Some Approaches to Rational Verification in Multiagent Systems

Muhammad Najib

Department of Computer Science
University of Oxford
Oxford, UK

RADICAL, 2017

Outline

Outline

Classical Verification

- Given a system P and formal specification φ

Classical Verification

- Given a system P and formal specification φ
- *Correctness*: Does the behaviour of P reflect φ?

Rational Verification

- How do we define correctness in multiagent systems?

Rational Verification

- How do we define correctness in multiagent systems?
- Each agent has her own goal, and the goals are not necessarily aligned

Rational Verification

- How do we define correctness in multiagent systems?
- Each agent has her own goal, and the goals are not necessarily aligned
- Unlike classical verification, there is no single "litmus test" for system correctness

Rational Verification

- Agents are rational

Rational Verification

- Agents are rational
- Agents pursue their interests strategically

Rational Verification

- Agents are rational
- Agents pursue their interests strategically
- An appropriate framework for studying strategic interaction between self-interested agents: **game theory**

Outline

Nash Equilibrium

- Many *solution concepts* have been proposed

Nash Equilibrium

- Many *solution concepts* have been proposed
- **Nash Equilibrium** (NE) is the most widely-used

Nash Equilibrium

- Many *solution concepts* have been proposed
- **Nash Equilibrium** (NE) is the most widely-used
- A player moving away from NE will be worse off

Nash Equilibrium

- Many *solution concepts* have been proposed
- **Nash Equilibrium** (NE) is the most widely-used
- A player moving away from NE will be worse off
- Moving away (from NE) is irrational

Outline

Model Checking

- A system P into a finite state model (*e.g.* Kripke structure)

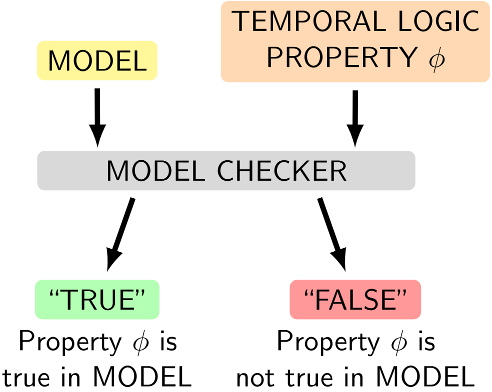


Figure 1: Basic structure of model checking.

Model Checking

- Efficient model checking algorithm for CTL exists

Model Checking

- Efficient model checking algorithm for CTL exists
- LTL model checking is more complex (PSPACE-c)

Model Checking

- Efficient model checking algorithm for CTL exists
- LTL model checking is more complex (PSPACE-c)
- Symbolic MC with BDDs allows very big number of states

Model Checking

- Efficient model checking algorithm for CTL exists
- LTL model checking is more complex (PSPACE-c)
- Symbolic MC with BDDs allows very big number of states
- Active research and development

Outline

SRML

- SRML is a strict subset of Reactive Module Language (RML)

SRML

- SRML is a strict subset of Reactive Module Language (RML)
- A module in SRML consist of:

SRML

- SRML is a strict subset of Reactive Module Language (RML)
- A module in SRML consist of:
  - *interface*: name, list of controlled Boolean variables

SRML

- SRML is a strict subset of Reactive Module Language (RML)
- A module in SRML consist of:
  - *interface*: name, list of controlled Boolean variables
  - *guarded commands*: defines choices available at each state

SRML

Formally, an SRML module $m_i = (\Phi_i, I_i, U_i)$, where:

- $\Phi_i \subseteq \Phi$, set of controlled variables

SRML

Formally, an SRML module $m_i = (\Phi_i, I_i, U_i)$, where:

- $\Phi_i \subseteq \Phi$, set of controlled variables
- I_i is a finite set of **init** guarded commands s.t. $\forall g \in I_i$, $ctr(g) \subseteq \Phi_i$

SRML

Formally, an SRML module $m_i = (\Phi_i, I_i, U_i)$, where:

- $\Phi_i \subseteq \Phi$, set of controlled variables
- I_i is a finite set of **init** guarded commands s.t. $\forall g \in I_i$, $ctr(g) \subseteq \Phi_i$
- U_i is a finite set of **update** guarded commands s.t. $\forall g \in U_i$, $ctr(g) \subseteq \Phi_i$

SRML

module *toggle* **controls** x

init

$:: \top \rightsquigarrow x' := \top;$

$:: \top \rightsquigarrow x' := \bot;$

update

$:: \neg x \rightsquigarrow x' := \top;$

$:: x \rightsquigarrow x' := \bot;$

Figure 2: Example of module toggle in SRML.

