PRISM–games

Model Checking for Stochastic Games

Dave Parker
University of Oxford

Dagstuhl seminar “Stochastic Games”
June 2024
Verification with stochastic games

• How do we formally verify stochastic systems with…
  – multiple autonomous agents acting concurrently
  – competitive or collaborative behaviour between agents, often with differing/opposing goals
  – e.g. security protocols, algorithms for distributed consensus, energy management, autonomous robotics, auctions

• Probabilistic model checking for stochastic games
  – synthesis and verification of strategies for agents to provide guarantees on safety/performance/… in adversarial settings and stochastic environments
Probabilistic model checking

System requirements → Specification (temporal logic) → Probabilistic model -> Probabilistic model checking

Numerical results/analysis → Strategies/controllers

\[ \langle \langle r_1 \rangle \rangle _{P \geq 0.9} [\neg h \cup g ] \]
PRISM-games

- **PRISM-games**: prismmodelchecker.org/games
  - extension of PRISM for stochastic games
  - modelling language + model checking + user interface
  - explicit state & symbolic implementations; simulation

- **Example applications** (see web site for ~40 case studies)
  - attack–defence trees; network protocols; intrusion detection
  - human–in–the–loop UAV planning; multi–robot systems
  - autonomous driving; self–adaptive software architectures
  - collective decision making; team formation; trust models
Overview

- **Models & modelling**
  - stochastic multi-player games

- **Property specification**
  - temporal logics

- **Solving stochastic games**
  - algorithms, tools, case studies
  - turn-based/concurrent games
  - zero-sum/equilibria
Models & modelling
Stochastic multi-player games

**Turn-based stochastic games (TSGs)**

Transition function:
\[ \delta : S \times A \rightarrow \text{Dist}(S) \]

With state partition:
\[ S = S_1 \cup \ldots \cup S_n \]

Player \( i \) controls states \( S_i \)

---

**Concurrent stochastic games (CSGs)**
(also: Markov games, multi-agent MDPs)

Transition function:
\[ \delta : S \times (A_1 \cup \{\perp\}) \times \ldots \times (A_n \cup \{\perp\}) \rightarrow \text{Dist}(S) \]

With joint action space:
\[ A = A_1 \times \ldots \times A_n \]

Actions chosen simultaneously

Stochastic multi-player games

**Turn-based stochastic games (TSGs)**

- Strategies (for player $i$)
  - $\sigma_i : (S \times A)^* S_i \to \text{Dist}(A)$

**Concurrent stochastic games (CSGs)**
(also: Markov games, multi-agent MDPs)

- Strategies (for player $i$)
  - $\sigma_i : (S \times A)^* S \to \text{Dist}(A_i \cup \{\bot\})$

- $\sigma_i$ can be deterministic/randomised, memoryless/finite-memory/…
- Strategy profile $\sigma = (\sigma_1, \ldots, \sigma_n)$ for all $n$ players
- Probability space $\Pr_s^\sigma(\psi)$, or (reward-based) expectation $E_s^\sigma(X)$
Modelling with turn–based games

- **Turn–based stochastic games**
  - well suited to some (but not all) scenarios

Shared autonomy: human–robot control

Uncontrollable/unknown navigation interference

TSG models

Shared autonomy: human–robot control

Uncontrollable/unknown navigation interference

TSG models
Modelling with concurrent games

- Concurrent stochastic games
  - example: CSG for 2 robots on a 3x1 grid
Modelling with concurrent games

- **Concurrent stochastic games**
  - example: CSG for 2 robots on a 3x1 grid
PRISM(–games) modelling language

- **PRISM modelling language**
  - de-facto standard for probabilistic model checkers
  - key ingredients: modules, variables, guarded commands
  - language features: nondeterminism + probability, parallel composition, costs/rewards, parameters

