Multi-Agent Verification & Control with Probabilistic Model Checking

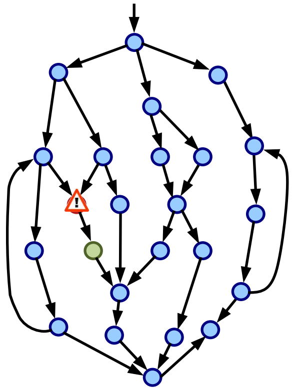
Dave Parker

University of Oxford

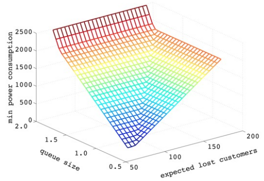
QEST @ CONFEST, Antwerp, Sep 2023

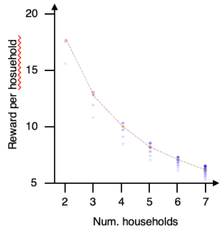
Probabilistic model checking

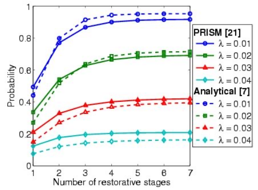
- Models & logics for automatic verification of stochastic systems
- Builds on an (increasingly) wide range of disciplines
  - logic, automata, Markov models, optimisation, SMT, simulation, control, AI, …
- Key strengths: exhaustive + numeric analysis
  - often subtle interplay between probability + nondeterminism
  - numerical results & trends can help identify flaws
  - enabled by advances in scalability, e.g., symbolic (BDD-based) methods
- Exploits flexibility of formal modelling languages & logics
  - consistency across wide range of models & properties



$P_{>0.999}$ [$\square$(*trigger* $\rightarrow$ $\diamond^{\leq 20}$ *deploy*)]



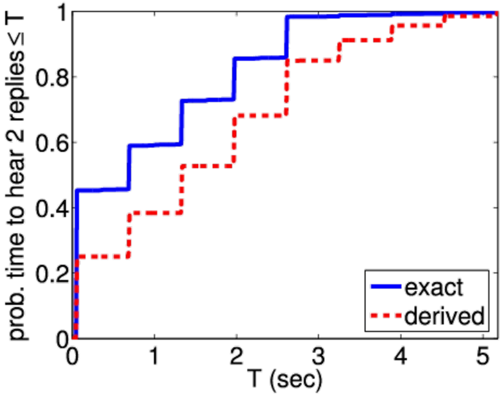




Example: Bluetooth

- Device discovery between a pair of Bluetooth devices
  - performance essential for this phase
- Complex discovery process
  - two asynchronous 28-bit clocks
  - pseudo-random hopping between 32 frequencies
  - random waiting scheme to avoid collisions

- Probabilistic model checking
  - worst-case expected time and probability for successful discovery
  - 17,179,869,184 initial configurations
  - exhaustive numerical analysis via symbolic model checking
  - highlights flaws in a simpler, analytic analysis

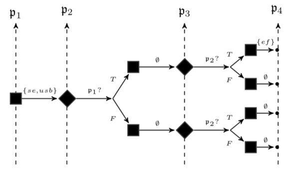


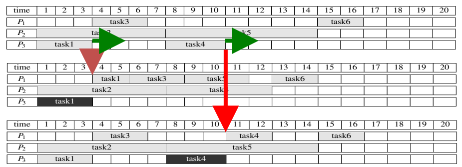
Trends in probabilistic model checking

- Increasingly expressive/powerful classes of model
  - real-time, partial observability, epistemic uncertainty, multi-agent, …
  - leading to ever widening range of application domains

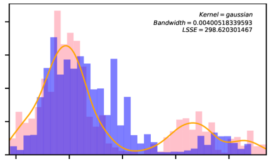
CTMC, CSG, DTMC, LTS, MDP, POMDP, POPTA, PTA, STPG, SMG, TPTG, IDTMC, IMDP

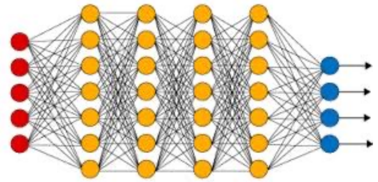
- From verification problems to control/synthesis
  - “correct-by-construction” from temporal logic specifications





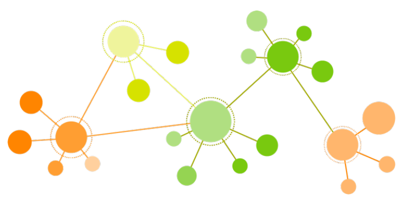
- Increasing use/integration of learning
  - either to support modelling/verification
  - or deployed within the systems being verified

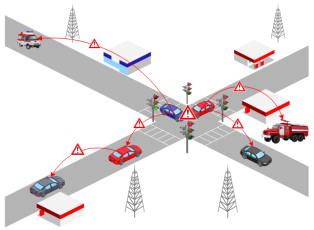




Stochastic multi-agent systems

- How do we verify/control stochastic systems with…
  - multiple agents acting autonomous and concurrently
  - competitive or collaborative behaviour between agents, possibly with differing goals
  - learnt components for e.g. control/perception