SRML Arena

- An SRML arena $A = (N, \Phi, m_1, \ldots, m_n)$

SRML Arena

- An SRML arena $A = (N, \Phi, m_1, \ldots, m_n)$
- $N = \{1, \ldots, n\}$ a set of agents

SRML Arena

- An SRML arena $A = (N, \Phi, m_1, \ldots, m_n)$
- $N = \{1, \ldots, n\}$ a set of agents
- $m_i = (\Phi_i, I_i, U_i)$

SRML Arena

- An SRML arena $A = (N, \Phi, m_1, \ldots, m_n)$
- $N = \{1, \ldots, n\}$ a set of agents
- $m_i = (\Phi_i, I_i, U_i)$
- $\{\Phi_1, \ldots, \Phi_n\}$ is a partition of Φ

Outline

Reactive Module Games

- RMG $G = (A, \gamma_1, \ldots, \gamma_n)$

Reactive Module Games

- RMG $G = (A, \gamma_1, \ldots, \gamma_n)$
- $A = (N, \Phi, m_1, \ldots, m_n)$

Reactive Module Games

- RMG $G = (A, \gamma_1, \ldots, \gamma_n)$
- $A = (N, \Phi, m_1, \ldots, m_n)$
- γ_i is the goal (given by a temporal logic formula) of player i

Reactive Module Games

- RMG $G = (A, \gamma_1, \ldots, \gamma_n)$
- $A = (N, \Phi, m_1, \ldots, m_n)$
- γ_i is the goal (given by a temporal logic formula) of player i
- γ_i in LTL, CTL formula for LTL, CTL RMG, respectively

Strategy

- A *strategy* tells what action should be taken by a player in each possible situation

Strategy

- A *strategy* tells what action should be taken by a player in each possible situation
- A function that maps what an agent “sees” at each state to her action (LTL RMG) or set of actions (CTL RMG)

Strategy

- A *strategy* tells what action should be taken by a player in each possible situation
- A function that maps what an agent "sees" at each state to her action (LTL RMG) or set of actions (CTL RMG)
- Let V_i^t, V_{-i}^t be valuation of $\Phi_i, \Phi \backslash \Phi_i$ at time t, respectively

Strategy

- A *strategy* tells what action should be taken by a player in each possible situation
- A function that maps what an agent “sees” at each state to her action (LTL RMG) or set of actions (CTL RMG)
- Let V_i^t, V_{-i}^t be valuation of $\Phi_i, \Phi \backslash \Phi_i$ at time t, respectively
    - memoryless: $\sigma_i : V_{-i}^t \to V_i^t$ (LTL RMG), $\sigma_i : V_{-i}^t \to 2^{V_i^t}$ (CTL RMG)

Strategy

- A *strategy* tells what action should be taken by a player in each possible situation
- A function that maps what an agent "sees" at each state to her action (LTL RMG) or set of actions (CTL RMG)
- Let V_i^t, V_{-i}^t be valuation of $\Phi_i, \Phi \backslash \Phi_i$ at time t, respectively
    - memoryless: $\sigma_i : V_{-i}^t \rightarrow V_i^t$ (LTL RMG), $\sigma_i : V_{-i}^t \rightarrow 2^{V_i^t}$ (CTL RMG)
    - memoryful: $\sigma_i : V_{-i}^{[0,t]} \rightarrow V_i^t$ (LTL RMG), $\sigma_i : V_{-i}^{[0,t]} \rightarrow 2^{V_i^t}$ (CTL RMG)

Strategy

- A *strategy* tells what action should be taken by a player in each possible situation
- A function that maps what an agent "sees" at each state to her action (LTL RMG) or set of actions (CTL RMG)
- Let V_i^t, V_{-i}^t be valuation of $\Phi_i, \Phi \backslash \Phi_i$ at time t, respectively
  - memoryless: $\sigma_i : V_{-i}^t \rightarrow V_i^t$ (LTL RMG), $\sigma_i : V_{-i}^t \rightarrow 2^{V_i^t}$ (CTL RMG)
  - memoryful: $\sigma_i : V_{-i}^{[0,t]} \rightarrow V_i^t$ (LTL RMG), $\sigma_i : V_{-i}^{[0,t]} \rightarrow 2^{V_i^t}$ (CTL RMG)
- Strategy in LTL RMG is *deterministic*, CTL RMG *non-deterministic*

