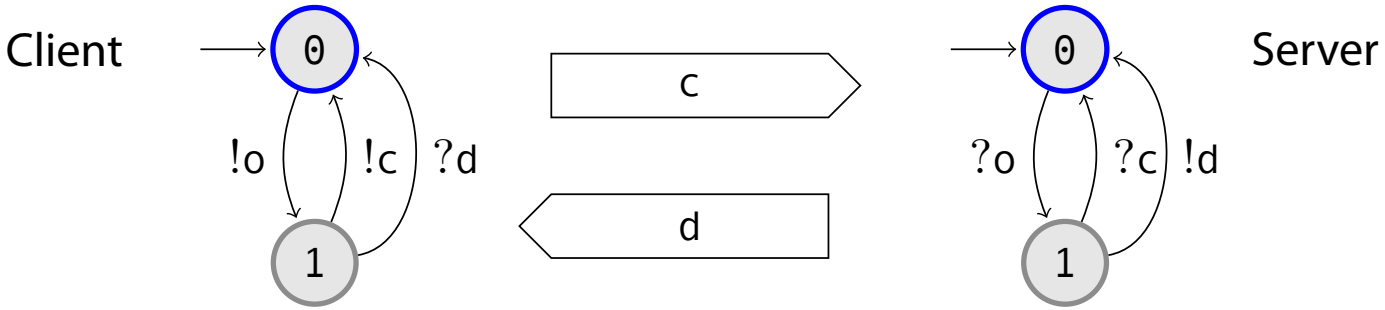
Example: Connection / Disconnection Protocol



$$\langle (0,0),(\varepsilon,\varepsilon)\rangle \xrightarrow{!o\ !c} \langle (0,0),(oc,\varepsilon)\rangle \xrightarrow{?o} \langle (0,1),(c,\varepsilon)\rangle \xrightarrow{!d}$$

FIFO Systems

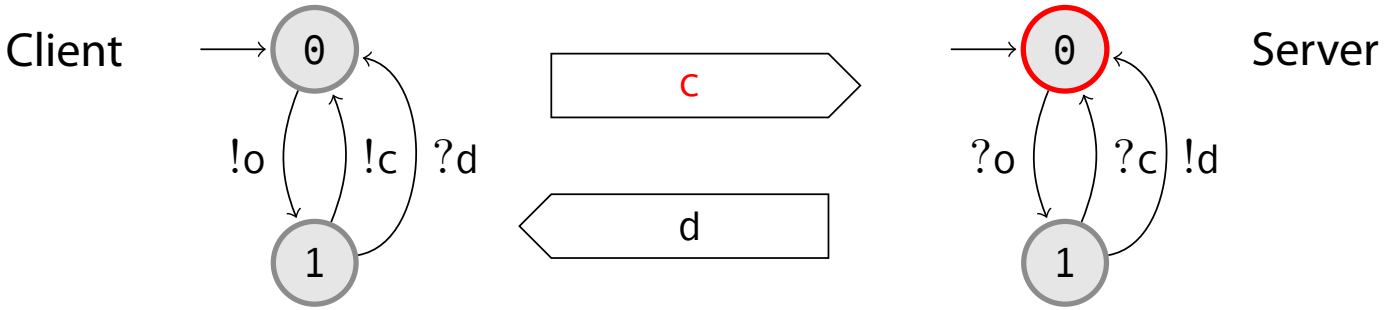
Example: Connection / Disconnection Protocol



$$\langle (0,0),(\varepsilon,\varepsilon)\rangle \xrightarrow{!o\ !c} \langle (0,0),(oc,\varepsilon)\rangle \xrightarrow{?o} \langle (0,1),(c,\varepsilon)\rangle \xrightarrow{!d} \langle (0,0),(c,d)\rangle$$

FIFO Systems

Example: Connection / Disconnection Protocol



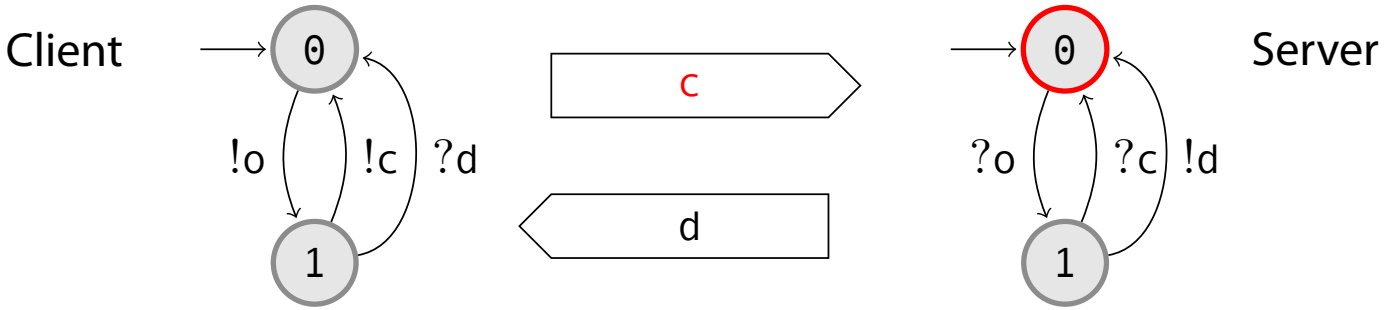
$$\langle (0,0),(\varepsilon,\varepsilon)\rangle \xrightarrow{!o\ !c} \langle (0,0),(oc,\varepsilon)\rangle \xrightarrow{?o} \langle (0,1),(c,\varepsilon)\rangle \xrightarrow{!d} \langle (0,0),(c,d)\rangle$$

↯ (local) deadlock ↯

unspecified reception

FIFO Systems

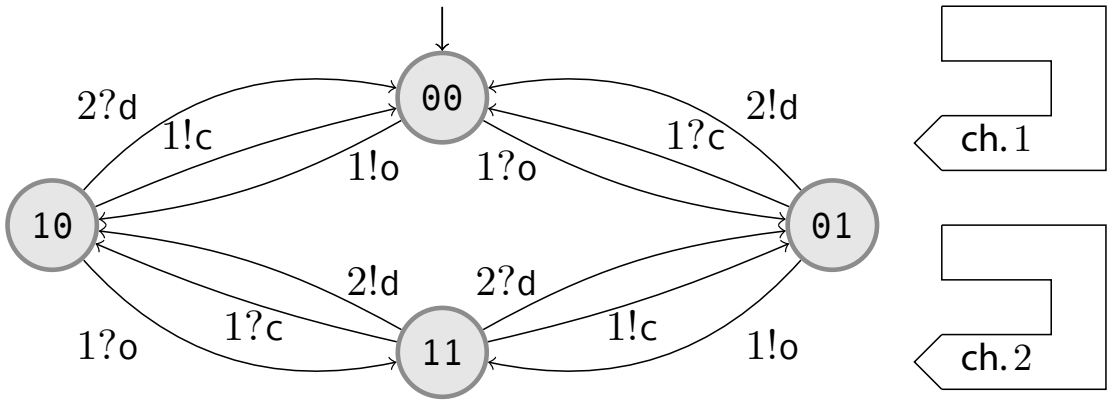
Example: Connection / Disconnection Protocol



$$\langle (0,0),(\varepsilon,\varepsilon)\rangle \xrightarrow{!o\ !c} \langle (0,0),(oc,\varepsilon)\rangle \xrightarrow{?o} \langle (0,1),(c,\varepsilon)\rangle \xrightarrow{!d} \langle (0,0),(c,d)\rangle$$

- let $Init = \{(0,0,(\varepsilon,\varepsilon)\}$
- let $Bad = S_1 \times \{0\} \times c \cdot M^* \times M^*$
- verify safety: is *Bad* not reachable from *Init* ?

FIFO Automata
Product Automaton



- asynchronous product automaton
- note: channels need not be point-to-point
- used as our point of departure for verification

Verification of FIFO Automata

- Turing-powerful model
  - reachability is undecidable [Brand&Zafiropulo '83]
- Decidable sub-classes
  - Bounded channels
  - Lossy channels [Abdulla&Jonsson '96]
  - Half-duplex communication [Cécé&Finkel '05]
- Approximation of the reachability set
  - ↓ Acceleration (may not terminate) [Boigelot&Godefroid '99]
  - ↑ Widening (may be inconclusive) [Le Gall&… '06]
- How about CEGAR? [Clarke&… '03]
  - Successfull approach for software model-checking

Verification of FIFO Automata

- Turing-powerful model
  - reachability is undecidable [Brand&Zafiropulo '83]
- Decidable sub-classes
  - Bounded channels
  - Lossy channels [Abdulla&Jonsson '96]
  - Half-duplex communication [Cécé&Finkel '05]
- Approximation of the reachability set
  - ↓ Acceleration (may not terminate) [Boigelot&Godefroid '99]
  - ↑ Widening (may be inconclusive) [Le Gall&… '06]
- How about CEGAR? [Clarke&… '03]
  - Successfull approach for software model-checking

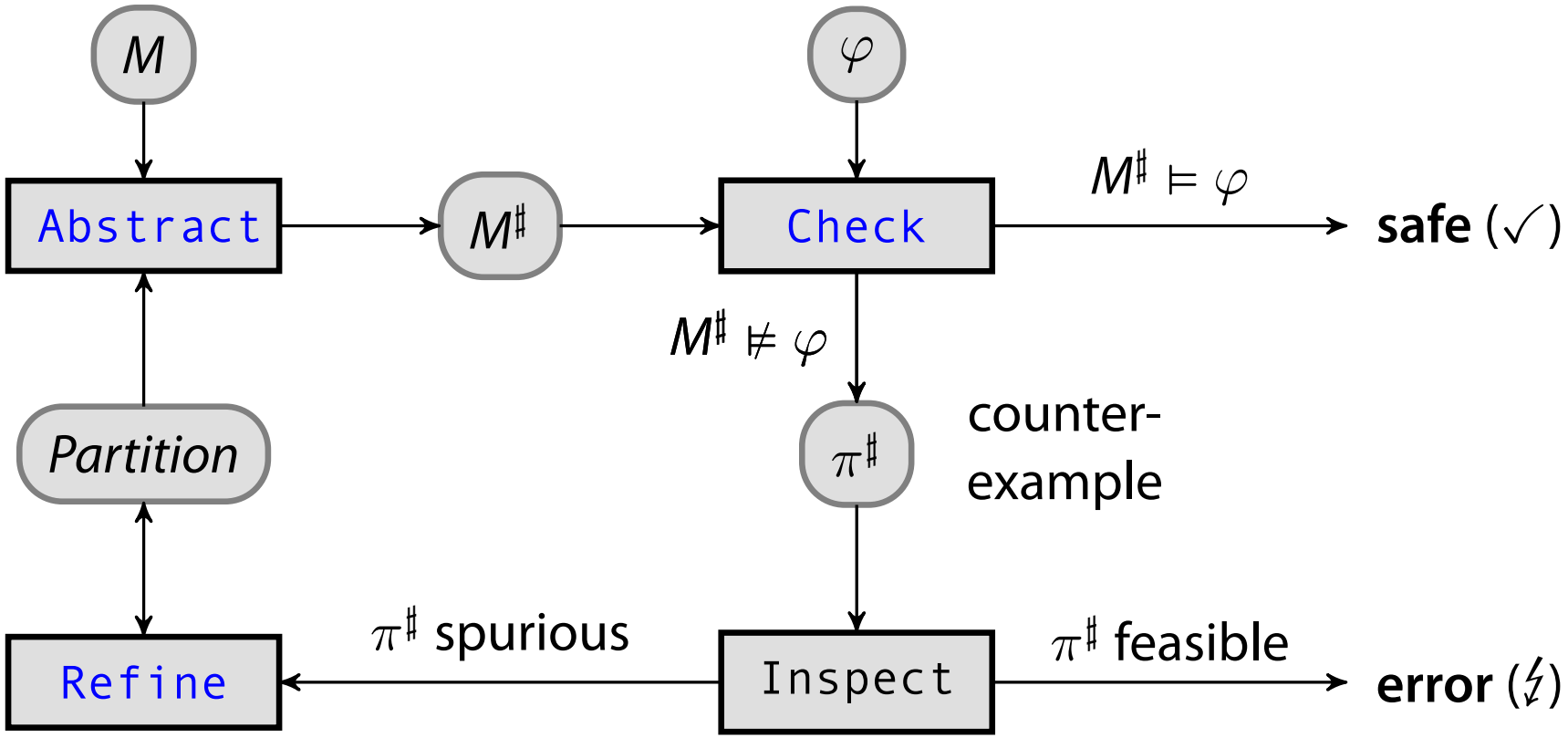
Verification of FIFO Automata

- Turing-powerful model
  - reachability is undecidable [Brand&Zafiropulo '83]
- Decidable sub-classes
  - Bounded channels
  - Lossy channels [Abdulla&Jonsson '96]
  - Half-duplex communication [Cécé&Finkel '05]
- Approximation of the reachability set
  - ↓ Acceleration (may not terminate) [Boigelot&Godefroid '99]
  - ↑ Widening (may be inconclusive) [Le Gall&… '06]
- How about CEGAR? [Clarke&… '03]
  - Successfull approach for software model-checking