- Applications:
  - distributed protocols for consensus/security
  - multi-robot systems
  - autonomous vehicles

- This talk:
  - probabilistic model checking with stochastic multi-player games
  - models, logics, algorithms, tools, examples

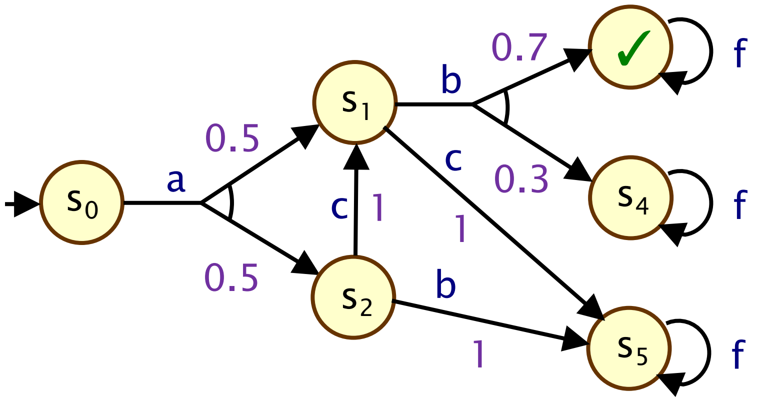
Overview

- Stochastic multi-player games
- Concurrent stochastic games
- Equilibria for stochastic games
- Neuro-symbolic games
- Challenges & directions

Stochastic games

Starting point: MDPs

- Markov decision processes (MDPs)
  - strategies (or policies) σ resolve actions based on history
  - e.g.: $P_{max=?}$ [F ✓] = $\sup_\sigma Pr_s^\sigma$(F ✓)
  - what is the maximum probability of reaching ✓ achievable by any strategy σ?



$\delta : S \times A \rightarrow Dist(S)$

- Key solution method: value iteration
  - values p(s) are the least fixed point of:

$$p(s) = \begin{cases} 1 & \text{if } s \models ✓ \\ \max_a \sum_{s'} \delta(s,a)(s') \cdot p(s') & \text{otherwise} \end{cases}$$

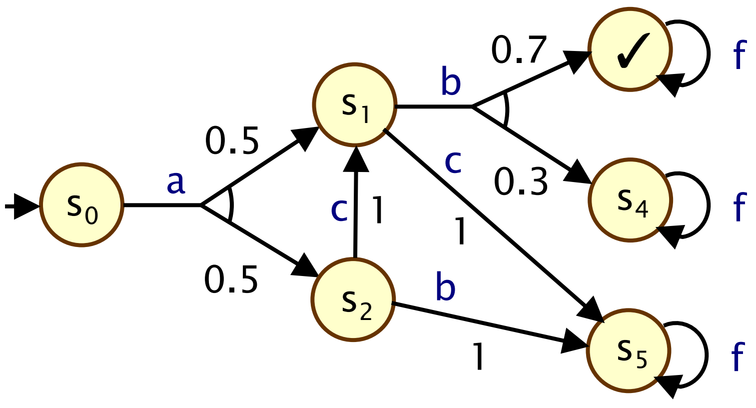
  - also amenable to symbolic (BDD-based) implementation

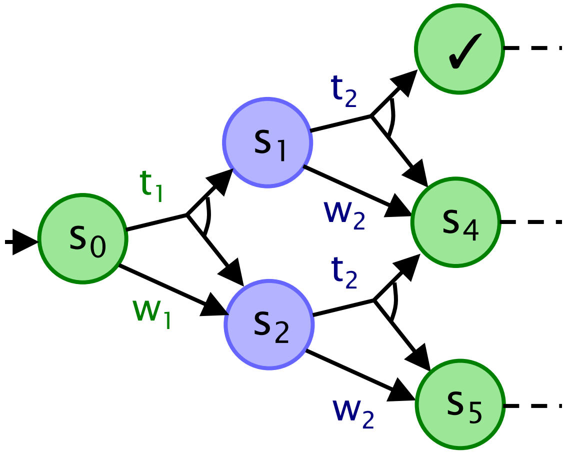
Stochastic multi-player games

- (Turn-based) stochastic multi-player games
  - strategies + probability + multiple players
  - player i controls subset of states S_i

Markov
decision processes
(MDPs)

Turn-based
stochastic games
(TSGs)





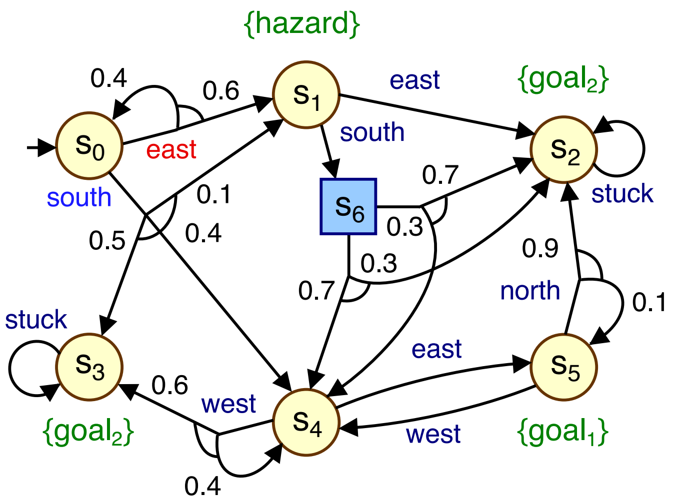
$\delta : S \times A \rightarrow \mathrm{Dist}(S)$