Strategy Profile

- A strategy profile $\vec{\sigma} = (\sigma_1, \ldots, \sigma_n)$

Strategy Profile

- A strategy profile $\vec{\sigma} = (\sigma_1, \ldots, \sigma_n)$
- LTL RMG: $\vec{\sigma}$ induces a run (an infinite word) $\rho(\vec{\sigma})$

Strategy Profile

- A strategy profile $\vec{\sigma} = (\sigma_1, \ldots, \sigma_n)$
- LTL RMG: $\vec{\sigma}$ induces a run (an infinite word) $\rho(\vec{\sigma})$
- CTL RMG: $\vec{\sigma}$ induces a Kripke structure $K_{\vec{\sigma}}$

Ex: P2P Protocol

Consider a P2P protocol with two peers: Alice and Bob

- At each time-step peers either tries to download or upload

Ex: P2P Protocol

Consider a P2P protocol with two peers: Alice and Bob

- At each time-step peers either tries to download or upload
- In order for one peer to download successfully, the other must be uploading at the same time

Ex: P2P Protocol

Consider a P2P protocol with two peers: Alice and Bob

- At each time-step peers either tries to download or upload
- In order for one peer to download successfully, the other must be uploading at the same time
- Both peers are interested in downloading infinitely often

P2PP in RMG

- $G_{P2P} = (A, \gamma_a, \gamma_b)$

P2PP in RMG

- $G_{P2P} = (A, \gamma_a, \gamma_b)$
- $A = (\{a, b\}, \{u_a, u_b, d_a, d_b\}, m_a, m_b)$

P2PP in RMG

- $G_{P2P} = (A, \gamma_a, \gamma_b)$
- $A = (\{a, b\}, \{u_a, u_b, d_a, d_b\}, m_a, m_b)$
- $\gamma_a = \mathbf{GF}(d_a \wedge u_b)$, $\gamma_b = \mathbf{GF}(d_b \wedge u_a)$

P2PP Arena in SRML

module m_a **controls** u_a, d_a	**module** m_b **controls** u_b, d_b
init	**init**
$:: \top \rightsquigarrow u'_a := \top, d'_a := \bot;$	$:: \top \rightsquigarrow u'_b := \top, d'_b := \bot;$
$:: \top \rightsquigarrow u'_a := \bot, d'_a := \top;$	$:: \top \rightsquigarrow u'_b := \bot, d'_b := \top;$
update	**update**
$:: \top \rightsquigarrow u'_a := \top, d'_a := \bot;$	$:: \top \rightsquigarrow u'_b := \top, d'_b := \bot;$
$:: \top \rightsquigarrow u'_a := \bot, d'_a := \top;$	$:: \top \rightsquigarrow u'_b := \bot, d'_b := \top;$

Figure 3: P2PP arena in SRML.

P2PP Structure

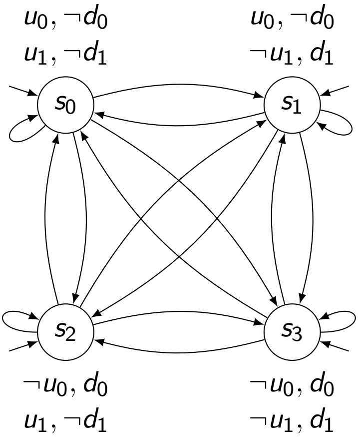


Figure 4: The structure of P2PP arena.

Outline

NE in RMG

- $\vec{\sigma} = (\sigma_1, \ldots, \sigma_{i-1}, \sigma_i, \sigma_{i+1}, \ldots, \sigma_n)$

NE in RMG

- $\vec{\sigma} = (\sigma_1, \ldots, \sigma_{i-1}, \sigma_i, \sigma_{i+1}, \ldots, \sigma_n)$
- $\vec{\sigma}' = (\sigma_1, \ldots, \sigma_{i-1}, \sigma_i', \sigma_{i+1}, \ldots, \sigma_n)$

NE in RMG

- $\vec{\sigma} = (\sigma_1, \ldots, \sigma_{i-1}, \sigma_i, \sigma_{i+1}, \ldots, \sigma_n)$
- $\vec{\sigma}' = (\sigma_1, \ldots, \sigma_{i-1}, \sigma_i', \sigma_{i+1}, \ldots, \sigma_n)$
- Define preference relation $\succsim_i$:

$$\vec{\sigma} \succsim_i \vec{\sigma}' \quad \text{iff} \quad \vec{\sigma}' \models \gamma_i \quad \text{implies} \quad \vec{\sigma} \models \gamma_i.$$