- **PRISM–games modelling language**
  - adds: player specifications, joint update distributions
csg
player p1 user1 endplayer
player p2 user2 endplayer
// Users (senders)
module user1
    s1 : [0..1] init 0; // has player 1 sent?
    e1 : [0..emax] init emax; // energy level of player 1
    [w1] true -> (s1'=0); // wait
    [t1] e1>0 -> (s1'=c'?0:1) & (e1'=e1-1); // transmit
endmodule
module user2 = user1 [ s1=s2, e1=e2, w1=w2, t1=t2 ] endmodule
// Channel: used to compute joint probability distribution for transmission failure
module channel
    c : bool init false; // is there a collision?
    [t1,w2] true -> q1 : (c'=false) + (1-q1) : (c'=true); // only user 1 transmits
    [w1,t2] true -> q1 : (c'=false) + (1-q1) : (c'=true); // only user 2 transmits
    [t1,t2] true -> q2 : (c'=false) + (1-q2) : (c'=true); // both users transmit
endmodule
PRISM(-games) modelling language

csg
player p1 user1 endplayer
player p2 user2 endplayer
// Users (senders)
module user1
    s1 : [0..1] init 0; // has player 1 sent?
e1 : [0..emax] init emax; // energy level of player 1
[w1] true -> (s1'=0); // wait
[t1] e1>0 -> (s1'=c'?0:1) & (e1'=e1-1); // transmit
endmodule
module user2 = user1 [s1=s2, e1=e2, w1=w2, t1=t2] endmodule
// Channel: used to compute joint probability distribution for transmission failure
module channel
    c : bool init false; // is there a collision?
[t1,w2] true -> q1 : (c'=false) + (1-q1) : (c'=true); // only user 1 transmits
[w1,t2] true -> q1 : (c'=false) + (1-q1) : (c'=true); // only user 2 transmits
[t1,t2] true -> q2 : (c'=false) + (1-q2) : (c'=true); // both users transmit
endmodule
PRISM(-games) modelling language

Variables define the model state

Guarded commands describe (probabilistic) state updates

```
csg
player p1 user1 endplayer
player p2 user2 endplayer
// Users (senders)
module user1
  s1 : [0..1] init 0; // has player 1 sent?
  e1 : [0..emax] init emax; // energy level of player 1
  [w1] true -> (s1'=0); // wait
  [t1] e1>0 -> (s1'=c'? 0 : 1) & (e1'=e1-1); // transmit
endmodule
module user2 = user1 [ s1=s2, e1=e2, w1=w2, t1=t2 ] endmodule
// Channel: used to compute joint probability distribution for transmission failure
module channel
  c : bool init false; // is there a collision?
  [t1,w2] true -> q1 : (c'=false) + (1-q1) : (c'=true); // only user 1 transmits
  [w1,t2] true -> q1 : (c'=false) + (1-q1) : (c'=true); // only user 2 transmits
  [t1,t2] true -> q2 : (c'=false) + (1-q2) : (c'=true); // both users transmit
endmodule
```
PRISM(-games) modelling language

player p1 user1 endplayer
player p2 user2 endplayer

// Users (senders)
module user1
    s1 : [0..1] init 0; // has player 1 sent?
    e1 : [0..emax] initemax, // energy level of player 1
    [w1] true -> (s1'=0); // wait
    [t1] e1>0 -> (s1'=c' ? 0 : 1) & (e1'=e1-1); // transmit
endmodule

module user2 = user1 [ s1=s2, e1=e2, w1=w2, t1=t2 ] endmodule

// Channel: used to compute joint probability distribution for transmission failure
module channel
    c : bool init false; // is there a collision?
    [t1,w2] true -> q1 : (c'=false) + (1-q1) : (c'=true); // only user 1 transmits
    [w1,t2] true -> q1 : (c'=false) + (1-q1) : (c'=true); // only user 2 transmits
    [t1,t2] true -> q2 : (c'=false) + (1-q2) : (c'=true); // both users transmit
endmodule

Each player comprises one or more modules
Players have distinct actions, executed simultaneously

csg

// Users (senders)
module user1

// Channel: used to compute joint probability distribution for transmission failure
module channel