$S = S_1 \uplus \ldots \uplus S_n$

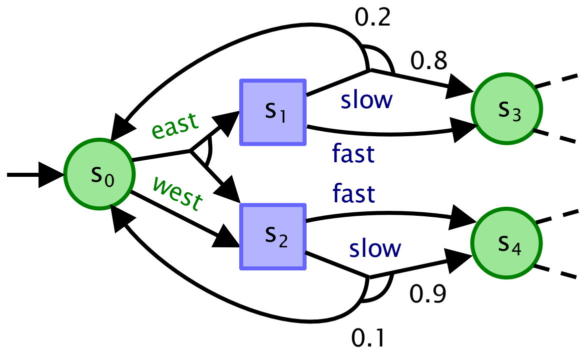
Modelling with turn-based games

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

Uncontrollable/unknown
navigation interference



Shared autonomy:
human-robot control



Property specification: rPATL

- rPATL (reward probabilistic alternating temporal logic)
  - zero-sum, branching-time temporal logic for stochastic games
  - coalition operator ⟨⟨C⟩⟩ of ATL
    + probabilistic (P) and reward (R) operators

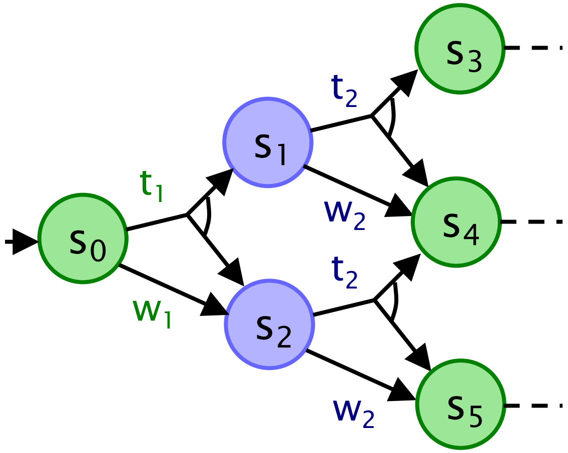
- Example:
  - $\langle\langle\{robot_1, robot_3\}\rangle\rangle\ P_{max=?}\ [\ F\ (goal_1 \vee goal_3)\ ]$
  - "what strategies for robots 1 and 3 maximise the probability of reaching their goal locations, regardless of the strategies of other players"

Can be seen as a mixture of control and verification

- Other additions:
  - (co-safe) linear temporal logic
    $\neg zone_3\ U\ (room_1 \wedge (F\ room_4 \wedge F\ room_5))$
  - nested specifications
    $\langle\langle\{robot_1, robot_3\}\rangle\rangle\ R_{min=?}\ [$
    $\langle\langle\{robot_1\}\rangle\rangle\ P_{\geq 0.99}\ [\ F^{\leq 10}\ base\ ]$
    $U\ (zone_1 \wedge (F\ zone_4))\ ]$
    "minimise expected time for joint task, while ensuring base reliably reached"

Model checking rPATL

- Main task: checking individual P and R operators
  - reduces to solving a (zero-sum) stochastic 2-player game
  - e.g. max/min reachability probability: $\sup_{\sigma_1} \inf_{\sigma_2} \Pr_s^{\sigma_1,\sigma_2}(F ✓)$
  - complexity: NP ∩ coNP (if we omit some reward operators)



- We again use value iteration
  - values p(s) are the least fixed point of:

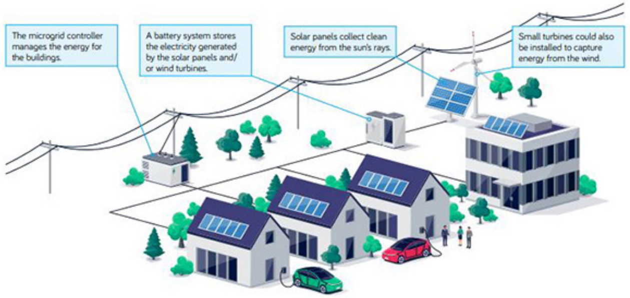
$$p(s) = \begin{cases} 1 & \text{if } s \models ✓ \\ \max_a \Sigma_{s'} \delta(s,a)(s') \cdot p(s') & \text{if } s \not\models ✓ \text{ and } s \in S_1 \\ \min_a \Sigma_{s'} \delta(s,a)(s') \cdot p(s') & \text{if } s \not\models ✓ \text{ and } s \in S_2 \end{cases}$$

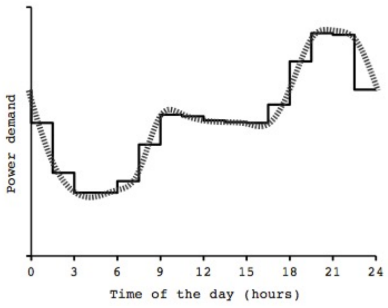
  - and more: graph-algorithms, sequences of fixed points, …