NE in RMG

- $\vec{\sigma} = (\sigma_1, \ldots, \sigma_{i-1}, \sigma_i, \sigma_{i+1}, \ldots, \sigma_n)$
- $\vec{\sigma}' = (\sigma_1, \ldots, \sigma_{i-1}, \sigma_i', \sigma_{i+1}, \ldots, \sigma_n)$
- Define preference relation $\succsim_i$:

$$\vec{\sigma} \succsim_i \vec{\sigma}' \quad \text{iff} \quad \vec{\sigma}' \models \gamma_i \quad \text{implies} \quad \vec{\sigma} \models \gamma_i.$$

- A strategy profile $\vec{\sigma}$ is said to be a **Nash equilibrium** of G if for all players i and all strategies $\vec{\sigma}' = (\sigma_1, \ldots, \sigma_{i-1}, \sigma_i', \sigma_{i+1}, \ldots, \sigma_n)$ we have $\vec{\sigma} \succsim_i \vec{\sigma}'$

NE in RMG

- $\vec{\sigma} = (\sigma_1, \ldots, \sigma_{i-1}, \sigma_i, \sigma_{i+1}, \ldots, \sigma_n)$
- $\vec{\sigma}' = (\sigma_1, \ldots, \sigma_{i-1}, \sigma_i', \sigma_{i+1}, \ldots, \sigma_n)$
- Define preference relation $\succsim_i$:

$$\vec{\sigma} \succsim_i \vec{\sigma}' \quad \text{iff} \quad \vec{\sigma}' \models \gamma_i \quad \text{implies} \quad \vec{\sigma} \models \gamma_i.$$

- A strategy profile $\vec{\sigma}$ is said to be a **Nash equilibrium** of G if for all players i and all strategies $\vec{\sigma}' = (\sigma_1, \ldots, \sigma_{i-1}, \sigma_i', \sigma_{i+1}, \ldots, \sigma_n)$ we have $\vec{\sigma} \succsim_i \vec{\sigma}'$
- Write $NE(G)$ for the set of pure strategy Nash equilibria

Outline

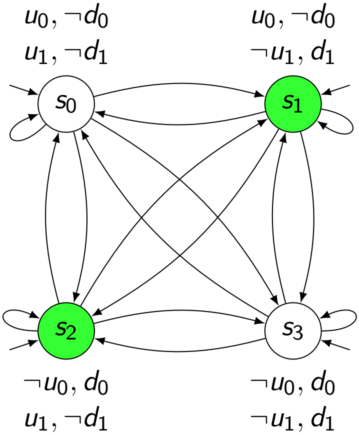
NE-Emptiness

Problem (NE-EMPTINESS)

Given a multiagent system G. Is it the case that $NE(G) \neq \emptyset$?

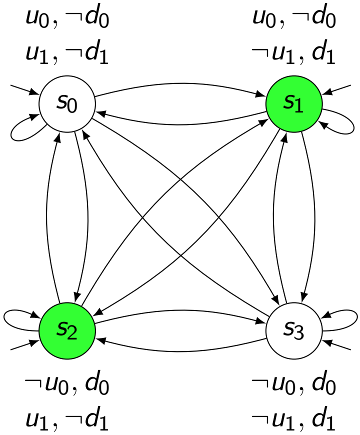
NE-Emptiness

- Obviously, $NE(G_{P2P}) \neq \emptyset$ is true



NE-Emptiness

- Obviously, $NE(G_{P2P}) \neq \emptyset$ is true
- A run that visits s_1 and s_2 infinitely often



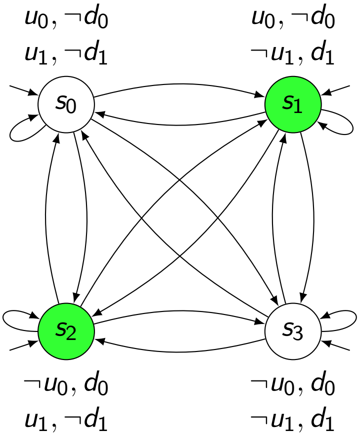
E-Nash

Problem (E-NASH)

Given a multiagent system G and temporal formula φ. Is it the case that $\rho(\vec{\sigma}) \models \varphi$ in any $\vec{\sigma} \in NE(G)$?