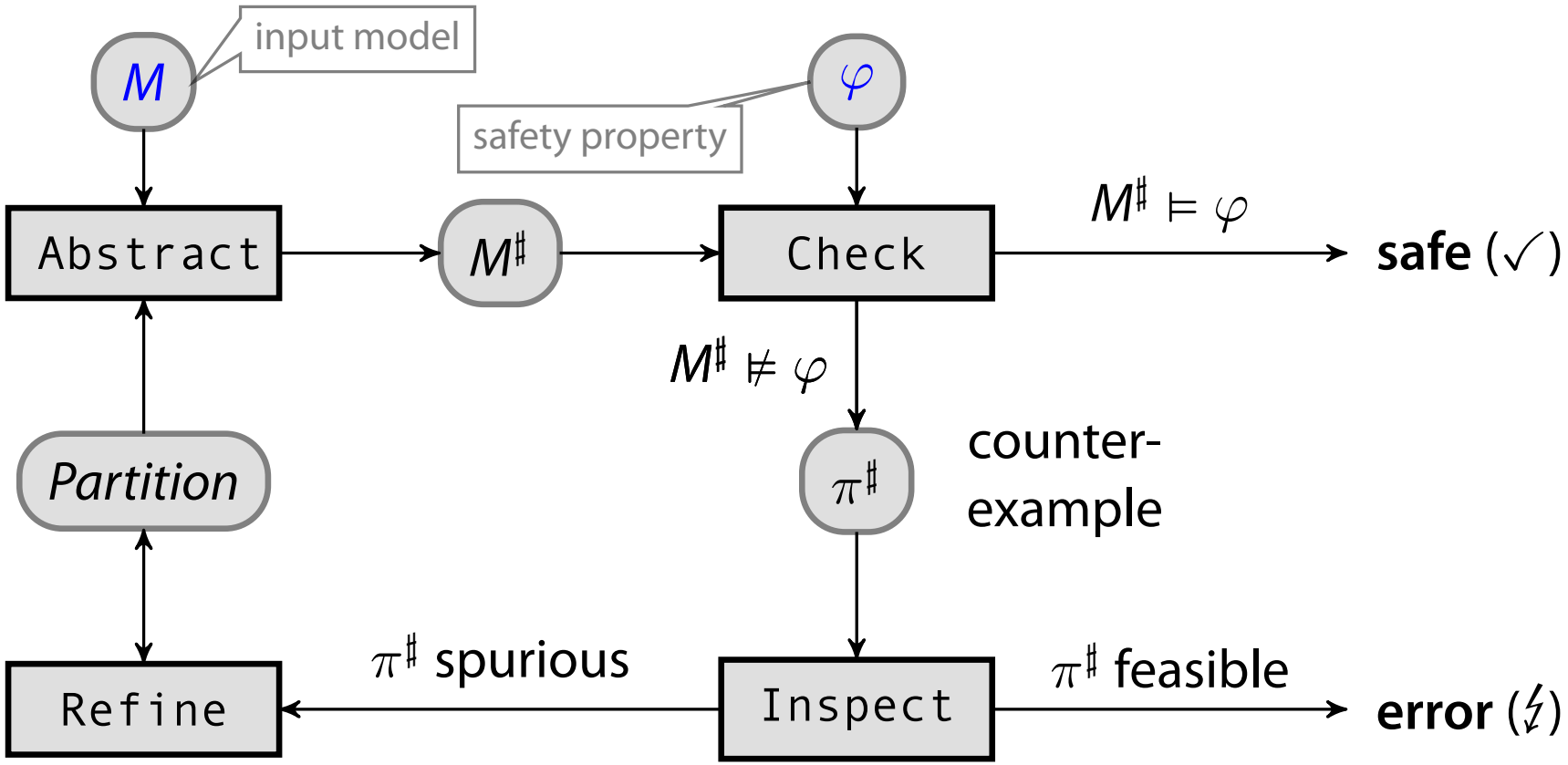
Verification of FIFO Automata

- Turing-powerful model
  - reachability is undecidable [Brand&Zafiropulo '83]
- Decidable sub-classes
  - Bounded channels
  - Lossy channels [Abdulla&Jonsson '96]
  - Half-duplex communication [Cécé&Finkel '05]
- Approximation of the reachability set
  - ↓ Acceleration (may not terminate) [Boigelot&Godefroid '99]
  - ↑ Widening (may be inconclusive) [Le Gall&… '06]
- How about CEGAR? [Clarke&… '03]
  - Successfull approach for software model-checking