- Implementation
  - symbolic (BDD-based) version also developed
  - big gains on some models
  - also benefits for strategy compactness

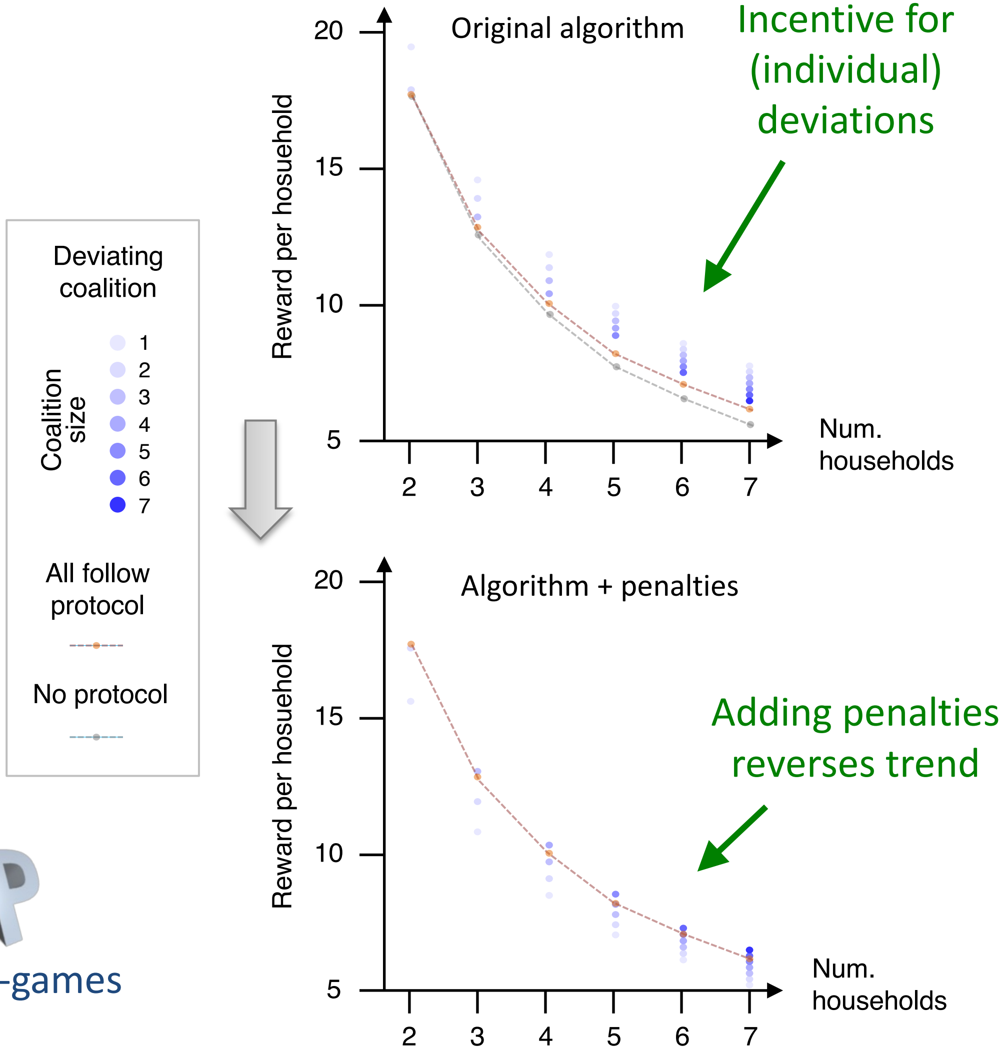
Example: Energy protocols

- Demand management protocol for microgrids
  - randomised back-off to minimise peaks
- Stochastic game model + rPATL
  - allow users to collaboratively cheat (ignore protocol)
  - TSGs of up to ~6 million states
  - exposes protocol weakness (incentive for clients to act selfishly)
  - propose/verify simple fix using penalties





PRISM-games



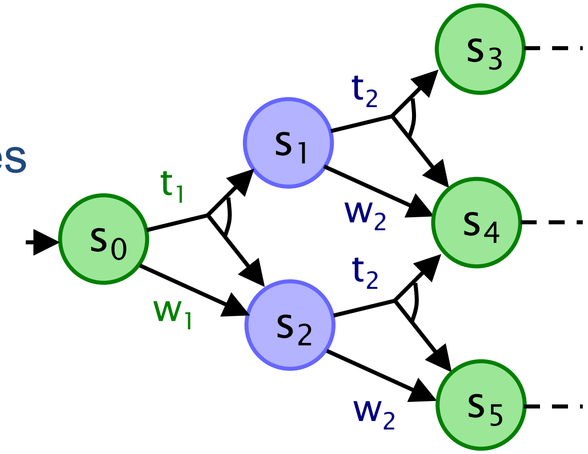
Concurrent stochastic games

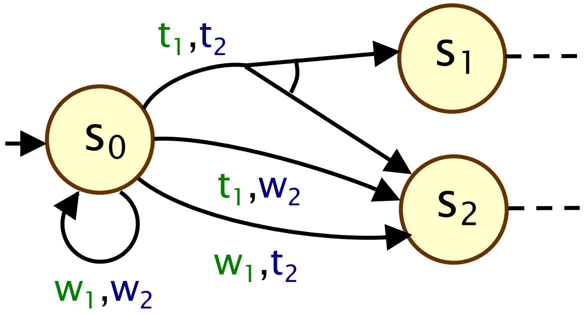
Concurrent stochastic games

- Need a more realistic model of components operating concurrently

- Concurrent stochastic games (CSGs)
  - (also known as Markov games, multi-agent MDPs)
  - players choose actions concurrently & independently
  - jointly determines (probabilistic) successor state