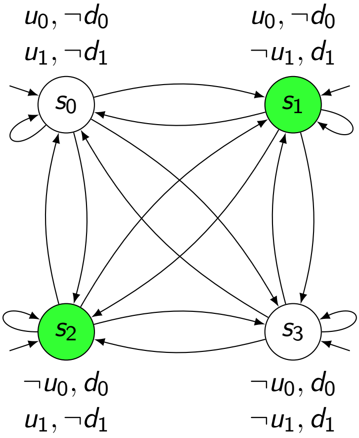
E-Nash

- Let $\varphi = \mathbf{GF}(d_a \wedge u_b)$



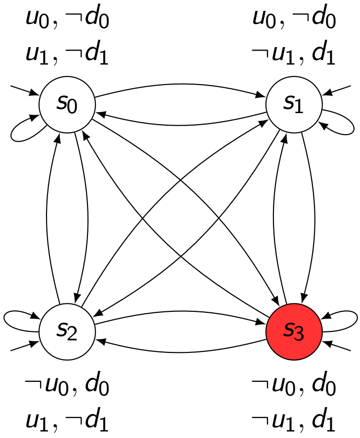
E-Nash

- Let $\varphi = \mathbf{GF}(d_a \wedge u_b)$
- $\exists \vec{\sigma} \in NE(G_{P2P}).\rho(\vec{\sigma}) \models \varphi$ is true



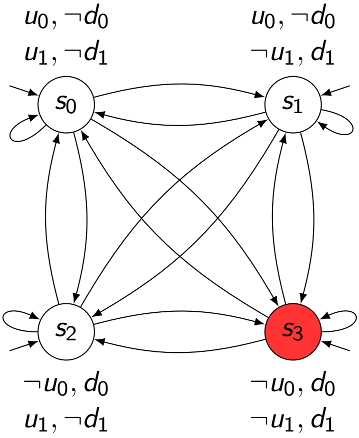
A-Nash

- Let $\varphi = \mathbf{GF}(d_a \wedge u_b)$



A-Nash

- Let $\varphi = \mathbf{GF}(d_a \wedge u_b)$
- $\forall \vec{\sigma} \in NE(G_{P2P}).\rho(\vec{\sigma}) \models \varphi$ is false



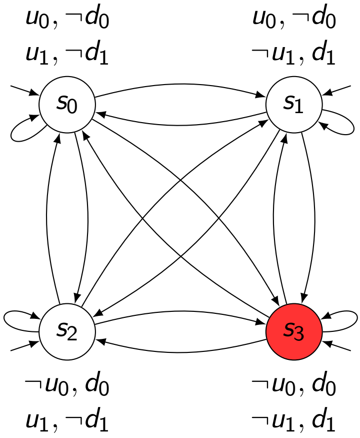
NE-Membership

Problem (NE-Membership)

Given a multiagent system G and strategy profile $\vec{\sigma}$. Is it the case that $\vec{\sigma} \in NE(G)$?

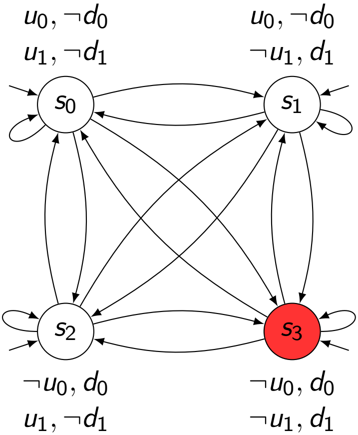
NE-Membership

- Let $\vec{\sigma} = (\sigma_a, \sigma_b)$ where each σ_i prescribes only download



NE-Membership

- Let $\vec{\sigma} = (\sigma_a, \sigma_b)$ where each σ_i prescribes only download
- Then $\vec{\sigma} \in NE(G)$ is true



Outline

Computational Complexity Results

	LTL RMG	CTL RMG
NE-Emptiness	2EXPTIME-c	2EXPTIME-hard
E-Nash	2EXPTIME-c	2EXPTIME-hard
A-Nash	2EXPTIME-c	2EXPTIME-hard
NE-Membership	PSPACE-c	2EXPTIME-c

Table 1: Overview of computational complexity results [Gutierrez et al., 2017]

Outline

MCMAS

- MCMAS [Lomuscio et al., 2015] uses interpreted systems [Fagin et al., 1995] for representation

MCMAS

- MCMAS [Lomuscio et al., 2015] uses interpreted systems [Fagin et al., 1995] for representation
- Uses *global* and *local* states to capture epistemic properties