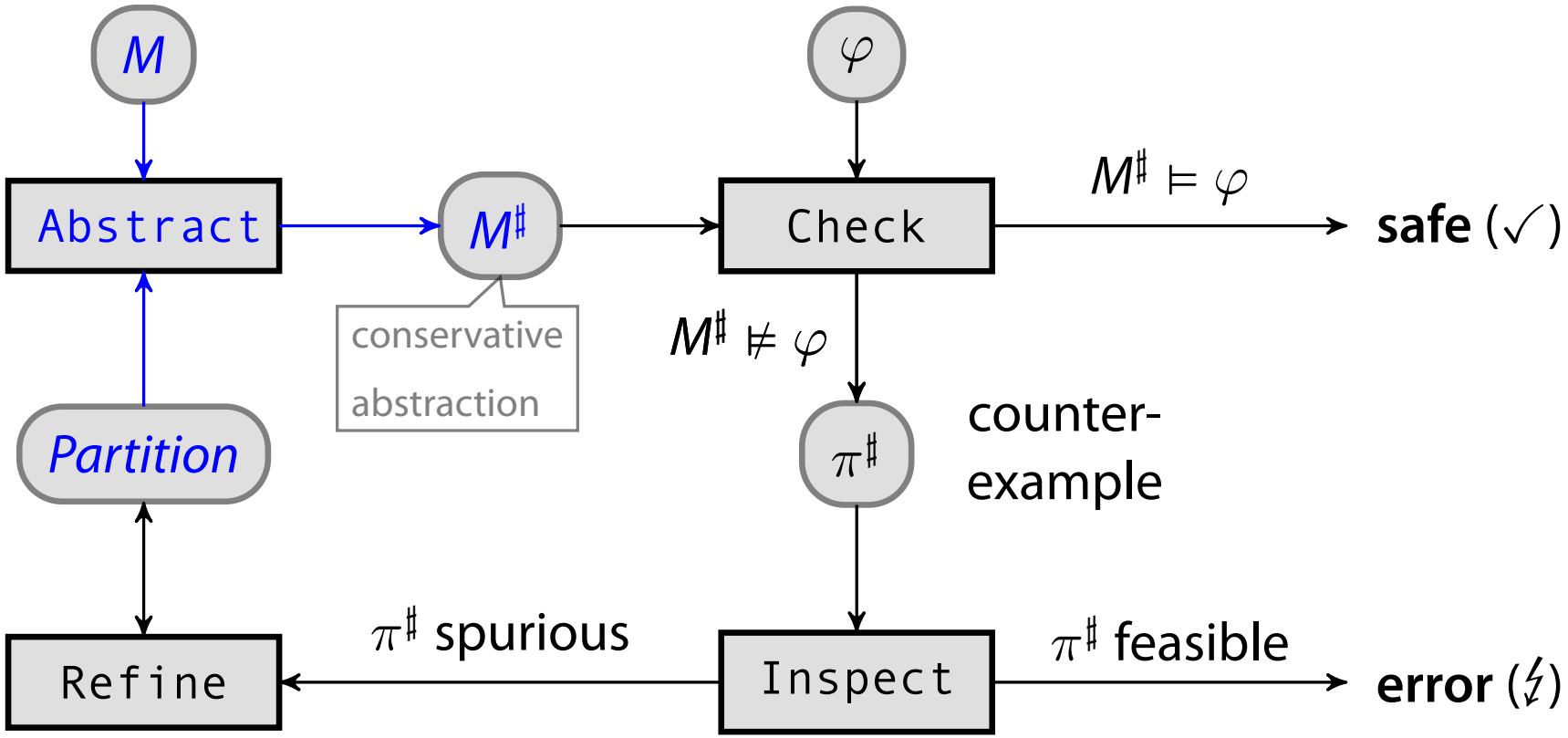
CEGAR
the general approach



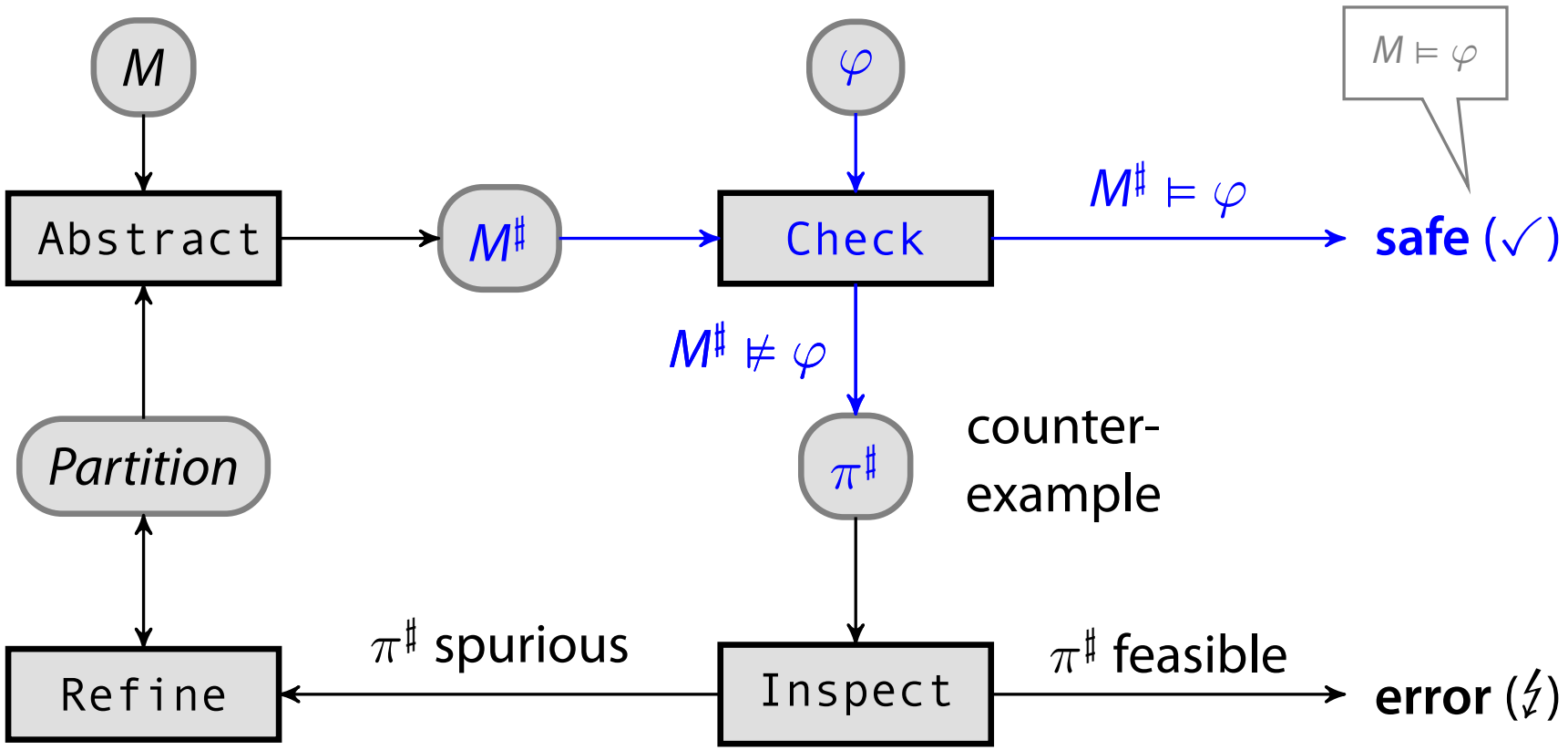
CEGAR
the general approach



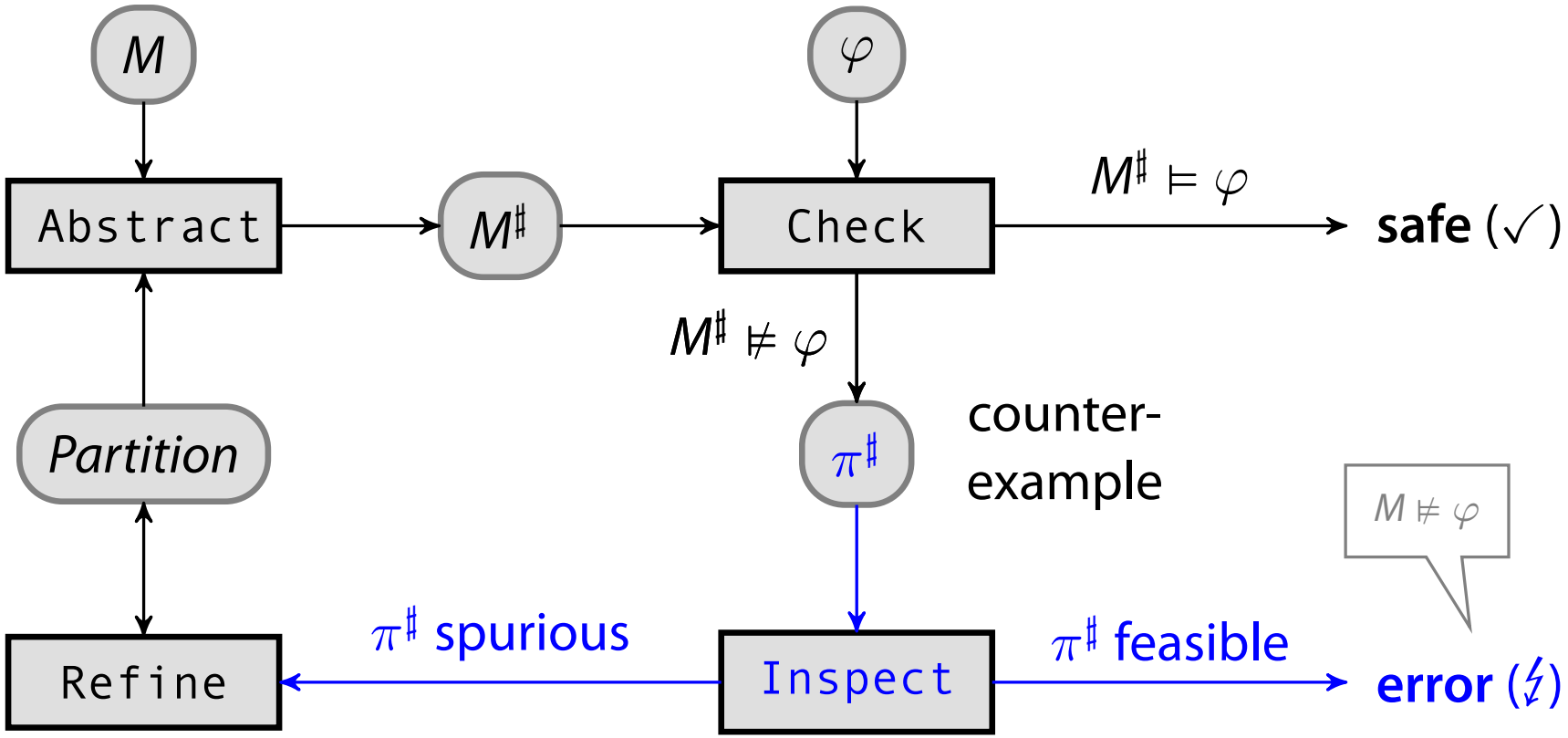
CEGAR
the general approach



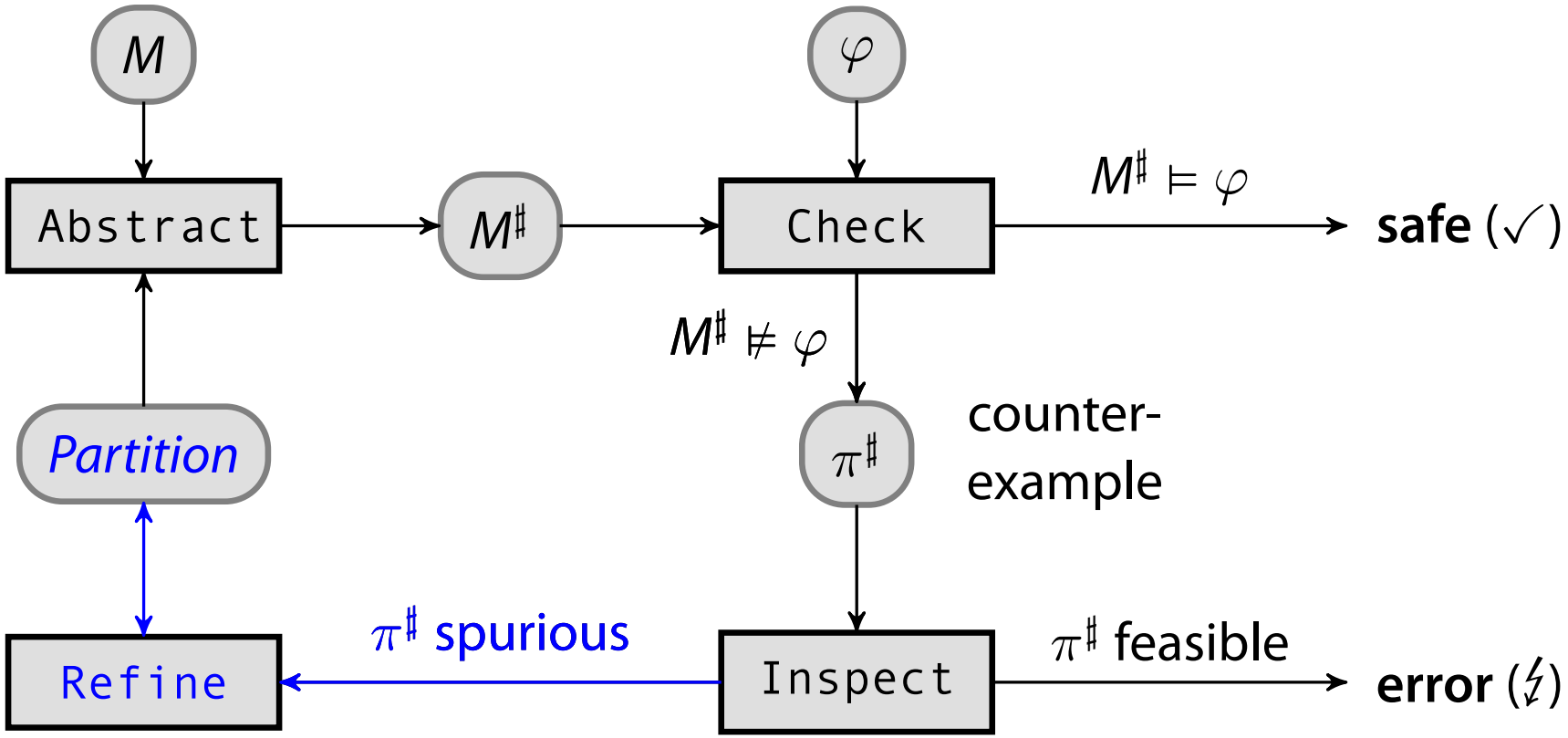
CEGAR
the general approach



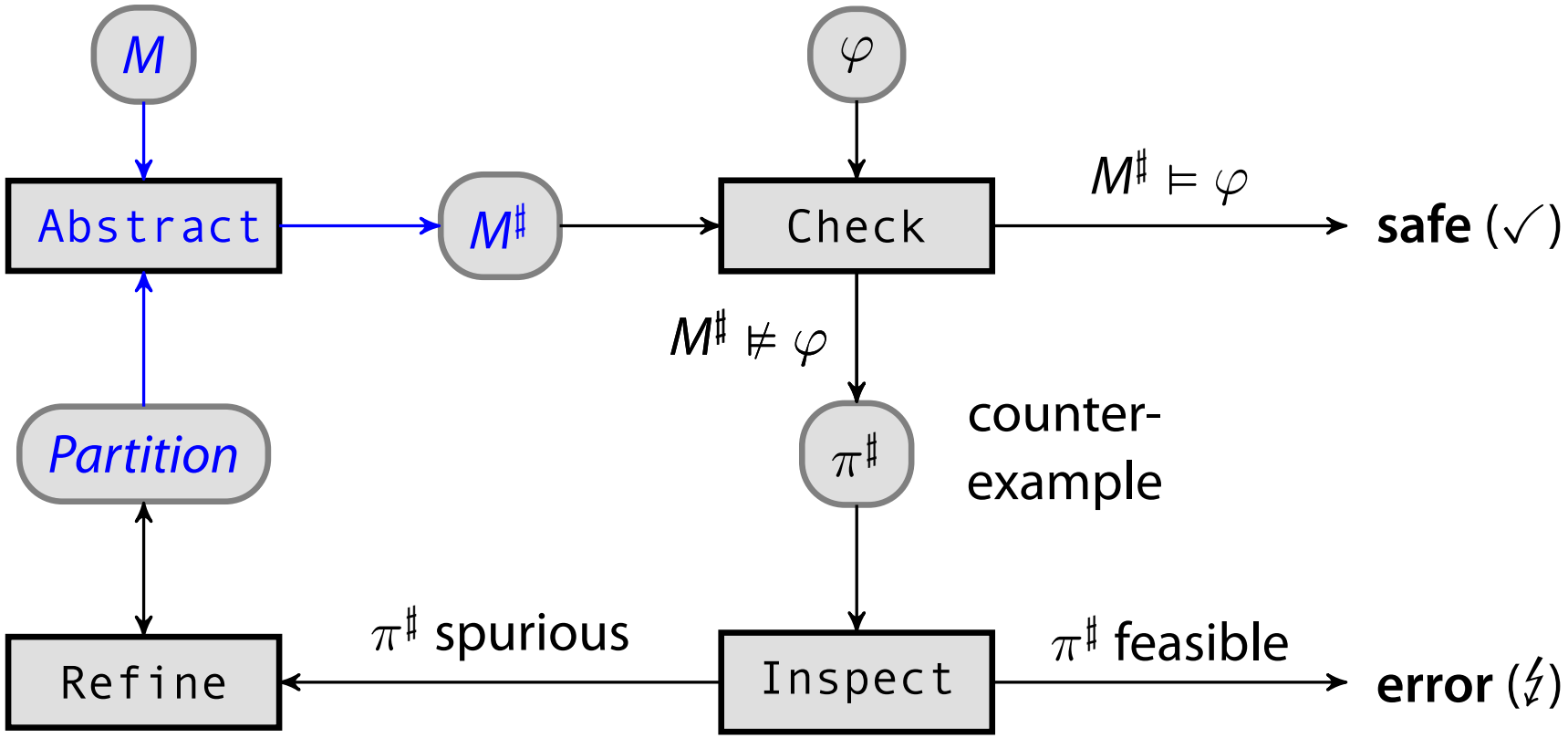
CEGAR
the general approach



CEGAR
the general approach

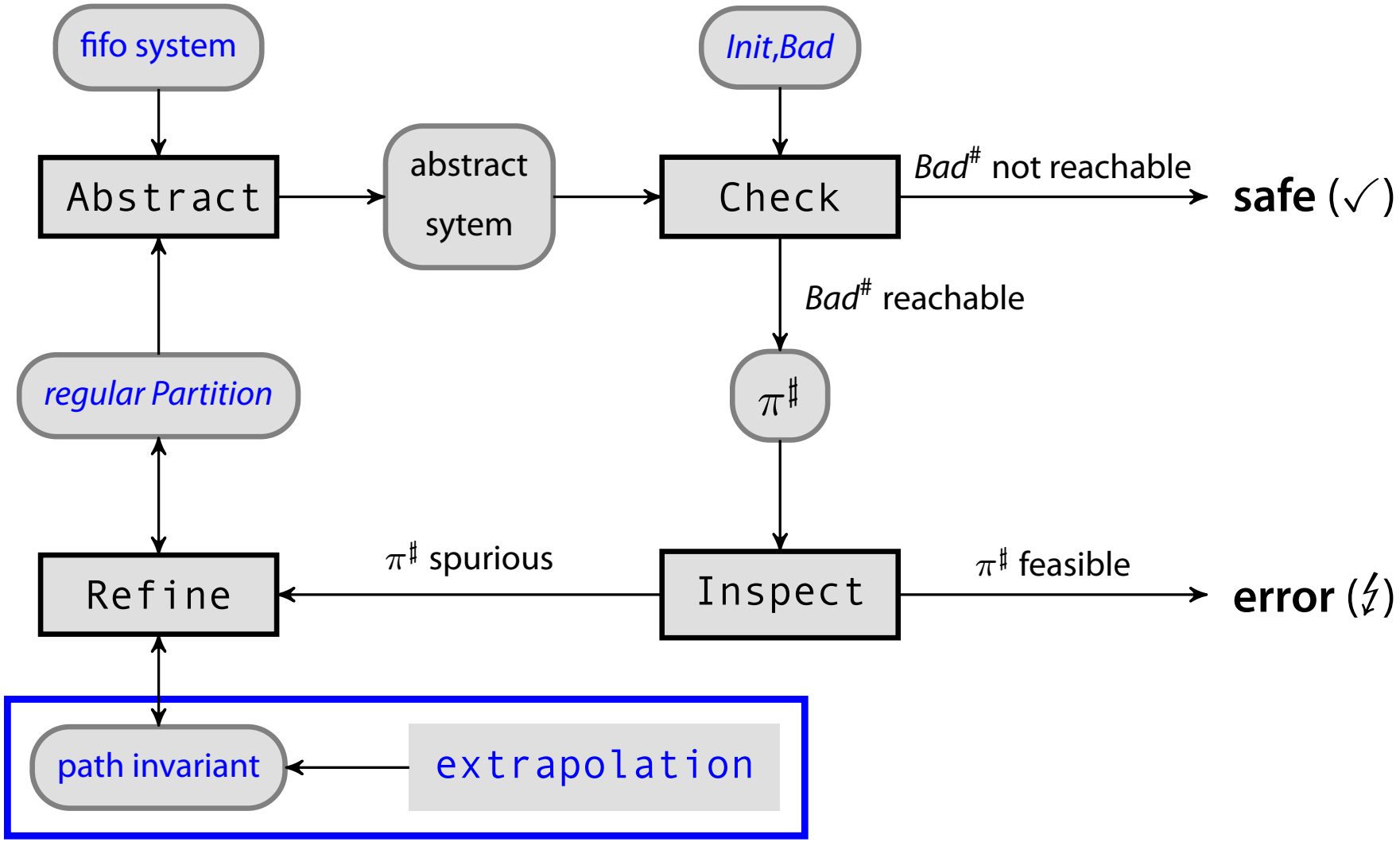


CEGAR
the general approach

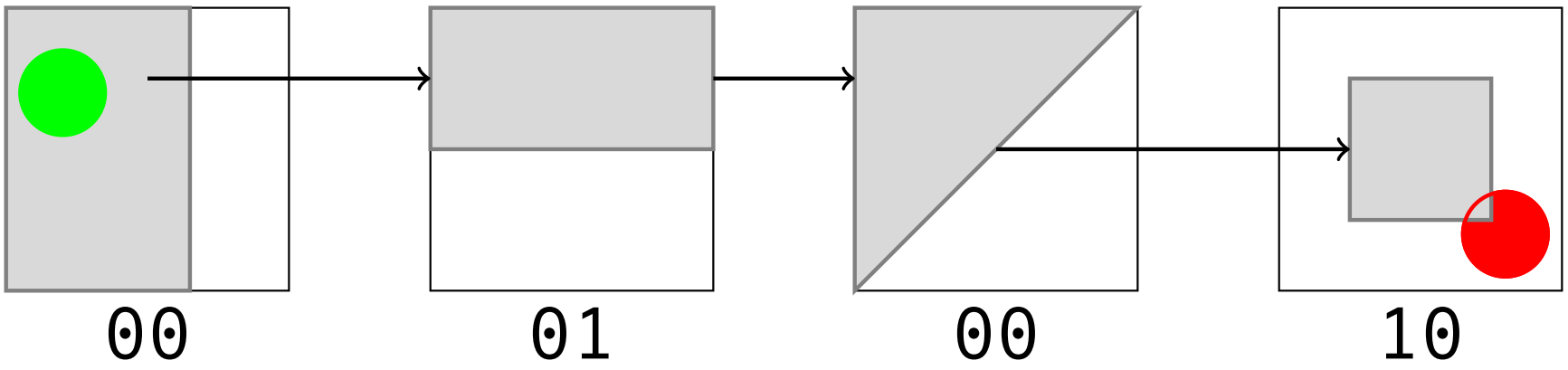


CEGAR

…adapted to our setting



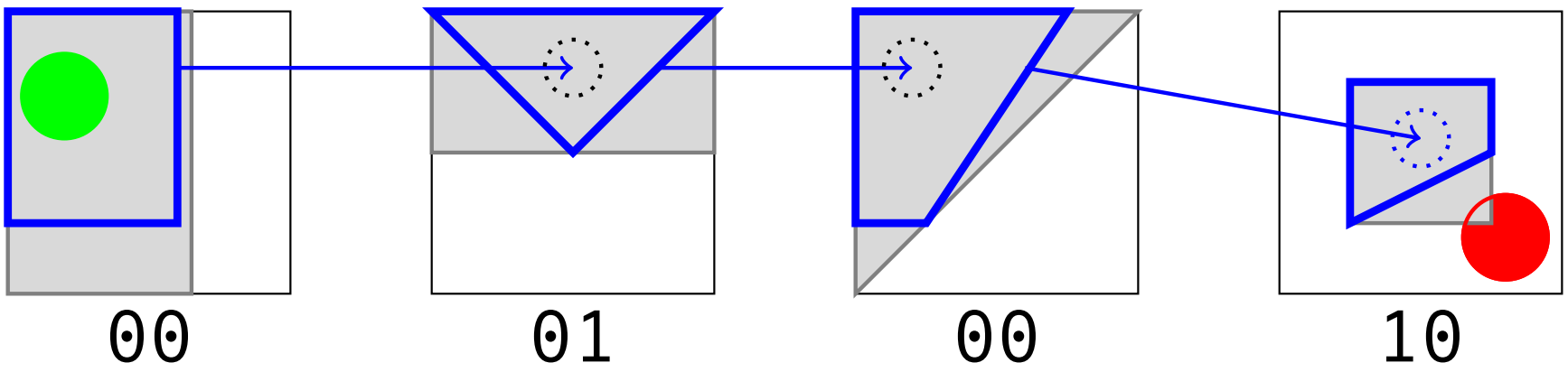
Path Invariants



start with spurious abstract counterexample

- initally includes *Init*
- reaches abstract state not disjoint with *Bad*

Path Invariants

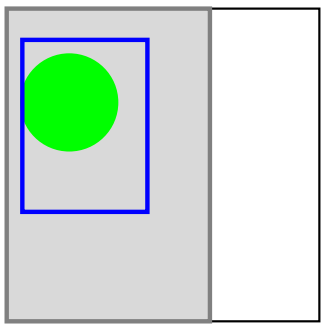


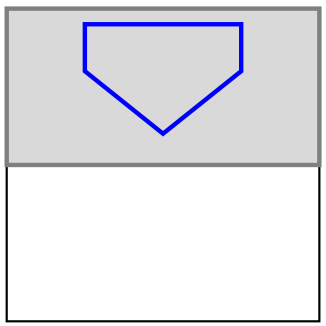
find path invariant

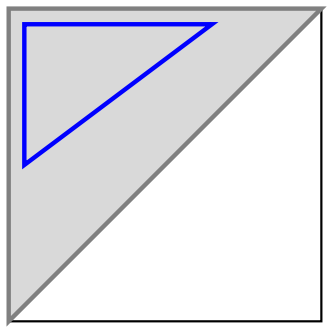
- initally includes *Init*
- closed under steps of original counterexample
- finally disjoint with *Bad*

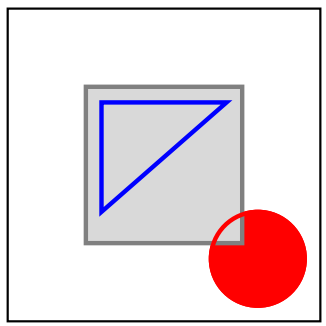
$L_0, \ldots, L_n$ such that
(i) $Init \subseteq L_0$
(ii) $post(L_i) \subseteq L_{i+1}$