**PRISM(-games) modelling language**

```plaintext
// Users (senders)
module user1
  s1 : [0..1] init 0; // has player 1 sent?
  e1 : [0..emax] init emax; // energy level of player 1
  [w1] true -> (s1'=0); // wait
  [t1] e1>0 -> (s1'=c' ? 0 : 1) & (e1'=e1-1); // transmit
endmodule

module user2 = user1 [ s1=s2, e1=e2, w1=w2, t1=t2 ] endmodule

// Channel: used to compute joint probability distribution for transmission failure
module channel
  c : bool init false; // is there a collision?
  [t1,w2] true -> q1 : (c'=false) + (1-q1) : (c'=true); // only user 1 transmits
  [w1,t2] true -> q1 : (c'=false) + (1-q1) : (c'=true); // only user 2 transmits
  [t1,t2] true -> q2 : (c'=false) + (1-q2) : (c'=true); // both users transmit
endmodule
```

Variable updates can refer to other variables updated simultaneously.

Action lists used to specify synchronisation.
PRISM(–games) modelling language

• **PRISM modelling language**
  – de-facto standard for probabilistic model checkers
  – key ingredients: modules, variables, guarded commands
  – language features: nondeterminism + probability, parallel composition, costs/rewards, parameters

• **PRISM–games modelling language**
  – adds: player specifications, joint update distributions

• **Some observations:**
  – simple/low-level: no control flow/functions, limited types, …
  + uniform language for many types of probabilistic model
  + translations exist from more expressive languages
  + well suited to symbolic methods (NB: but not to simulation)
Temporal logic
Temporal logic: rPATL

- **Temporal logic for stochastic games**
  - unambiguous, flexible & tractable behavioural specification
  - basis: rPATL (reward probabilistic alternating temporal logic)

- **rPATL is a branching–time logic (extending CTL) with:**
  - coalition operator $\langle\langle C \rangle\rangle$ of ATL
  - probabilistic operator $P$ of PCTL
  - generalised (expected) reward operator $R$ from PRISM
  - i.e.: zero–sum, probabilistic reachability + exp. cumul. reward

- **Example:**
  - $\langle\langle \{r_1, r_3\} \rangle\rangle \ P_{>0.99} [ F_{\leq 10} (goal_1 \lor goal_3) ]$
  - “robots 1 and 3 have a strategy to ensure that the probability of reaching a goal location within 10 steps is >0.99, regardless of the strategies of other players”
Temporal logic: rPATL

- **Temporal logic for stochastic games**
  - unambiguous, flexible & tractable behavioural specification
  - basis: rPATL (reward probabilistic alternating temporal logic)

- **rPATL is a branching–time logic (extending CTL) with:**
  - coalition operator $\langle\langle C \rangle\rangle$ of ATL
  - probabilistic operator $P$ of PCTL
  - generalised (expected) reward operator $R$ from PRISM
  - i.e.: zero–sum, probabilistic reachability + exp. cumul. reward

- **Semantics:**
  - $s \models \langle\langle C \rangle\rangle P_{\bowtie q}[\psi]$ iff:
  - “there exist strategies for players in coalition C such that, for all strategies of the other players, the probability of path formula $\psi$ being true from state $s$ satisfies $\bowtie q$”
Temporal logic

- **Simple examples (rPATL)**
  - Probabilistic reachability
    \[ \langle\langle r_1 \rangle\rangle P_{\geq 0.7} [ F \text{ goal}_1 ] \]
    \[ \langle\langle r_1 \rangle\rangle P_{\geq 0.6} [ F_{\leq 10} \text{ goal}_1 ] \]
  - Probabilistic safety/invariance
    \[ \langle\langle r_1 \rangle\rangle P_{\geq 0.99} [ G\neg\text{hazard} ] \]
  - Probabilistic reach–avoid
    \[ \langle\langle r_1 \rangle\rangle P_{\geq 0.99} [ \neg\text{hazard} U \text{ goal}_1 ] \]
  - Expected cost/reward
    \[ \langle\langle r_1 \rangle\rangle R^{\text{steps}}_{\leq 4} [ F \text{ goal}_1 ] \]
  - Numerical (“optimise”) queries
    \[ \langle\langle r_1 \rangle\rangle P_{\max=?} [ F \text{ goal}_1 ] \]
    \[ \langle\langle r_1 \rangle\rangle R^{\text{time}}_{\min=?} [ F \text{ goal}_1 ] \]