MCMAS

- MCMAS [Lomuscio et al., 2015] uses interpreted systems [Fagin et al., 1995] for representation
- Uses *global* and *local* states to capture epistemic properties
- Latest implementation supports ATL and SL

MCMAS

- MCMAS [Lomuscio et al., 2015] uses interpreted systems [Fagin et al., 1995] for representation
- Uses *global* and *local* states to capture epistemic properties
- Latest implementation supports ATL and SL
- SL quite expressive, possible to reason about NE

MCMAS

Pros:

- Has been around for more than 10 years

Cons:

MCMAS

Pros:

- Has been around for more than 10 years
- Support GUI

Cons:

MCMAS

Pros:

- Has been around for more than 10 years
- Support GUI
- Written in a fast language (C/C++)

Cons:

MCMAS

Pros:

- Has been around for more than 10 years
- Support GUI
- Written in a fast language (C/C++)
- Multiplatform

Cons:

MCMAS

Pros:

- Has been around for more than 10 years
- Support GUI
- Written in a fast language (C/C++)
- Multiplatform
- Symbolic model checking with BDDs

Cons:

MCMAS

Pros:

- Has been around for more than 10 years
- Support GUI
- Written in a fast language (C/C++)
- Multiplatform
- Symbolic model checking with BDDs

Cons:

- Current implementation (SLK) only supports memoryless strategy

MCMAS

Pros:

- Has been around for more than 10 years
- Support GUI
- Written in a fast language (C/C++)
- Multiplatform
- Symbolic model checking with BDDs

Cons:

- Current implementation (SLK) only supports memoryless strategy
- Verbosity of ISPL

MCMAS

Pros:

- Has been around for more than 10 years
- Support GUI
- Written in a fast language (C/C++)
- Multiplatform
- Symbolic model checking with BDDs

Cons:

- Current implementation (SLK) only supports memoryless strategy
- Verbosity of ISPL
- No direct support for rational verification

Outline

EAGLE

- EAGLE [Toumi et al., 2015] is a prototype tool for equilibrium checking

EAGLE

- EAGLE [Toumi et al., 2015] is a prototype tool for equilibrium checking
- Particularly solves NE-Membership

EAGLE

- EAGLE [Toumi et al., 2015] is a prototype tool for equilibrium checking
- Particularly solves NE-MEMBERSHIP
- Accepts CTL as specification language

EAGLE

- EAGLE [Toumi et al., 2015] is a prototype tool for equilibrium checking
- Particularly solves NE-MEMBERSHIP
- Accepts CTL as specification language
- Needs two inputs: system (modelled as CTL RMG) $G = (A, \gamma_1, \ldots, \gamma_n)$ and strategy profile $\vec{\sigma} = (\sigma_1, \ldots, \sigma_n)$

EAGLE

EAGLE's basic algorithm:

1. Build $K_{\vec{\sigma}}$

[1] calling a CTL model checker oracle

[2] calling a CLT SAT oracle; A_{CTL} is CTL representation of $A = (N, \Phi, m_1, \ldots, m_n)$

EAGLE

EAGLE's basic algorithm:

1. Build $K_{\vec{\sigma}}$
2. If $\exists i \in N$ s.t. $(K_{\vec{\sigma}} \not\models \gamma_i)$[1] and $Sat(A_{CTL} \wedge \gamma_i)$[2] returns true, then output "NO"; otherwise "YES"

[1] calling a CTL model checker oracle

[2] calling a CLT SAT oracle; A_{CTL} is CTL representation of $A = (N, \Phi, m_1, \ldots, m_n)$

EAGLE

Pros:

- Uses SRML which is quite compact

Cons:

EAGLE

Pros:

- Uses SRML which is quite compact
- Strategies are not memoryless

Cons:

EAGLE

Pros:

- Uses SRML which is quite compact
- Strategies are not memoryless
- No need to devise meta-algorithm for rational verification

Cons:

EAGLE

Pros:

- Uses SRML which is quite compact
- Strategies are not memoryless
- No need to devise meta-algorithm for rational verification

Cons:

- Still a prototype and not optimised yet

EAGLE

Pros:

- Uses SRML which is quite compact
- Strategies are not memoryless
- No need to devise meta-algorithm for rational verification

Cons:

- Still a prototype and not optimised yet
- Uses CTL which relatively less intuitive from designer POV

EAGLE

Pros:

- Uses SRML which is quite compact
- Strategies are not memoryless
- No need to devise meta-algorithm for rational verification

Cons:

- Still a prototype and not optimised yet
- Uses CTL which relatively less intuitive from designer POV
- CTL SAT bottleneck

EAGLE

Pros:

- Uses SRML which is quite compact
- Strategies are not memoryless
- No need to devise meta-algorithm for rational verification

Cons:

- Still a prototype and not optimised yet
- Uses CTL which relatively less intuitive from designer POV
- CTL SAT bottleneck
- No GUI

Outline

EAGLE with BDD CTL SAT

- As reported in [Toumi et al., 2015], CTL SAT subroutine is the bottleneck

EAGLE with BDD CTL SAT

- As reported in [Toumi et al., 2015], CTL SAT subroutine is the bottleneck
- Can we check CTL SAT symbolically with BDD [Marrero, 2005]?

Outline

EVE

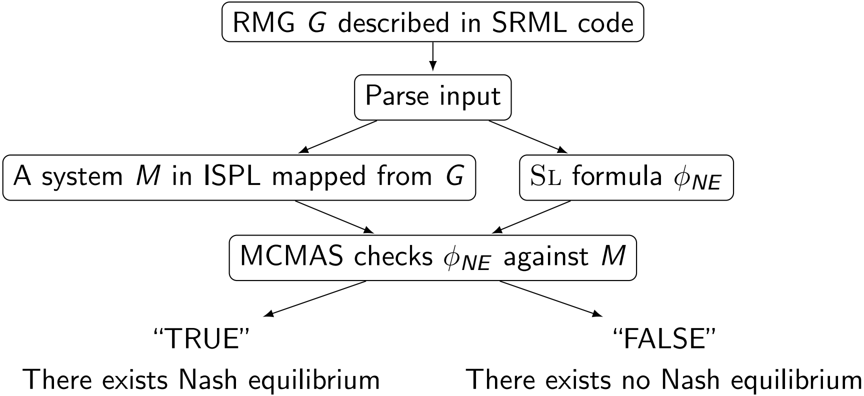


Figure 5: The implementation structure of EVE.

EVE

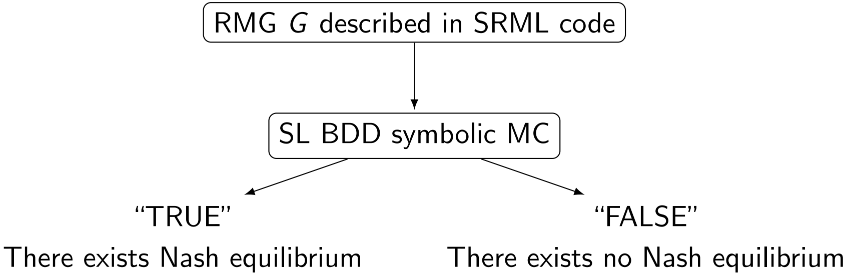


Figure 6: Inject RMG directly into SL BDD symbolic model checker.

Outline

The Punisher(s)

Lemma ([Gutierrez et al., 2015])

ρ is sustained by a Nash equilibrium strategy profile iff every player j whose goal is not satisfied by ρ is punishable at ρ

The Punisher(s)

Nash equilibrium = Punishability + Memory

Why Parity?

- Memoryless determinacy

Why Parity?

- Memoryless determinacy
- Solves the problem of keeping track deviating run

Why Parity?

- Memoryless determinacy
- Solves the problem of keeping track deviating run
- Finite number of memoryless strategies

Why Parity?

- Memoryless determinacy
- Solves the problem of keeping track deviating run
- Finite number of memoryless strategies
- Development of algorithms to solve PG (latest: quasipolynomial ([Calude et al., 2017], best paper award STOC 2017))

Basic Structure[3]

- $G_{LTL} \Rrightarrow G_{PAR}$

Matches theoretical bound of 2EXPTIME for LTL RMGs

[3] ongoing joint work: Julian Gutierrez, Giuseppe Perelli, Michael Wooldridge

Basic Structure[3]

- $G_{LTL} \Rrightarrow G_{PAR}$
- Compute punishing region PUN

Matches theoretical bound of 2EXPTIME for LTL RMGs

[3] ongoing joint work: Julian Gutierrez, Giuseppe Perelli, Michael Wooldridge

Basic Structure[3]

- $G_{LTL} \Rrightarrow G_{PAR}$
- Compute punishing region PUN
- $G_{PAR} \Rrightarrow G^{W}_{PAR}$ from $PUN_{N \setminus W}, W \subseteq N$ (winning coalition)