(iii) $L_n \subseteq \overline{Bad}$

Path Invariants





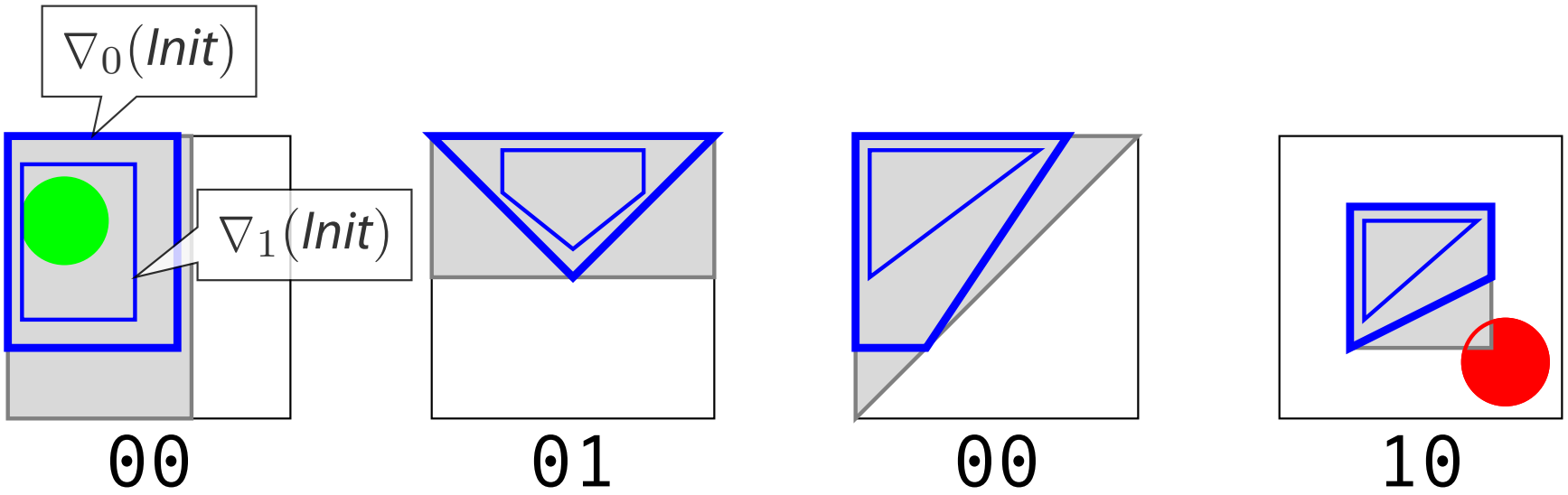




given a spurious counterexample, there is a family of path invariants

- there must be at least one
  as the counterexample was shown to be spurious

Path Invariants



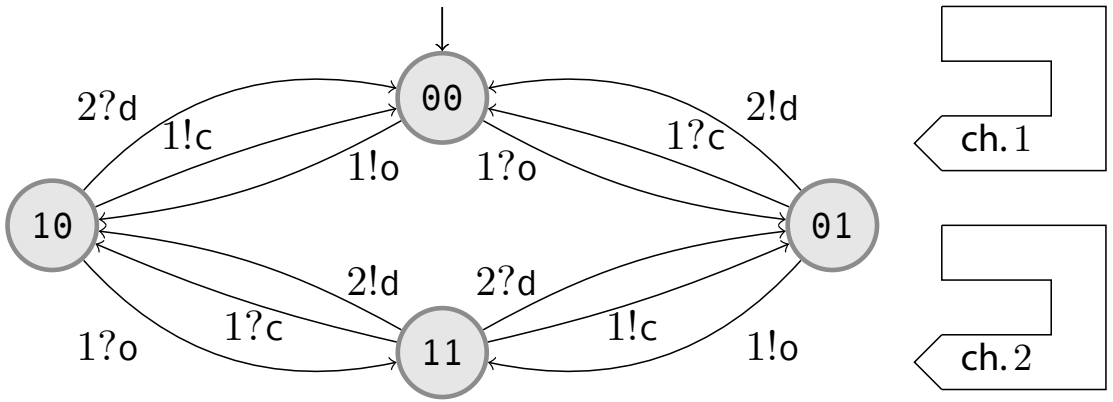
try to get most simple path invariant possible

- apply in each step parametrized extrapolation ∇
  - ∇_0 is maximal overapproximation
  - $\nabla_{k+1} \subseteq \nabla_k$
  - $\exists\, k_{max} : \nabla_{k_{max}} = id$

A Practical Example

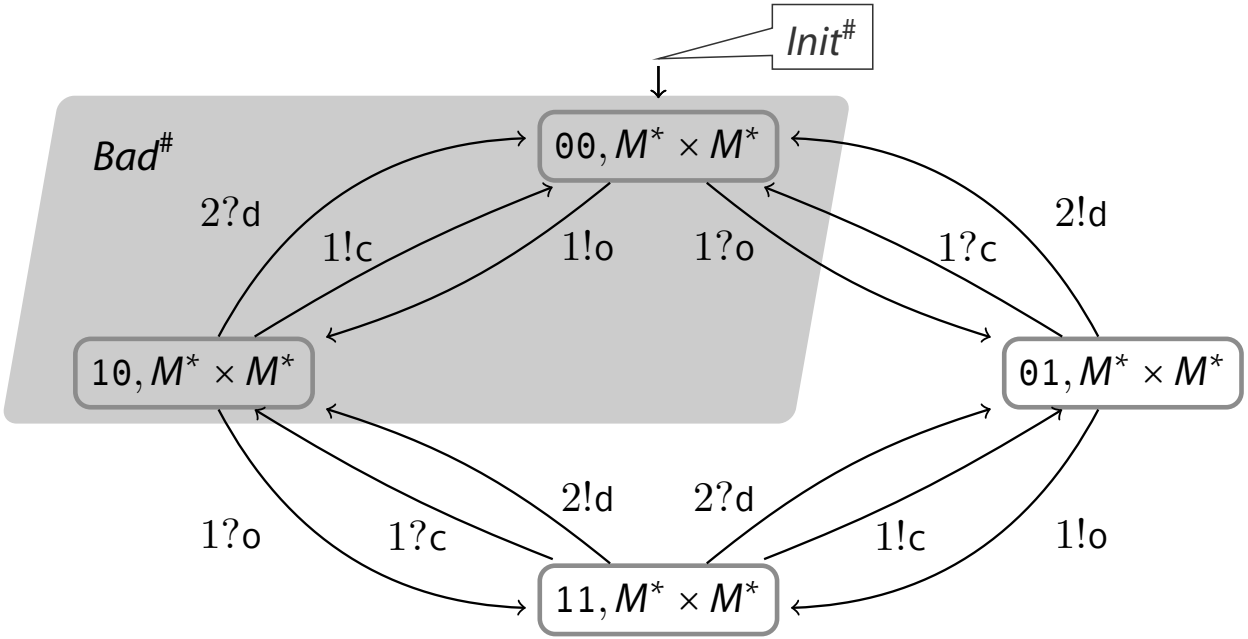
Safety Verification of the C/D-Protocol

Applying our Algorithm to the Connection / Disconnection Protocol
step 0 : basic abstraction

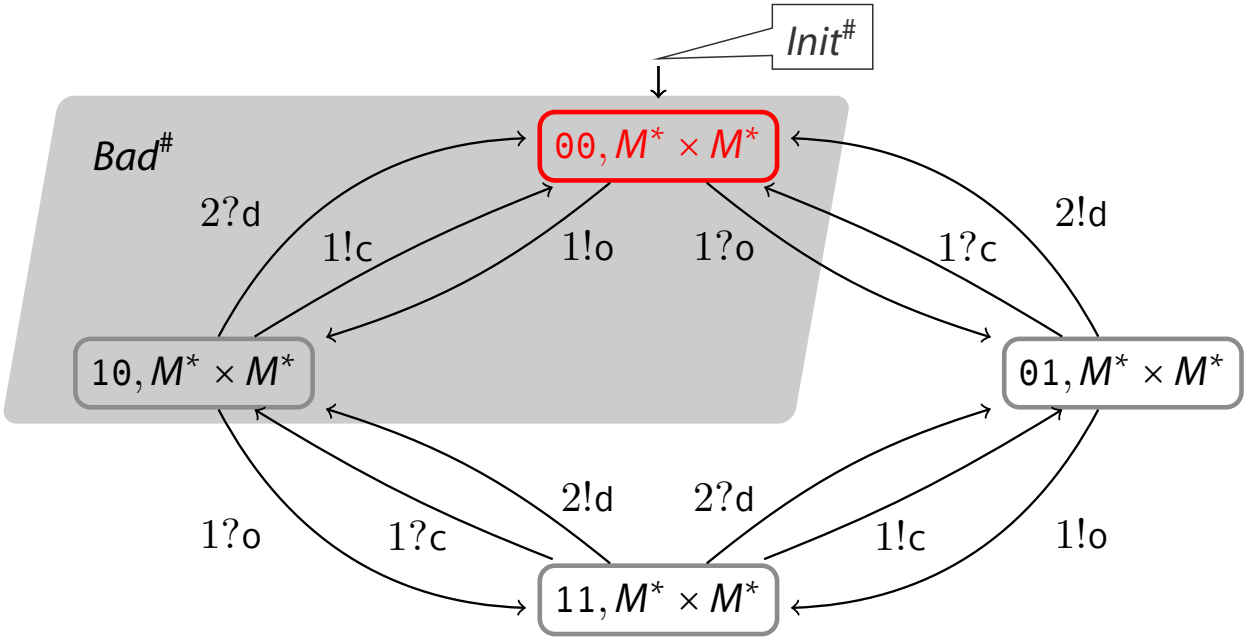


use partition abstraction

- abstract configurations $\langle\!\langle (q_1, q_2), L_1 \times L_2 \rangle\!\rangle$
- L_1, L_2 are regular languages over the message alphabet M
- abstract transitions via existential lift…



- initial partition: $M^* \times M^*$
- calculate $Init^{\#}$ and $Bad^{\#}$
- is set of abstract configs $Bad^{\#}$ reachable from $Init^{\#}$?