Turn-based stochastic games (TSGs)



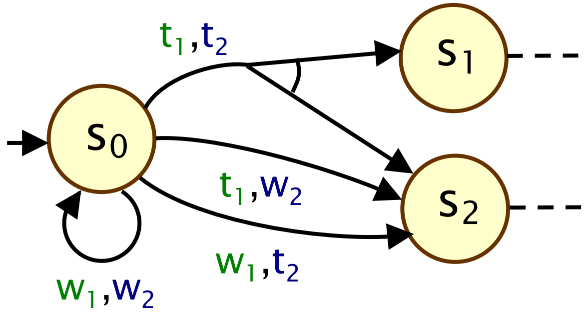


Concurrent stochastic games (CSGs)

$\delta : S \times (A_1 \cup \{\bot\}) \times \ldots \times (A_n \cup \{\bot\}) \rightarrow Dist(S)$

rPATL model checking for CSGs

- Same overall rPATL model checking algorithm
  - key ingredient is now solving (zero-sum) 2-player CSGs (PSPACE)
  - note that optimal strategies are now randomised



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

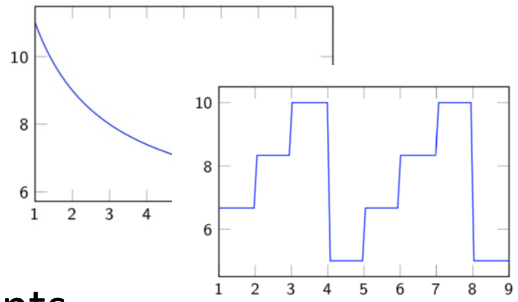
$$p(s) = \begin{cases} 1 & \text{if } s \models ✓ \\ val(Z) & \text{if } s \not\models ✓ \end{cases}$$

  - where Z is the matrix game with $z_{ij} = \Sigma_{s'}\ \delta(s,(a_i,b_j))(s')\cdot p(s')$

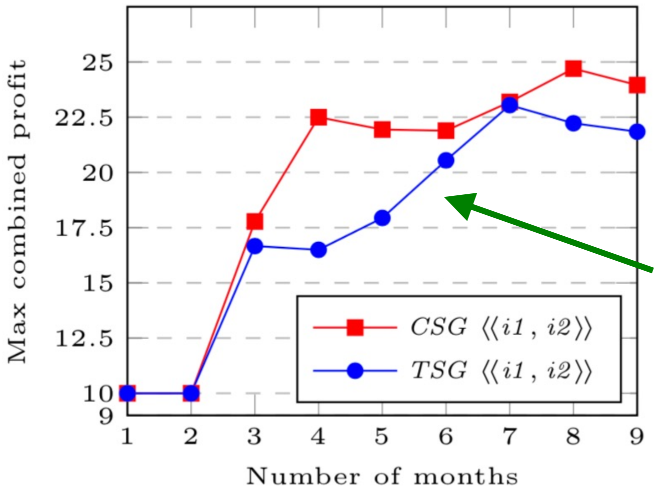
- Implementation
  - matrix games solved as linear programs
    - (LP problem of size |A|)
  - required for every iteration/state
    - which is the main bottleneck
  - but we solve CSGs of ~3 million states

Example: Future markets investor

- 3-player CSG modelling interactions between:
  - stock market, evolves stochastically
  - two investors i_1, i_2 decide when to invest
  - market decides whether to bar investors
  - various profit models; reduced for simultaneous investments



- Investor strategy synthesis via rPATL model checking
  - $\langle\langle investor_1, investor_2 \rangle\rangle\ R^{profit_{1,2}}_{max=?}\ [\ F\ finished_{1,2}\ ]$
  - non-trivial optimal (randomised) investment strategies
  - concurrent game (CSG) yields more realistic results (market has less observational power over investors)



Too pessimistic: unrealistic strategy for adversary

Equilibria for stochastic games

Equilibria-based properties

- Beyond zero-sum games:
  - players/components may have distinct objectives but which are not directly opposing (zero-sum)
- We use Nash equilibria (NE)
  - no incentive for any player to unilaterally change strategy
  - actually, we use ε-NE, which always exist for CSGs

$\sigma=(\sigma_1,\ldots,\sigma_n)$ is an ε-NE for objectives $X_1,\ldots,X_n$ iff:
for all i : $E_s^{\sigma}(X_i) \geq \sup \{ E_s^{\sigma'}(X_i) \mid \sigma'=\sigma_{-i}[\sigma_i'] \text{ and } \sigma_i' \in \Sigma_i \} - \varepsilon$