**Example TSG: robot navigation**

(Players = robots \( r_1, r_2 \))

![TSG diagram showing robot navigation](diagram.png)
rPATL and beyond

• Nested specifications in rPATL
  – \( \langle\langle \{r_1, r_3\} \rangle \rangle R_{\min=\text{?}} [ \langle\langle \{r_1\} \rangle \rangle P \geq 0.99 [ F \leq 10 \text{ base} ] U (\text{goal}_1 \lor \text{goal}_3) ] \)
  – “minimise expected time for joint task between \( r_1 \) and \( r_3 \),
    whilst ensuring \( r_1 \) can always reliably return to base”

• More expressive temporal specifications
  – e.g. (co–safe) linear temporal logic (LTL)
  – \( \langle\langle \{r_1\} \rangle \rangle P_{\max=\text{?}} [ (G \neg \text{hazard}) \land (GF \text{ goal}_1) ] \)
  – “maximise the probability visiting \( \text{goal}_1 \)
    infinitely often and avoiding hazards”

• Non–zero–sum: e.g. Nash equilibria
  – \( \langle\langle \{r_1\} : \{r_3\} \rangle \rangle (R_{\min=\text{?}} [ F \text{ goal}_1 ] + R_{\min=\text{?}} [ F \text{ goal}_3 ]) \)
  – “minimise the time to reach the goal for each robot”
Solving stochastic games
Model checking rPATL for TSGs

• **Main task: checking individual P and R operators**
  – reduces to solving a (zero–sum) 2–player TSG
  – e.g. max/min reachability probability: \( \sup_{\sigma_1} \inf_{\sigma_2} \Pr_{s_{\sigma_1,\sigma_2}} (F \checkmark) \)
  – optimal strategies are memoryless/deterministic
  – complexity: \( \text{NP} \cap \text{coNP} \) (if we omit some reward operators)

• **We use value iteration**
  – values \( p(s) \) are the least fixed point of:
    \[
    p(s) = \begin{cases} 
    1 & \text{if } s \models \checkmark \\
    \max_a \sum_{s'} \delta(s,a)(s') \cdot p(s') & \text{if } s \not\models \checkmark \text{ and } s \in S_1 \\
    \min_a \sum_{s'} \delta(s,a)(s') \cdot p(s') & \text{if } s \not\models \checkmark \text{ and } s \in S_2 
    \end{cases}
    \]
  – and more: graph–algorithms, sequences of fixed points, …
rPATL for TSGs: Implementation

- **Value iteration for TSGs**
  - similar efficiency and scalability to MDPs
  - (TSGs of, say, $10^7$ states easily solvable)

- **Also symbolic (BDD-based) implementation**
  - exploits model structure/regularity
  - big gains on some models
  - also benefits for strategy compactness

- **Other solution methods (and tools) exist**
  - strategy iteration, quadratic programming
  - interval/optimistic value iteration (for accuracy guarantees)
  - PRISM-games (and extensions), Tempest, PET, EPMC, …
  - see QComp’23 [ABB+24]
Example: Energy protocols

- Demand management protocol for microgrids [CFK+13b]
  - randomised back-off to minimise peaks

- Stochastic game model + rPATL
  - users can collaboratively cheat (i.e., ignore the protocol)
  - TSGs of up to ~6 million states
  - exposes protocol weakness (incentive to act selfishly)
  - propose/verify simple protocol fix using penalties

\[
\langle\langle C \rangle\rangle R^{r_{C_{\text{max}}}} = [F^0 \text{ end}] / |C|
\]
Model checking rPATL for CSGs

- Reduces to solving (zero-sum) 2-player CSGs
  - optimal strategies are now randomised (problem is in PSPACE)

- We again use a value iteration based approach
  - e.g. max/min reachability probabilities
  - \( \sup_{\sigma_1} \inf_{\sigma_2} \Pr_s^{\sigma_1,\sigma_2} (F \checkmark) \) for all states \( s \)
  - values \( p(s) \) are the least fixed point of:
    \[
    p(s) = \begin{cases} 
    1 & \text{if } s \models \checkmark \\
    \text{val}(Z) & \text{if } s \not\models \checkmark 
    \end{cases}
    \]
  - where \( Z \) is the matrix game with \( z_{ij} = \sum_s \delta(s,(a_i,b_j))(s') \cdot p(s') \)
Model checking rPATL for CSGs