- initial partition: $M^* \times M^*$
- calculate $Init^{\#}$ and $Bad^{\#}$
- is set of abstract configs $Bad^{\#}$ reachable from $Init^{\#}$?
- find simple counterexample: $\langle\!\langle \mathtt{00}, M^* \times M^* \rangle\!\rangle$

- found counterexample:

$$\langle\!\langle \mathtt{00}, M^* \times M^* \rangle\!\rangle$$

- counterexample is spurious:

$$\langle \mathtt{00}, (\varepsilon, \varepsilon) \rangle \notin \mathit{Bad}$$

- path invariant:

$$(\varepsilon \times \varepsilon)$$

- because

$$\mathit{Init} \subseteq \langle\!\langle \mathtt{00}, \varepsilon \times \varepsilon \rangle\!\rangle \subseteq \overline{\mathit{Bad}}$$

initial condition final condition

- found counterexample:

$$\langle\!\langle \mathtt{00}, M^* \times M^* \rangle\!\rangle$$

- counterexample is spurious:

$$\langle \mathtt{00}, (\varepsilon, \varepsilon) \rangle \notin \mathit{Bad}$$

- path invariant:

$$(\varepsilon \times \varepsilon)$$

- because

$$\mathit{Init} \quad \subseteq \quad \langle\!\langle \mathtt{00}, \varepsilon \times \varepsilon \rangle\!\rangle \quad \subseteq \quad \overline{\mathit{Bad}}$$

initial condition final condition

- found counterexample:

$$\langle\!\langle \mathtt{00}, M^* \times M^* \rangle\!\rangle$$

- counterexample is spurious:

$$\langle \mathtt{00}, (\varepsilon, \varepsilon) \rangle \notin \mathit{Bad}$$

- path invariant:

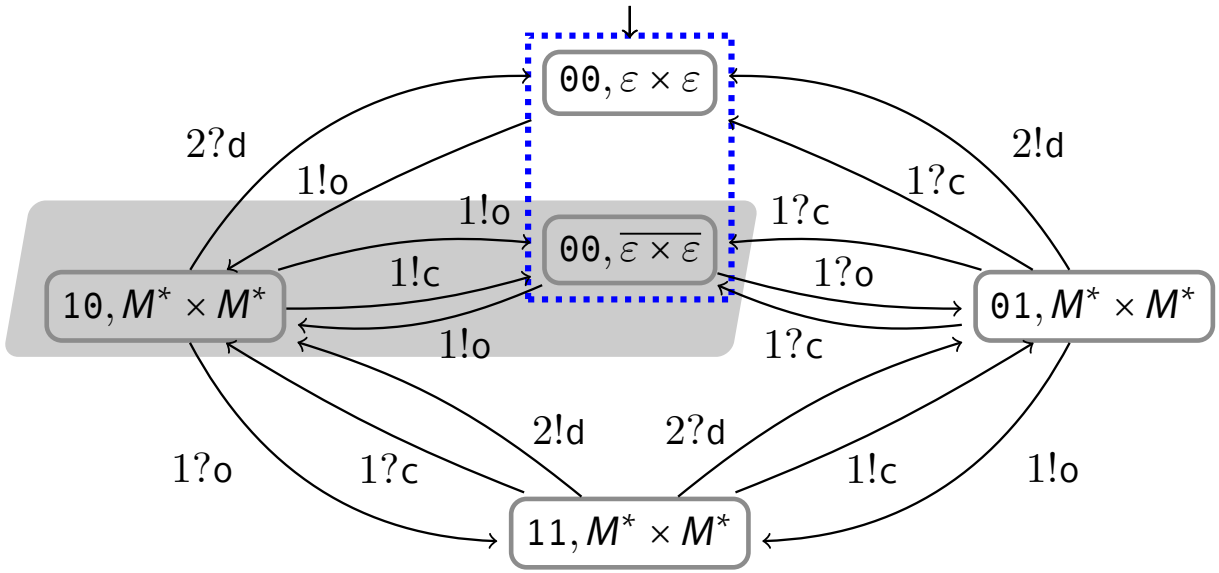
$$(\varepsilon \times \varepsilon)$$

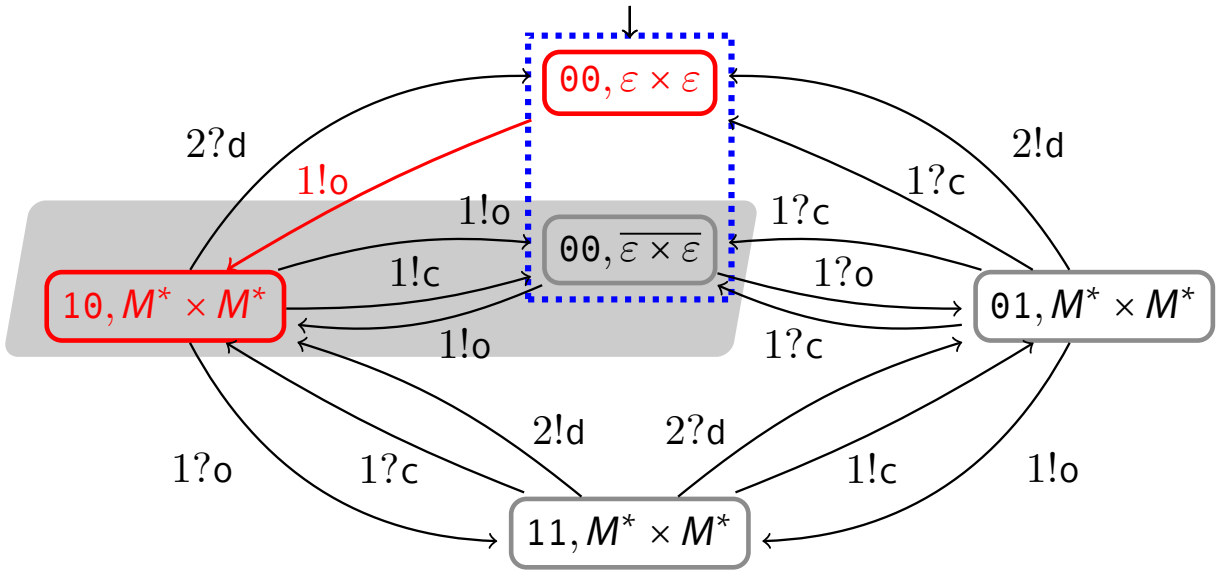
- because

$$\mathit{Init} \quad \subseteq \quad \langle\!\langle \mathtt{00}, \varepsilon \times \varepsilon \rangle\!\rangle \quad \subseteq \quad \overline{\mathit{Bad}}$$

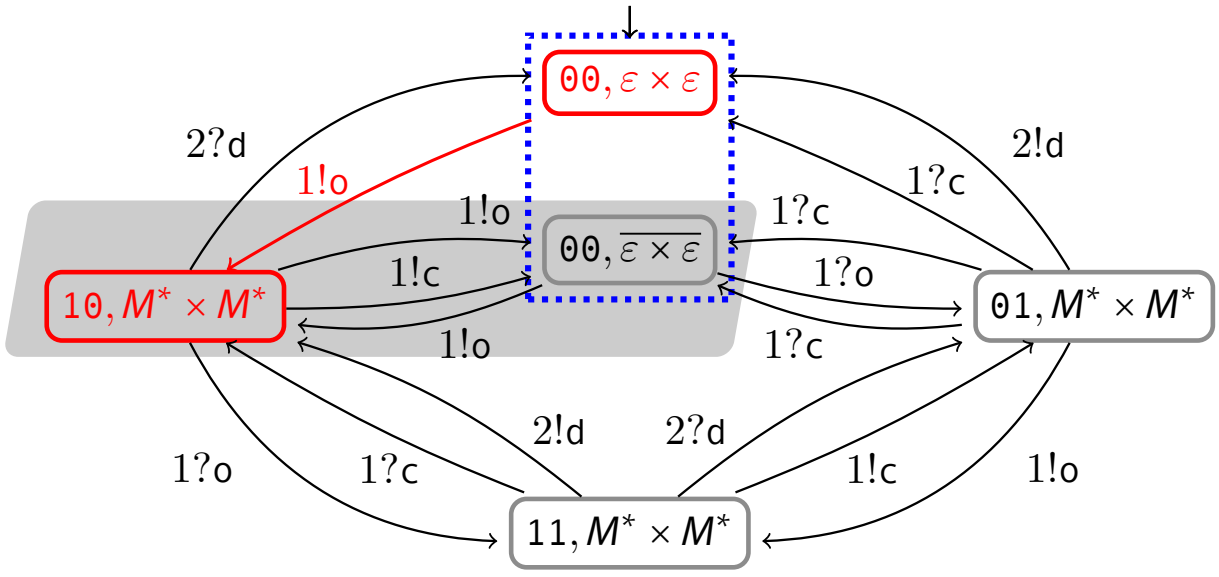
initial condition final condition

step 2 : counterexample

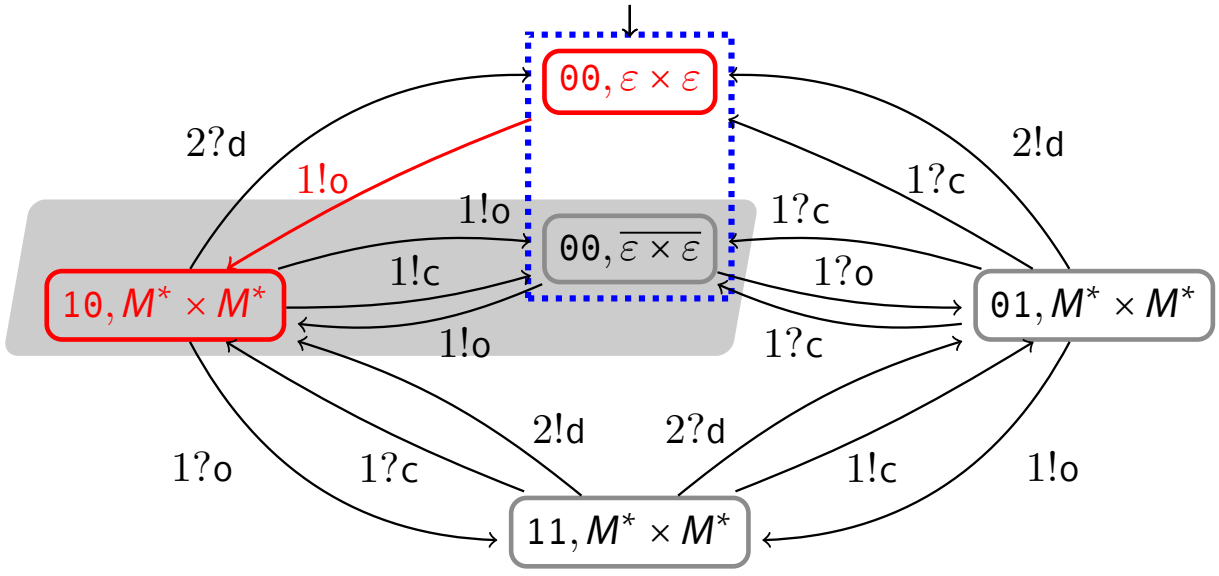




- find counterexample: $\langle\!\langle \mathtt{00}, \varepsilon \times \varepsilon \rangle\!\rangle \xrightarrow{!o} \langle\!\langle \mathtt{10}, M^* \times M^* \rangle\!\rangle$



- find counterexample: $\langle\!\langle \mathtt{00}, \varepsilon \times \varepsilon \rangle\!\rangle \xrightarrow{!o} \langle\!\langle \mathtt{10}, M^* \times M^* \rangle\!\rangle$
- …is spurious as $\langle \mathtt{00}, (\varepsilon, \varepsilon) \rangle \xrightarrow{!o} \langle \mathtt{10}, (o, \varepsilon) \rangle \notin \mathit{Bad}$

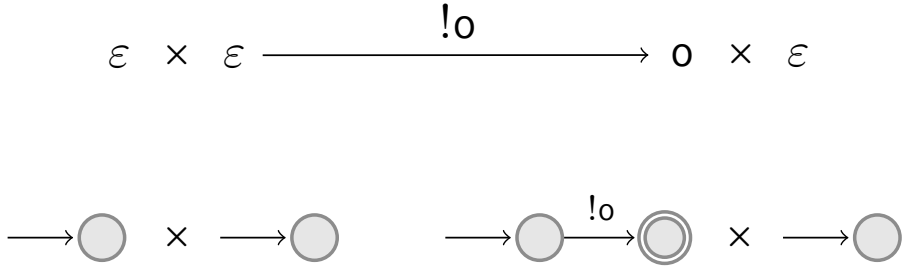


- find counterexample: $\langle\!\langle \mathtt{00}, \varepsilon \times \varepsilon \rangle\!\rangle \xrightarrow{!o} \langle\!\langle \mathtt{10}, M^* \times M^* \rangle\!\rangle$
- …is spurious as $\langle \mathtt{00}, (\varepsilon, \varepsilon) \rangle \xrightarrow{!o} \langle \mathtt{10}, (o, \varepsilon) \rangle \notin \mathit{Bad}$
- (simple, strongest) path invariant $\varepsilon \times \varepsilon \;,\; o \times \varepsilon$