- We extend rPATL model checking for CSGs
  - with social-welfare Nash equilibria (SWNE)
  - i.e., NE which also maximise the joint sum $E_s^{\sigma}(X_1) + \ldots E_s^{\sigma}(X_n)$

Zero-sum properties

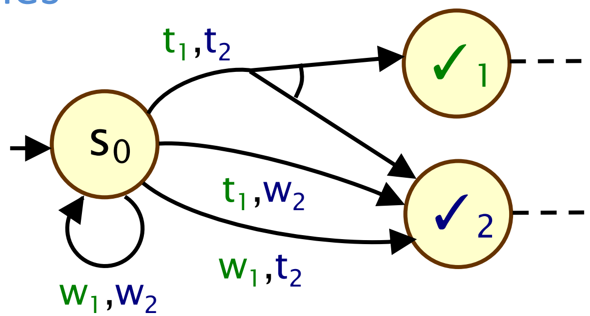
$\langle\langle robot_1 \rangle\rangle_{max=?} \, P\,[\, F^{\leq k}\, goal_1 \,]$

$\langle\langle robot_1{:}robot_2 \rangle\rangle_{max=?}$
$(P\,[\, F^{\leq k}\, goal_1 \,]+P\,[F^{\leq k}\, goal_2])$

Equilibria-based properties (SWNE)

Model checking for Nash equilibria

- Model checking for CSGs with equilibria
  - needs solution of bimatrix games
  - (basic problem is EXPTIME)
  - strategies need history and randomisation



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

- We further extend the value iteration approach:

$$p(s) = \begin{cases} (1,1) & \text{if } s \models \checkmark_1 \wedge \checkmark_2 \\ (1, p_{max}(s, \checkmark_2)) & \text{if } s \models \checkmark_1 \wedge \neg \checkmark_2 \\ (p_{max}(s, \checkmark_1), 1) & \text{if } s \models \neg \checkmark_1 \wedge \checkmark_2 \\ val(Z_1, Z_2) & \text{if } s \models \neg \checkmark_1 \wedge \neg \checkmark_2 \end{cases}$$

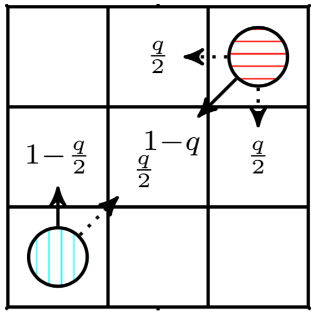
standard MDP analysis

bimatrix game

  - where Z_1 and Z_2 encode matrix games similar to before

Example: multi-robot coordination

- 2 robots navigating an m x m gridworld
  - start at opposite corners, goals are to navigate to opposite corners
  - obstacles modelled stochastically

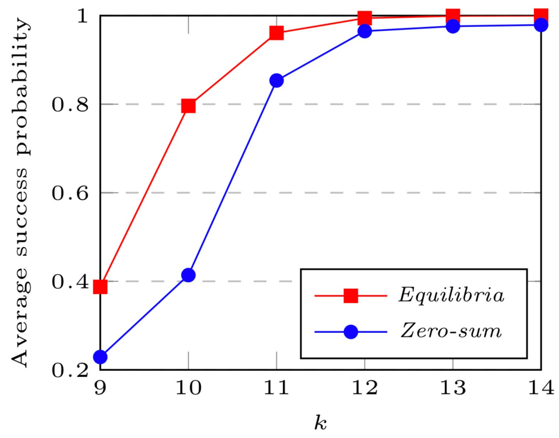


- We synthesise SWNEs to maximise the average probability of robots reaching their goals within time k
  - $\langle\langle robot1:robot2 \rangle\rangle_{max=?}$ (P [$F^{\leq k}$ $goal_1$]+P [$F^{\leq k}$ $goal_2$])
  - and compare to sequential strategy synthesis

Collaboration helps: better performance from equilibria

ε-NE found typically have ε=0

10 x 10 grid



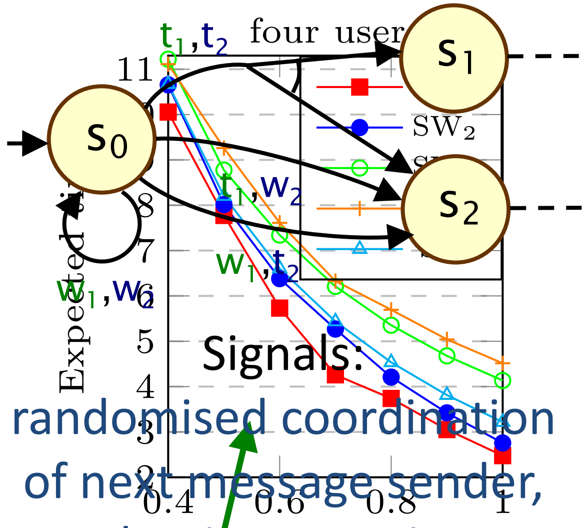
Faster and fairer equilibria

- 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 to players

Example: Aloha communication protocol

- Correlated equilibria
  - correlation: shared (probabilistic) signal + map to local strategies
  - synthesis: support enumeration + nonLP (Nash) -> LP (correlated)
  - experiments: 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