- Reduces to solving (zero-sum) 2-player CSGs
  - optimal strategies are now randomised (problem is in PSPACE)

- We again use a value iteration based approach
  - e.g. max/min reachability probabilities
  - \( \sup_{\sigma_1} \inf_{\sigma_2} \Pr_{s,\sigma_1,\sigma_2} (F \checkmark) \) for all states \( s \)
  - values \( p(s) \) are the least fixed point of:

\[
\sum_{s'} \delta(s, (a_i, b_j))(s') \cdot p(s')
\]

- Implementation
  - need to solve a matrix game at every state and every iteration
  - LP problem of size \(|A|\)
  - this is the main performance bottleneck
  - solve CSGs of \(~3\) million states
Example: Future markets investor

- Model of interactions between:
  - stock market, evolves stochastically
  - two investors \( i_1, i_2 \) decide when to invest
  - market decides whether to bar investors

- Modelled as a 3-player CSG
  - extends simpler model originally from [McIver/Morgan’07]
  - investing/barring decisions are simultaneous
  - profit reduced for simultaneous investments
  - market cannot observe investors’ decisions

- Analysed with rPATL model checking & strategy synthesis
  - distinct profit models considered: ‘normal market’, ‘later cash-ins’ and ‘later cash-ins with fluctuation’
  - comparison between TSG and CSG models
Example: Future markets investor

- Example rPATL query:
  - $\langle\langle\text{investor}_1,\text{investor}_2\rangle\rangle$ $R_{\text{profit}^{1,2}}^{\text{max}} = ?$ [ F finished$_{1,2}$ ]
  - i.e. maximising joint profit

- Results: with (left) and without (right) fluctuations
  - optimal (randomised) investment strategies synthesised
  - CSG yields more realistic results (market has less power due to limited observation of investor strategies)

Too pessimistic: unrealistic strategy for adversary
Equilibria–based properties
Equilibria-based properties

- **Non-zero-sum CSGs**
  - player objectives are distinct, but not directly opposing

- **For now: Nash equilibria (NE)** (we will later use other equilibria)
  - no incentive for any player to unilaterally change strategy
  - actually, we use $\varepsilon$-NE, which always exist for CSGs
  - a strategy profile $\sigma=(\sigma_1,\ldots,\sigma_n)$ for a CSG is an $\varepsilon$-NE for state $s$ and objectives $X_1,\ldots,X_n$ iff:
    - $\Pr_s^\sigma(X_i) \geq \sup \{ \Pr_s^{\sigma'}(X_i) \mid \sigma'=\sigma_{-i}[\sigma_i'] \text{ and } \sigma_i' \in \Sigma_i \} - \varepsilon$ for all $i$
    - we use subgame-perfect $\varepsilon$-NE, where this holds for all states $s$

- **To formulate the model checking (strategy synthesis) problem, we use social-welfare Nash equilibria (SWNE)**
  - these are NE which maximise the sum $E_s^\sigma(X_1) + \ldots E_s^\sigma(X_n)$
  - i.e., optimise the players combined goal
Extending rPATL: Equilibria

- We extend rPATL accordingly:

Zero-sum properties

\[ \langle \langle r_1 \rangle \rangle_{\text{max}} = \max P \left[ F \leq^k \text{goal}_1 \right] \]

find a robot 1 strategy which maximises the probability of it reaching its goal, regardless of robot 2

Equilibria-based properties

\[ \langle \langle r_1 : r_2 \rangle \rangle_{\text{max}} = \max (P \left[ F \leq^k \text{goal}_1 \right] + P \left[ F \leq^k \text{goal}_2 \right]) \]

find (SWNE) strategies for robots 1 and 2 where there is no incentive to change actions and which maximise joint goal probability
Equilibria model checking for CSGs