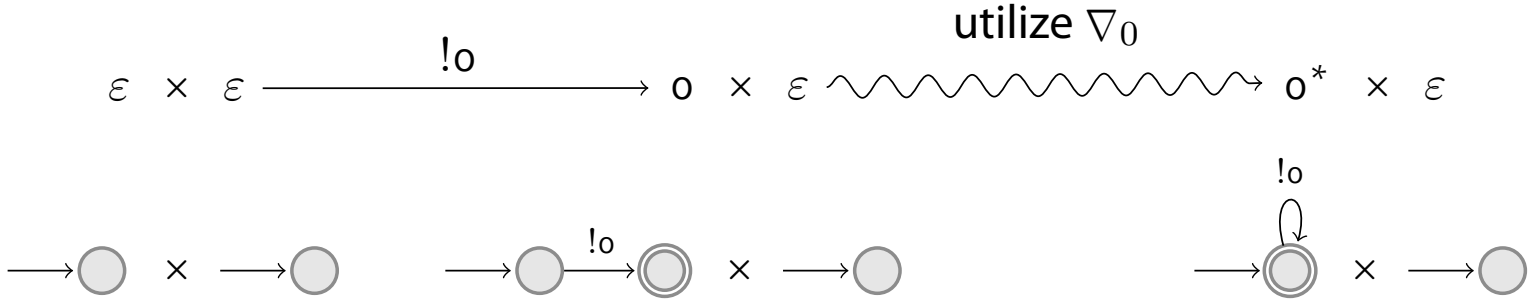
- apply extrapolation

$$\varepsilon \times \varepsilon$$

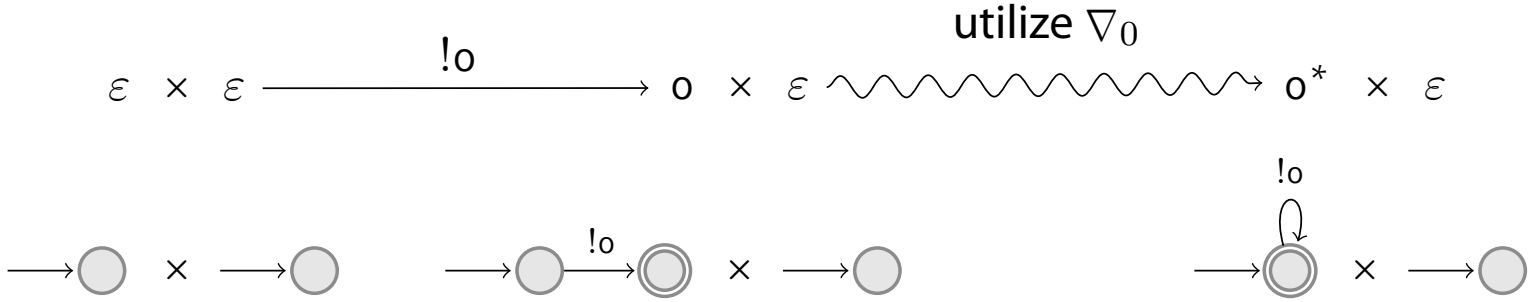
- apply extrapolation



- apply extrapolation

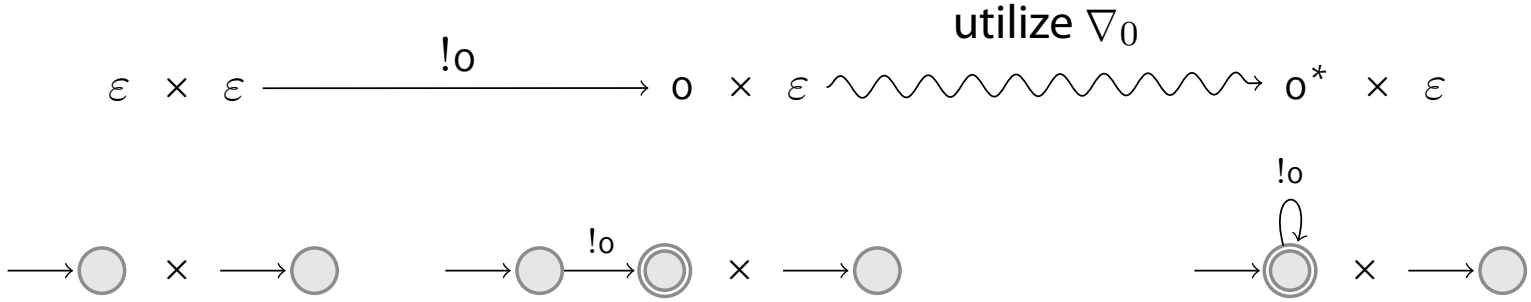


- apply extrapolation

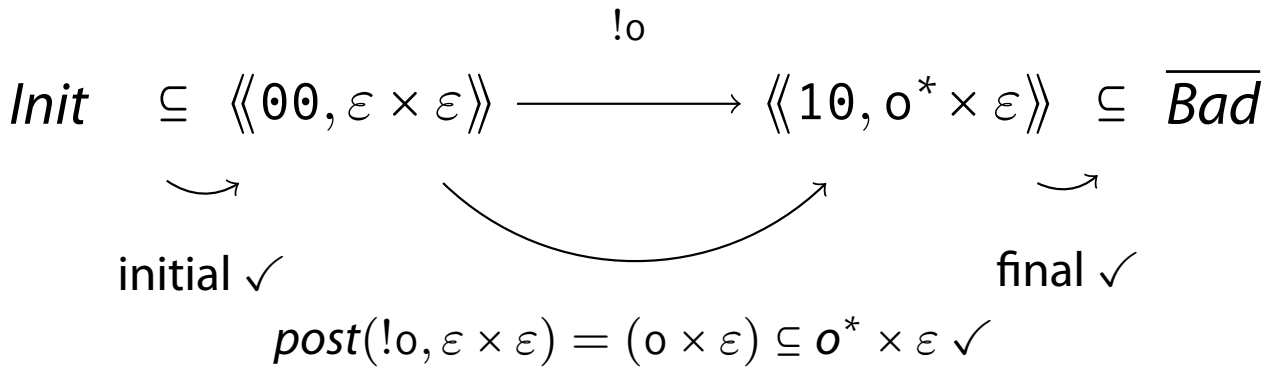


- extrapolated path invariant $\varepsilon \times \varepsilon$, o* $\times \varepsilon$

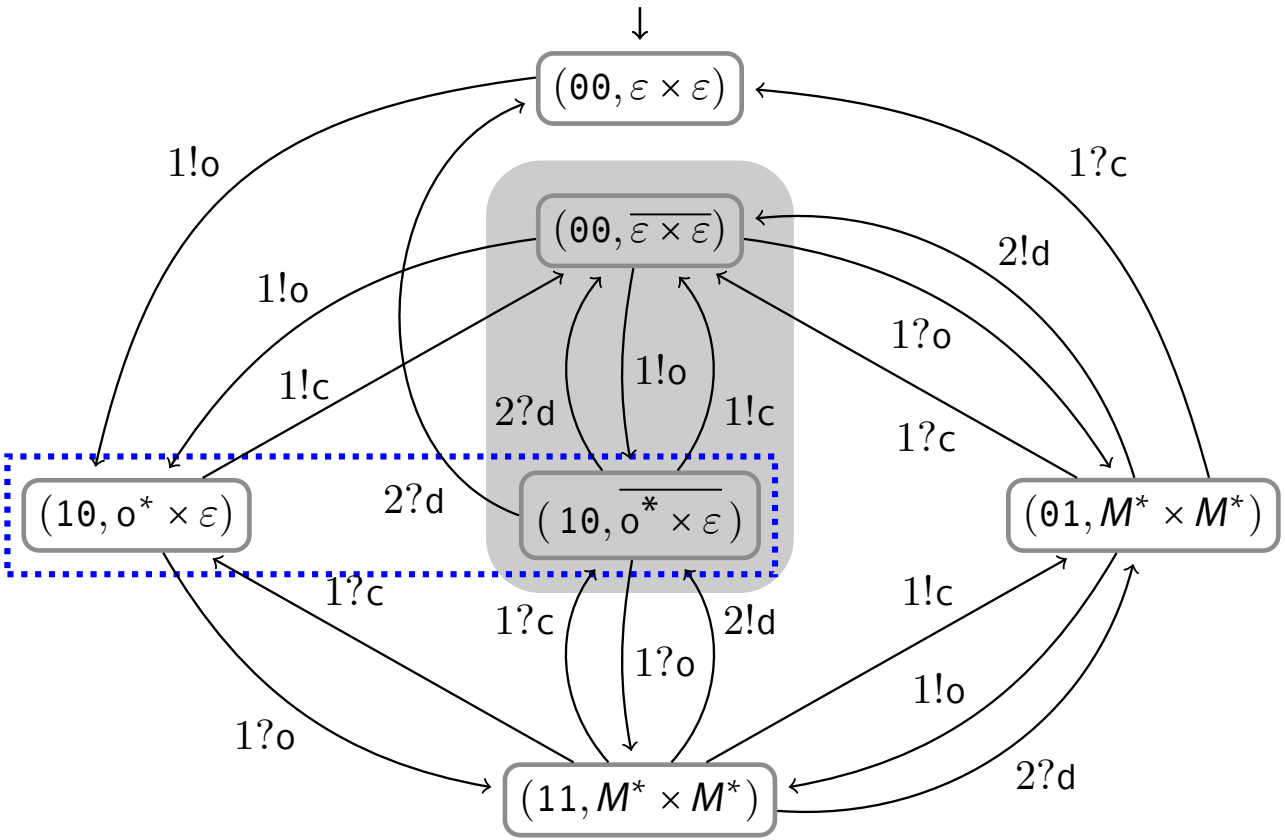
- apply extrapolation

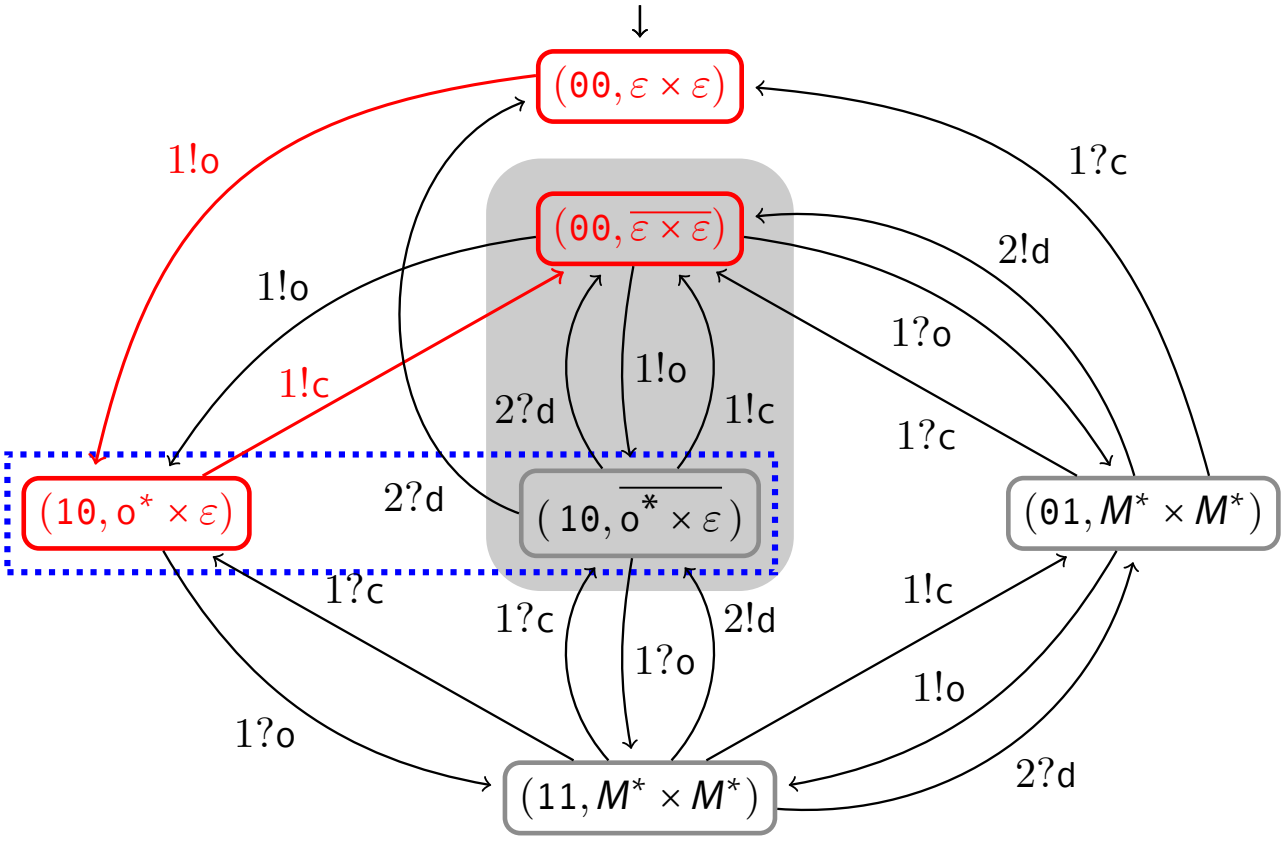


- extrapolated path invariant $\varepsilon \times \varepsilon$, $o^* \times \varepsilon$



step 3 : counterexample



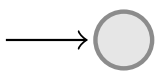


- (spurious) counterexample:

$$\langle\!\langle 00, \varepsilon \times \varepsilon * \rangle\!\rangle \xrightarrow{!o} \langle\!\langle 10, o^* \times \varepsilon \rangle\!\rangle \xrightarrow{!c} \langle\!\langle 00, \overline{\varepsilon \times \varepsilon} \rangle\!\rangle$$