Signals: randomised coordination of next message sender, adapting over time

social fairness (SF) more equitable than social welfare (WF_i)

Tool support: PRISM-games

- PRISM-games
  - supports turn-based/concurrent SGs, zero-sum/equilibria
    - and more (co-safe LTL, multi-objective, real-time extensions, …)
  - explicit-state and symbolic implementations
  - custom modelling language extending PRISM

```
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
```

- Growing interest: other (TSG) tools becoming available
  - Tempest, EPMC, PET, PRISM-games extensions

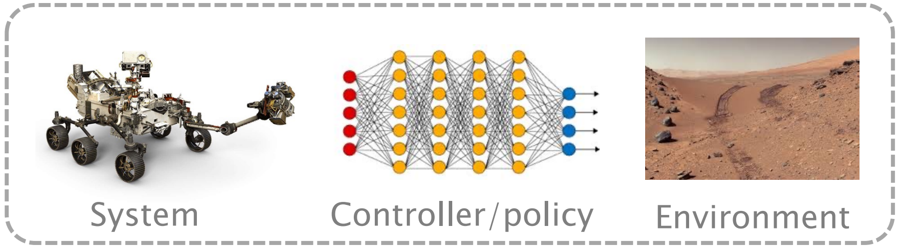
- Many other example application domains
  - attack-defence trees, self-adaptive software architectures, human-in-the-loop UAV mission planning, trust models, collective decision making, intrusion detection policies

prismmodelchecker.org/games/

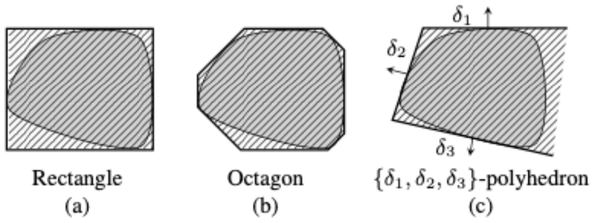
Neuro-symbolic games

Deep reinforcement learning

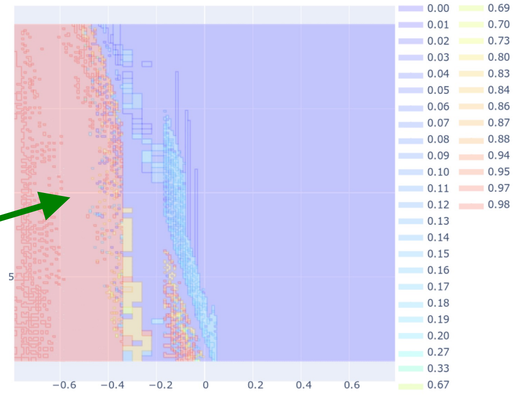
- Tackling more realistic problems
  - continuous state spaces & more complex dynamics
- Verification of learning-based systems
  - e.g., deep reinforcement learning
  - neural network (NN) learnt for strategy actions/values
- First steps: single-agent verification, fixed policy
  - deterministic dynamical system + control faults
  - combine polyhedral abstractions with probabilistic model checking
  - conservative abstraction of NN-controlled dynamics over a finite horizon, via MILP



deep reinforcement learning



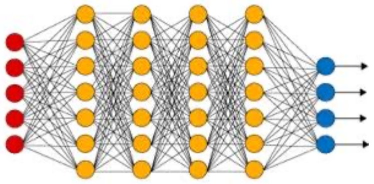
upper bounds on failure probabilities for initial regions

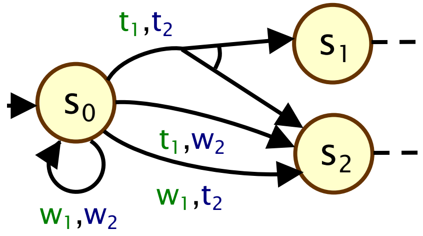


pendulum benchmark

Neuro-symbolic games

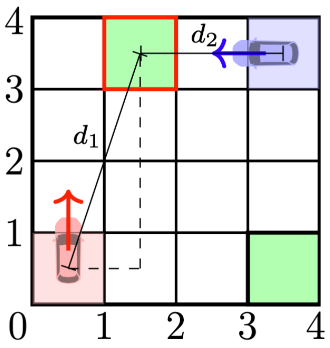
- Mixture of neural components + symbolic/logical components
  - simpler than end-to-end neural control problem; aids explainability
  - here: neural networks (or similar) for perception tasks
  - plus: local strategies for control decisions

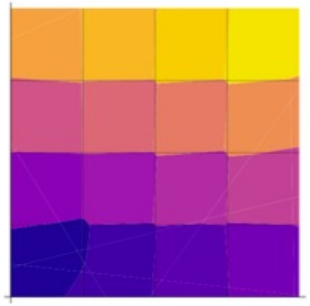




- Neuro-symbolic CSGs
  - finite-state agents + continuous-state environment E
    - $S = (Loc_1 \times Per_1) \times (Loc_2 \times Per_2) \times S_E$
  - agents use a (learnt) perception function to observe E
    - $obs_i : (Loc_1 \times Loc_2) \times S_E \rightarrow Per_i$
  - CSG-like joint actions update state probabilistically