- Model checking for CSGs with equilibria
  - first: 2-coalition case [KNPS19]
  - we need “stopping game” assumptions
  - requires solution of bimatrix games

- We further extend the value iteration approach:

\[
p(s) = \begin{cases} 
(1,1) & \text{if } s \models 1 \land 2 \\
(p_{\text{max}}(s, 1), 1) & \text{if } s \models 1 \land \neg 2 \\
(1, p_{\text{max}}(s, 1)) & \text{if } s \models \neg 1 \land 2 \\
\text{val}(Z_1, Z_2) & \text{if } s \models \neg 1 \land \neg 2 
\end{cases}
\]

- where \( Z_1 \) and \( Z_2 \) encode matrix games similar to before
Equilibria model checking for CSGs

- Model checking for CSGs with equilibria
  - first: 2–coalition case [KNPS19]
  - we need “stopping game” assumptions
  - requires solution of bimatrix games

- Implementation
  - we adapt a known approach using labelled polytopes, implemented via SMT
  - optimisations: filtering of dominated strategies
  - solve CSGs of ~2 million states

- Extension
  - n–coalition case in [QEST’20]
  - can’t use labelled polytopes
  - needs nonLPs for each state
  - poorer scalability
Example: multi–robot coordination

- 2 robots navigating an $N \times N$ grid
  - start at opposite corners, goals are to navigate to opposite corners
  - obstacles modelled stochastically: navigation in chosen direction fails with probability $q$

- Results (10 x 10 grid)
  - better performance obtained than using zero–sum methods, i.e., optimising for robot 1, then robot 2
  - $\epsilon$–NE found typically have $\epsilon=0$

$$\langle \langle \text{robot1:robot2} \rangle \rangle_{\text{max}} =? \ (P \ [F \leq k \ \text{goal}_1 ]+ P \ [F \leq k \ \text{goal}_2])$$
Faster and fairer equilibria

- Limitations of (social welfare) Nash equilibria for CSGs:
  1. can be **computationally expensive**, especially for >2 players
  2. social welfare optimality is **not** always equally beneficial

- **Correlated equilibria**
  - shared (probabilistic) signal
    + map to local strategies
  - synthesis: support enumeration
    + LP (>2 players needs nonLP for NE)
  - much faster to synthesise (4–20x faster)

- **Social fairness**
  - alternative optimality criterion: minimise **difference** in objectives
  - applies to both Nash/correlated: slight changes to optimisation

Example: Aloha communication protocol
Signals: randomised coordination of next message sender, adapting over time
Faster and fairer equilibria

- **Limitations of (social welfare) Nash equilibria for CSGs:**
  1. can be computationally expensive, especially for >2 players
  2. social welfare optimality is **not** always equally beneficial

- **Correlated equilibria**
  - shared (probabilistic) signal
    + map to local strategies
  - synthesis: support enumeration
    + LP (>2 players needs nonLP for NE)
  - much faster to synthesise (4–20x faster)

- **Social fairness**
  - alternative optimality criterion: minimise difference in objectives
  - applies to both Nash/correlated: slight changes to optimisation

---

**Example: Aloha communication protocol**

Social fairness (SF) more equitable than social welfare (WFᵢ)
Wrapping up
Summary

- Probabilistic model checking for stochastic games
  - turn-based and concurrent stochastic games
  - tools for modelling, construction & analysis of large games
  - temporal logics for property specification
  - value iteration based verification and strategy synthesis
  - wide range of interesting application domains & queries
Challenges & directions

- Partial information/observability
  - needed for practical applications
  - POSGs? DEC–POMDPs?

- Other game theory tools
  - e.g. Stackelberg equilibria

- Managing model uncertainty
  - learning + robust verification

- Accuracy of model checking results
  - value iteration improvements; exact methods

- Scalability & efficiency
  - e.g. symbolic methods, abstraction, symmetry reduction
  - sampling–based strategy synthesis methods
PRISM–games

- See the PRISM–games website for more info
  - prismmodelchecker.org/games/
  - documentation, examples, case studies, papers
  - downloads: 🍎 🐧 🍒
  - open source (GPLV2): 🐚