- path invariant via extrapolation: utilize ∇_0

$$\varepsilon \left(\times \ \varepsilon \right)$$



- path invariant via extrapolation: utilize ∇_0

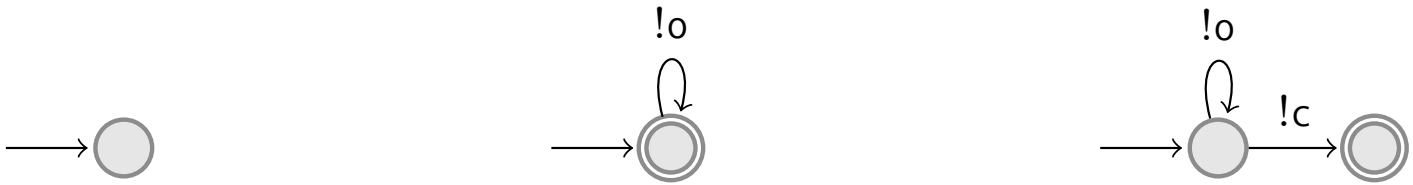
$$\varepsilon \left(\times \ \varepsilon\right) \xrightarrow{\text{!o}}$$

- path invariant via extrapolation: utilize ∇_0

$$\varepsilon\ (\times\ \varepsilon) \xrightarrow{!o} o^*\ (\times\ \varepsilon)$$

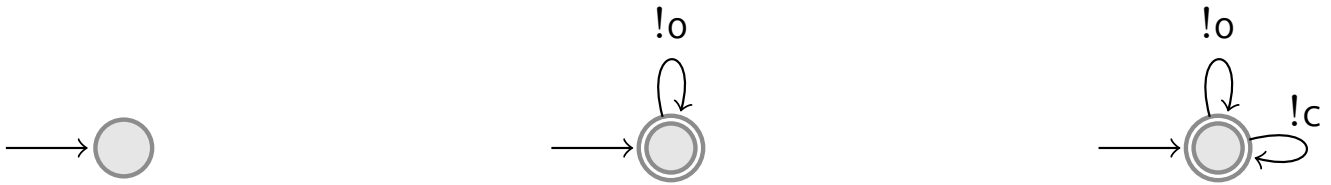
- path invariant via extrapolation: utilize ∇_0

$$\varepsilon\ (\times\ \varepsilon) \xrightarrow{\text{!o}} \text{o}^*\ (\times\ \varepsilon) \xrightarrow{\text{!c}}$$



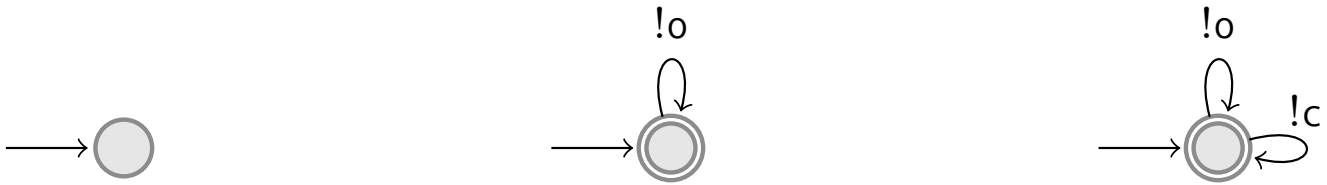
- path invariant via extrapolation: utilize ∇_0

$$\varepsilon\ (\times\ \varepsilon) \xrightarrow{!o} o^*\ (\times\ \varepsilon) \xrightarrow{!c} \{o, c\}^*\ (\times\ \varepsilon)$$



- path invariant via extrapolation: utilize ∇_0

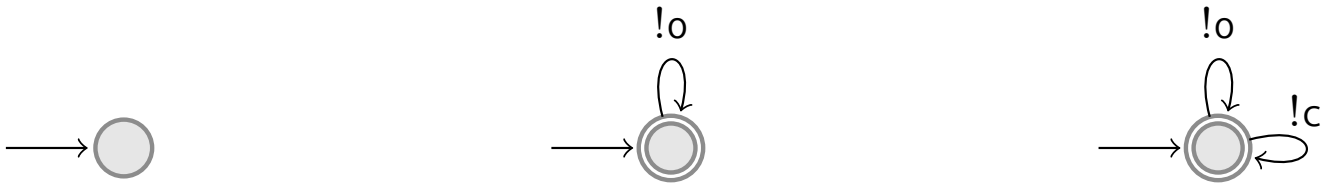
$$\varepsilon\ (\times\ \varepsilon) \xrightarrow{!o} o^*\ (\times\ \varepsilon) \xrightarrow{!c} \{o, c\}^*\ (\times\ \varepsilon)$$ *Bad*



- path invariant via extrapolation: utilize ∇_0

$$\varepsilon\,(\times\ \varepsilon) \xrightarrow{\text{!o}} \text{o}^*\,(\times\ \varepsilon) \xrightarrow{\text{!c}} \{\text{o},\text{c}\}^*\,(\times\ \varepsilon)$$

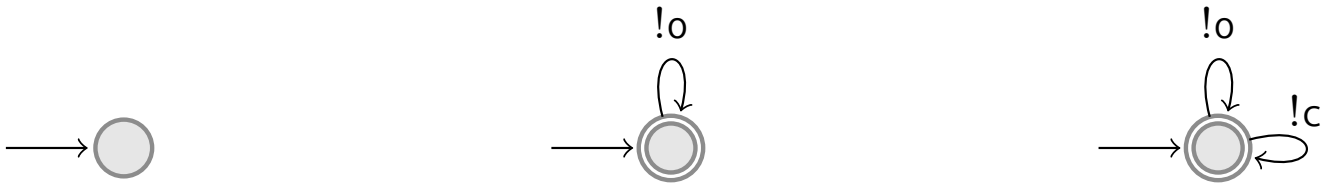
Bad



- use finer extrapolation: ∇_1

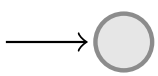
- path invariant via extrapolation: utilize ∇_0

$$\varepsilon\,(\times\ \varepsilon) \xrightarrow{!o} o^*\,(\times\ \varepsilon) \xrightarrow{!c} \{o,c\}^*\,(\times\ \varepsilon)$$ *Bad*



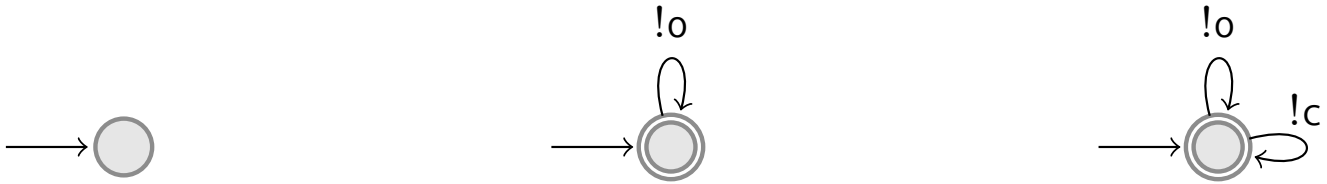
- use finer extrapolation: ∇_1

$$\varepsilon\,(\times\ \varepsilon)$$



- path invariant via extrapolation: utilize ∇_0

$$\varepsilon\ (\times\ \varepsilon) \xrightarrow{!o} o^*\ (\times\ \varepsilon) \xrightarrow{!c} \{o, c\}^*\ (\times\ \varepsilon)$$ *Bad*

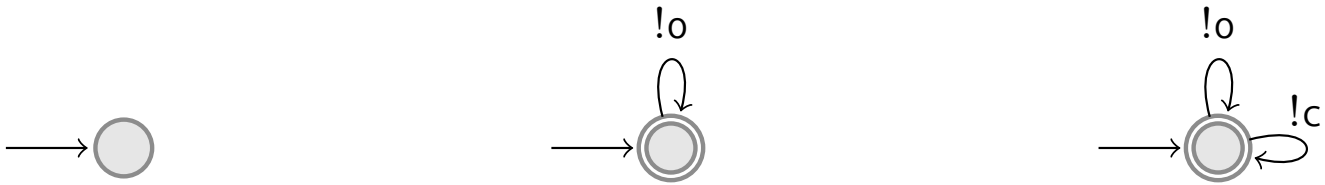


- use finer extrapolation: ∇_1

$$\varepsilon\ (\times\ \varepsilon) \xrightarrow{!o} o\ (\times\ \varepsilon)$$

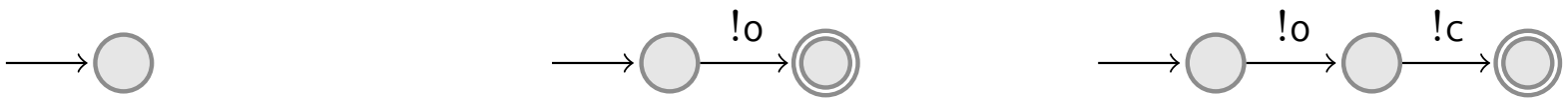
- path invariant via extrapolation: utilize ∇_0

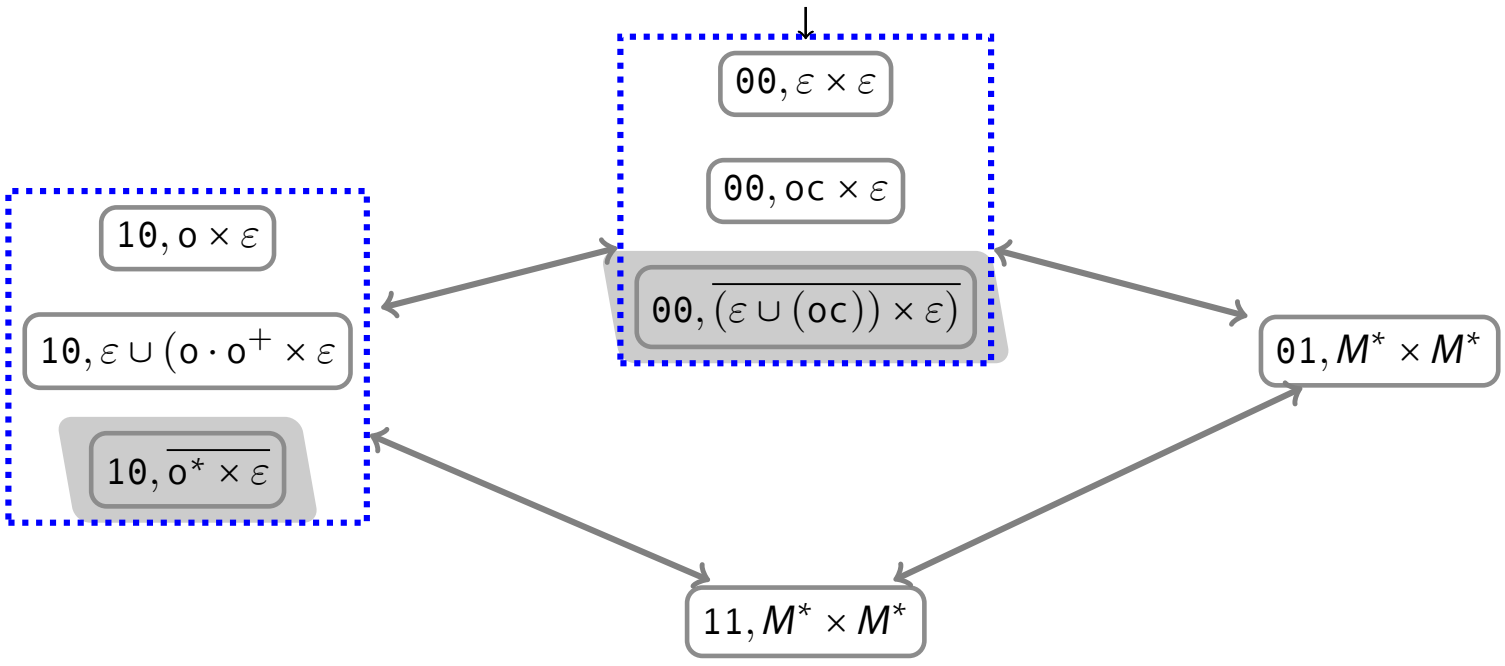
$$\varepsilon\ (\times\ \varepsilon) \xrightarrow{!o} o^*\ (\times\ \varepsilon) \xrightarrow{!c} \{o, c\}^*\ (\times\ \varepsilon)$$ *Bad*