- Example: dynamic vehicle parking
  - NN maps exact vehicle position to perceived grid cell

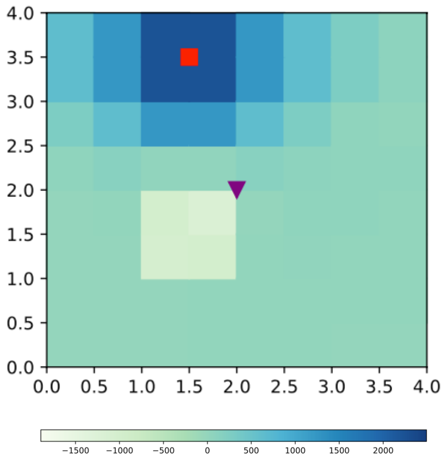




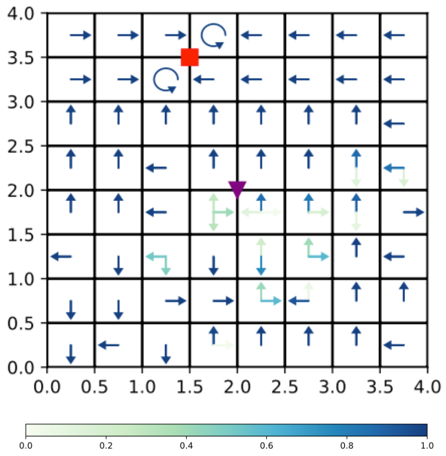
Model checking neuro-symbolic CSGs

- Strategy synthesis for zero-sum (discounted) expected reward
  - for now, we assume full observability
- Value iteration (VI) approach
  - continuous state-space decomposed into regions
  - further subdivision at each iteration
  - we define a class of piecewise-continuous value functions, preserved by NNs and VI
- Implementation
  - pre-image computations of NNs
  - polytope representations of regions
  - LPs to solve zero-sum games at each step

Dynamic vehicle parking with larger (8x8) grid and simpler (regression) perception



Value function (fragment)

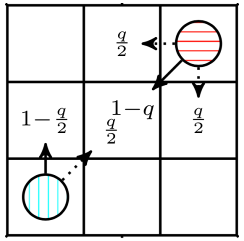


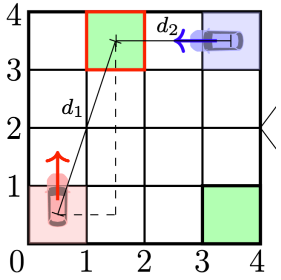
Optimal strategy (fragment)

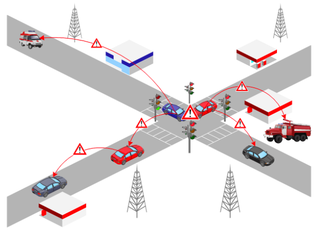
Wrapping up

III Challenges & directions

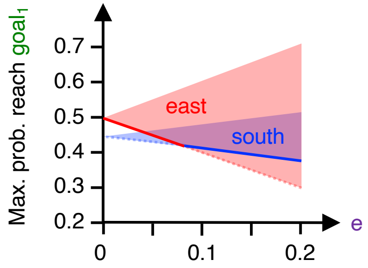
- Partial information/observability
  - e.g., leveraging progress on POMDPs







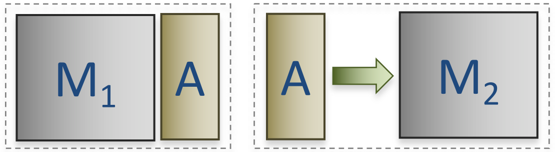
- Managing robustness and uncertainty
  - quantifying model uncertainty, e.g., from learning
  - stability of randomised strategies



- Modelling language design and extensions
  - e.g., more flexible interchange of components and strategies

```
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
```

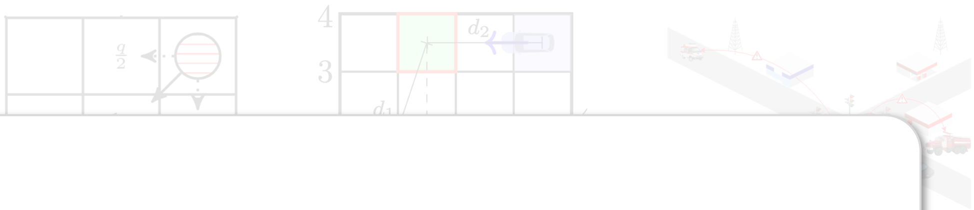
- Further classes of equilibria
  - e.g. Stackelberg equilibria for automotive/security applications



- Improving scalability & efficiency
  - e.g. symbolic methods for CSGs, compositional solution approaches

Challenges & directions

- Partial information/observability



- Joint work with:
  - Edoardo Bacci, Taolue Chen, Vojtěch Forejt, Marta Kwiatkowska, Gethin Norman, Gabriel Santos, Aistis Simaitis, Rui Yan

- Funded by: FUN2MODEL

European Research Council
Established by the European Commission

- More details here:

PRISM-games

prismmodelchecker.org/games/

  - e.g. symbolic methods for CSGs, compositional solution approaches