Matches theoretical bound of 2EXPTIME for LTL RMGs

[3] ongoing joint work: Julian Gutierrez, Giuseppe Perelli, Michael Wooldridge

Basic Structure[3]

- $G_{LTL} \Rrightarrow G_{PAR}$
- Compute punishing region PUN
- $G_{PAR} \Rrightarrow G^{W}_{PAR}$ from $PUN_{N \setminus W}, W \subseteq N$ (winning coalition)
- If $\exists W$ win in G^{W}_{PAR}, then yes; otherwise no

Matches theoretical bound of 2EXPTIME for LTL RMGs

[3] ongoing joint work: Julian Gutierrez, Giuseppe Perelli, Michael Wooldridge

Demo

- Consider a network composed of 2 clients: $client_a, client_b$ and 2 servers: $server_1, server_2$

Demo

- Consider a network composed of 2 clients: $client_a, client_b$ and 2 servers: $server_1, server_2$
- $client_a$ handles urgent tasks, so everytime it sends a request, needs to be served immediately

Demo

- Consider a network composed of 2 clients: $client_a, client_b$ and 2 servers: $server_1, server_2$
- $client_a$ handles urgent tasks, so everytime it sends a request, needs to be served immediately
- $client_b$ doesn't handle urgent task, no need to be served immediately

Demo

- Consider a network composed of 2 clients: $client_a, client_b$ and 2 servers: $server_1, server_2$
- $client_a$ handles urgent tasks, so everytime it sends a request, needs to be served immediately
- $client_b$ doesn't handle urgent task, no need to be served immediately
- $server_1$ is an old server, it needs longer rest time

Demo

- Consider a network composed of 2 clients: $client_a, client_b$ and 2 servers: $server_1, server_2$
- $client_a$ handles urgent tasks, so everytime it sends a request, needs to be served immediately
- $client_b$ doesn't handle urgent task, no need to be served immediately
- $server_1$ is an old server, it needs longer rest time
- $server_2$ needs shorter rest time

Demo

- $\gamma_a = \mathbf{G}(r_a \rightarrow \mathbf{X}(s_1 \vee s_2))$

Demo

- $\gamma_a = \mathbf{G}(r_a \rightarrow \mathbf{X}(s_1 \vee s_2))$
- $\gamma_b = \mathbf{G}(r_a \rightarrow \mathbf{F}(s_1 \vee s_2))$

Demo

- $\gamma_a = \mathbf{G}(r_a \rightarrow \mathbf{X}(s_1 \vee s_2))$
- $\gamma_b = \mathbf{G}(r_a \rightarrow \mathbf{F}(s_1 \vee s_2))$
- $\gamma_1 = \mathbf{GF}(\neg s_1 \wedge \mathbf{X}\neg s_1)$

Demo

- $\gamma_a = \mathbf{G}(r_a \rightarrow \mathbf{X}(s_1 \vee s_2))$
- $\gamma_b = \mathbf{G}(r_a \rightarrow \mathbf{F}(s_1 \vee s_2))$
- $\gamma_1 = \mathbf{GF}(\neg s_1 \wedge \mathbf{X} \neg s_1)$
- $\gamma_2 = \mathbf{GF} \neg s_2$

References I

Calude, C. S., Jain, S., Khoussainov, B., Li, W., and Stephan, F. (2017).
Deciding parity games in quasipolynomial time.
STOC.
To appear.

References II

Gutierrez, J., Harrenstein, P., and Wooldridge, M. (2017).
From model checking to equilibrium checking: Reactive modules for rational verification.
Artif. Intell., 248:123–157.

Lomuscio, A., Qu, H., and Raimondi, F. (2015).
Mcmas: an open-source model checker for the verification of multi-agent systems.
International Journal on Software Tools for Technology Transfer, pages 1–22.

References III

Marrero, W. (2005).
Using bdds to decide CTL.
In *Tools and Algorithms for the Construction and Analysis of Systems, 11th International Conference, TACAS 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005, Proceedings*, pages 222–236.

Toumi, A., Gutierrez, J., and Wooldridge, M. (2015).
Theoretical Aspects of Computing - ICTAC 2015: 12th International Colloquium, Cali, Colombia, October 29-31, 2015, Proceedings, chapter A Tool for the Automated Verification of Nash Equilibria in Concurrent Games, pages 583–594.
Springer International Publishing, Cham.