- use finer extrapolation: ∇_1

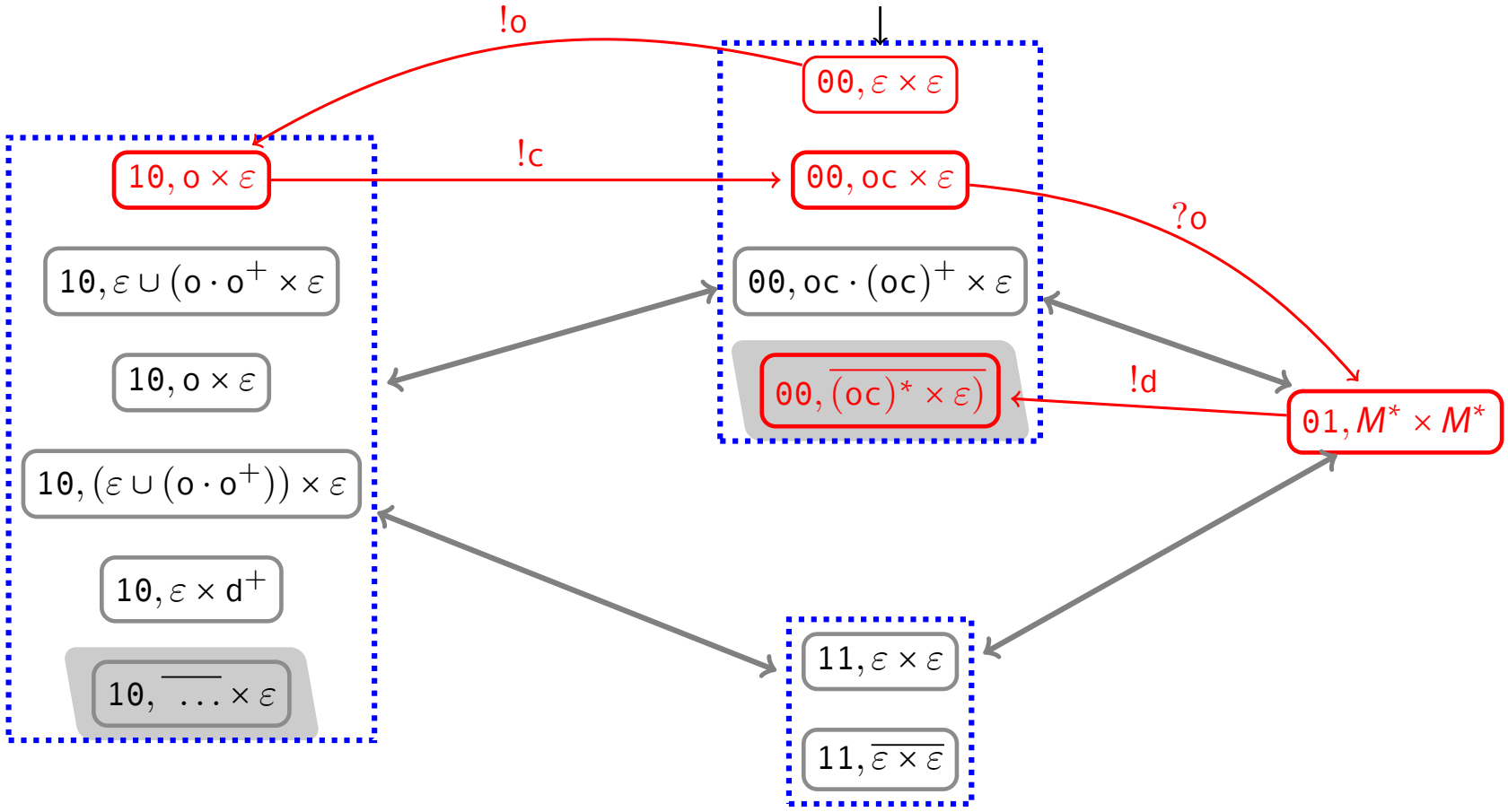
$$\varepsilon\ (\times\ \varepsilon) \xrightarrow{!o} o\ (\times\ \varepsilon) \xrightarrow{!c} oc\ (\times\ \varepsilon)$$ ✓ path invariant





- check, inspect, refine…

…step 8 : counterexample



- counterexample is feasible ! hence, *Bad* is reachable ↯

Technical Contributions

- parametrized extrapolation

$$\nabla : \mathbb{N} \to (Rec((M^*)^n) \to Rec((M^*)^n))$$

  - use finite automata quotienting
    wrt. colored bisimulation of parametrized depth

- different algorithms for path invariant generation
  - upinv : apply $post \circ \nabla_k$ along path, increase k if needed
  - apinv : split based on ∇_k (from ``failure'' node)
  - forward, backward variants

- partial termination results
  - terminates if fifo system is unsafe
  - and for fifo systems with a finite reachability set

Technical Contributions

- parametrized extrapolation

$$\nabla : \mathbb{N} \to (Rec((M^*)^n) \to Rec((M^*)^n))$$

  - use finite automata quotienting
    wrt. colored bisimulation of parametrized depth

- different algorithms for path invariant generation
  - upinv : apply $post \circ \nabla_k$ along path, increase k if needed
  - apinv : split based on ∇_k (from ``failure'' node)
  - forward, backward variants

- partial termination results
  - terminates if fifo system is unsafe
  - and for fifo systems with a finite reachability set

Technical Contributions

- parametrized extrapolation

$$\nabla : \mathbb{N} \to (Rec((M^*)^n) \to Rec((M^*)^n))$$

  - use finite automata quotienting
    wrt. colored bisimulation of parametrized depth

- different algorithms for path invariant generation
  - upinv : apply $post \circ \nabla_k$ along path, increase k if needed
  - apinv : split based on ∇_k (from ``failure'' node)
  - forward, backward variants

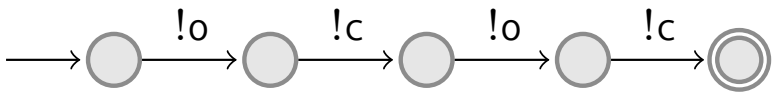
- partial termination results
  - terminates if fifo system is unsafe
  - and for fifo systems with a finite reachability set

Comparison to (some) Other Approaches

- acceleration based approaches
  - Lash/QDD, TReX/SRE
  - no counterexamples
  - our approach mimics acceleration

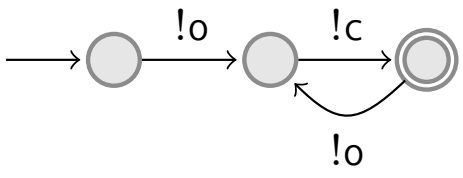
Comparison to (some) Other Approaches

- acceleration based approaches
  - Lash/QDD, TReX/SRE
  - no counterexamples
  - our approach mimics acceleration



Comparison to (some) Other Approaches

- acceleration based approaches
  - Lash/QDD, TReX/SRE
  - no counterexamples
  - our approach mimics acceleration



application of ∇_1
(bisimulation of depth 1)
leads to $(\mathtt{oc})^+$

Comparison to (some) Other Approaches

- acceleration based approaches
  - Lash/QDD, TReX/SRE
  - no counterexamples
  - our approach mimics acceleration
- ARMC - Abstract Regular Model Checking
  - approximative fixpoint iteration based on regular languages
  - global refinement / uniform precision
- our approach
  - local refinement / adaptive precision
  - flexible wrt. extrapolation and invariant generation
  - generic (not bound to automata with "regular" data)
  - counterexample guided

Comparison to (some) Other Approaches

- acceleration based approaches
  - Lash/QDD, TReX/SRE
  - no counterexamples
  - our approach mimics acceleration
- ARMC - Abstract Regular Model Checking
  - approximative fixpoint iteration based on regular languages
  - global refinement / uniform precision
- our approach
  - local refinement / adaptive precision
  - flexible wrt. extrapolation and invariant generation
  - generic (not bound to automata with "regular" data)
  - counterexample guided

Comparison to (some) Other Approaches

- acceleration based approaches
  - Lash/QDD, TReX/SRE
  - no counterexamples
  - our approach mimics acceleration
- ARMC - Abstract Regular Model Checking
  - approximative fixpoint iteration based on regular languages
  - global refinement / uniform precision
- our approach
  - local refinement / adaptive precision
  - flexible wrt. extrapolation and invariant generation
  - generic (not bound to automata with "regular" data)
  - counterexample guided

Empirical Evaluation

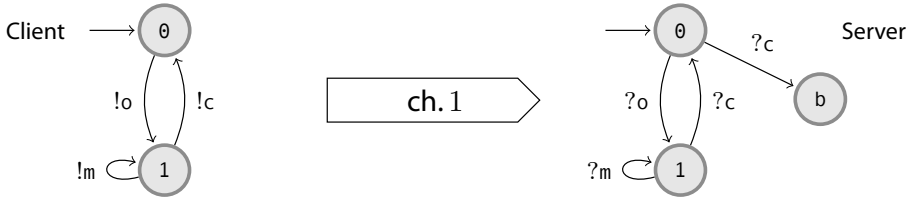
Some Benchmarks…

control automaton!

protocol	states/trans.	time [s]	mem [MiB]	loops	$states^\sharp/trans^\sharp$
ABP	16/64	2.13	1.58	208	274/1443
c/d protocol	5/17	0.01	0.61	6	11/32
nested c/d protocol	6/17	1.15	1.09	93	100/339
non-regular protocol	9/18	0.06	0.61	14	25/39
Peterson	10648/56628	2.14	32.09	51	10709/56939
(simplified) Tcp	196/588	1.38	2.06	183	431/1439
server with 2 clients	255/2160	9.61	4.97	442	731/7383
token ring	625/4500	4.57	6.42	319	1004/6956
sliding window	225/2010	0.93	2.55	148	388/2367

Some Benchmarks…

protocol	states/trans.	time [s]	mem [MiB]	loops	$\text{states}^\sharp/\text{trans}^\sharp$
ABP	16/64	2.13	1.58	208	274/1443
c/d protocol	5/17	0.01	0.61	6	11/32
nested c/d protocol	6/17	1.15	1.09	93	100/339
non-regular protocol	9/18	0.06	0.61	14	25/39
Peterson	10648/56628	2.14	32.09	51	10709/56939
(simplified) Tcp	196/588	1.38	2.06	183	431/1439
server with 2 clients	255/2160	9.61	4.97	442	731/7383
token ring	625/4500	4.57	6.42	319	1004/6956
sliding window	225/2010	0.93	2.55	148	388/2367

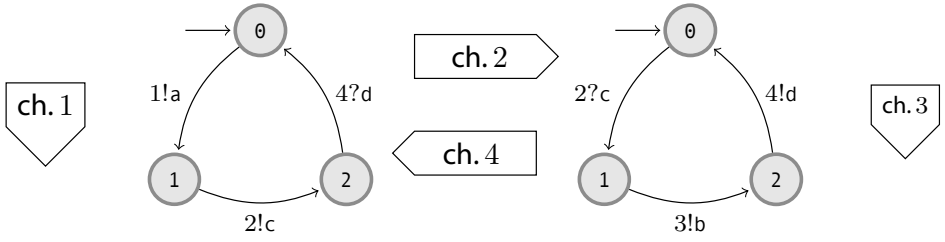


nested loops and acceleration

(TReX does not terminate in reasonable time)

Some Benchmarks…

protocol	states/trans.	time [s]	mem [MiB]	loops	states♯/trans♯
ABP	16/64	2.13	1.58	208	274/1443
c/d protocol	5/17	0.01	0.61	6	11/32
nested c/d protocol	6/17	1.15	1.09	93	100/339
non-regular protocol	9/18	0.06	0.61	14	25/39
Peterson	10648/56628	2.14	32.09	51	10709/56939
(simplified) Tcp	196/588	1.38	2.06	183	431/1439
server with 2 clients	255/2160	9.61	4.97	442	731/7383
token ring	625/4500	4.57	6.42	319	1004/6956
sliding window	225/2010	0.93	2.55	148	388/2367



use ch. 1 and ch. 3 as pushdowns

↯ non-regular ↯

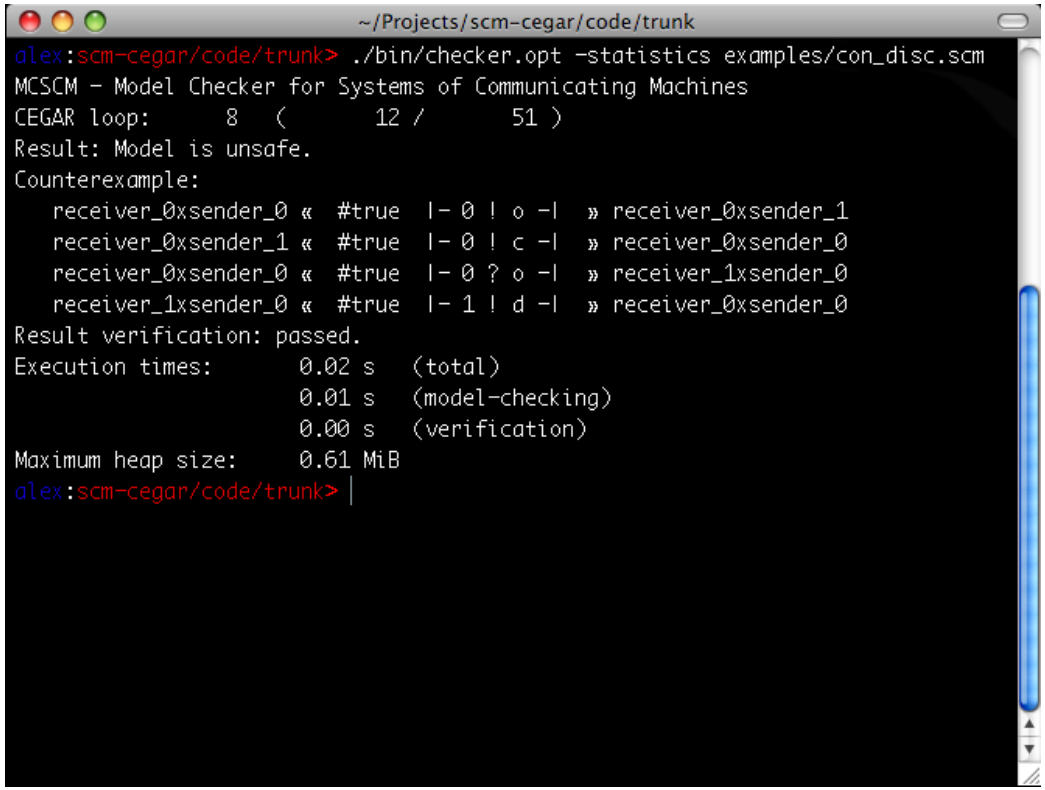
(not compatible with regular model checking)

Summary

- CEGAR for safety verification of infinite state systems
  - generic method
  - based on path invariants
  - and extrapolation
- adapted to fifo systems
  - by encoding partitions as regular languages
  - and utilizing parametrized colored bisimulation equivalence based quotienting for extrapolation
- implemented in tool McScM

MCSCM

- get it at
  `http://altarica.labri.fr/forge/projects/mcscm`
- based on libraries from Tristan Le Gall & Bertrand Jeannet (thanks !)
- BSD-style licence
- programmed in OCaml
- model-checking engines: cegar, lart, absint, armc.
- binary release for the impatient
- coffee-pause demo on demand !



MCSCM

- get it at
  `http://altarica.labri.fr/forge/projects/mcscm`
- based on libraries from Tristan Le Gall & Bertrand Jeannet (thanks !)
- BSD-style licence
- programmed in OCaml
- model-checking engines: cegar, lart, absint, armc.
- binary release for the impatient
- coffee-pause demo on demand !