PRISM-Games

Seminar for the Computational Models for Complex Systems course

Pier Paolo Tarasco – (619622) – p.tarasco@studenti.unipi.it

20/05/2022

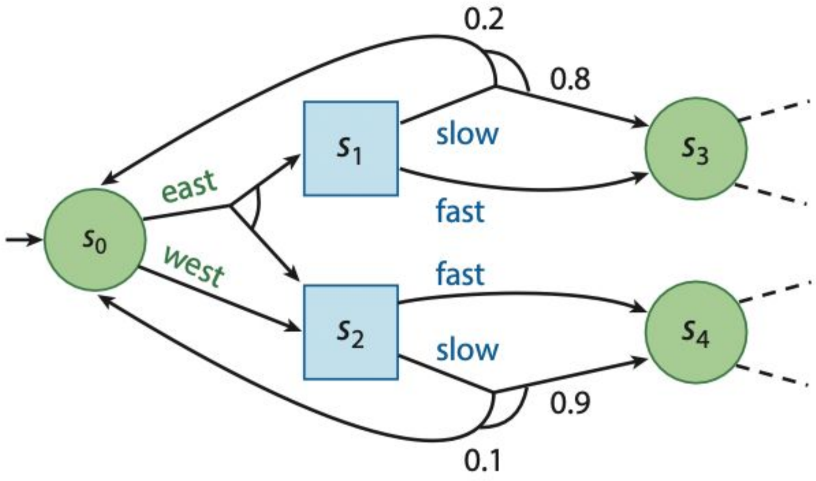
Outline

- Introduction to PRISM-Games
- Turn-based Stochastic multi-player games
- Studying trade-offs by Multi-objective strategies
- Demo: Verifying the NIM game
- Concurrent games
- Case study: UAV Modelling

An Introduction to PRISM-Games [1]

- Provides a tool to **verify** that a model satisfies some specifications
- We can also **synthesize** the optimal controller for a model
- PRISM-Games is an **extension** to the PRISM model checker to handle the behaviour of **multiple players** competing or cooperating to achieve their individual goals in the presence of adversarial or **uncertain** environments.
- There are many types of **Stochastic Games**, the first distinction is on the presence of a coordination:
    - **TSG (Also called SMG in PRISM)** → Turn-based game
    - **CSG** → Concurrent game

SMG: Turn-based Games [2]



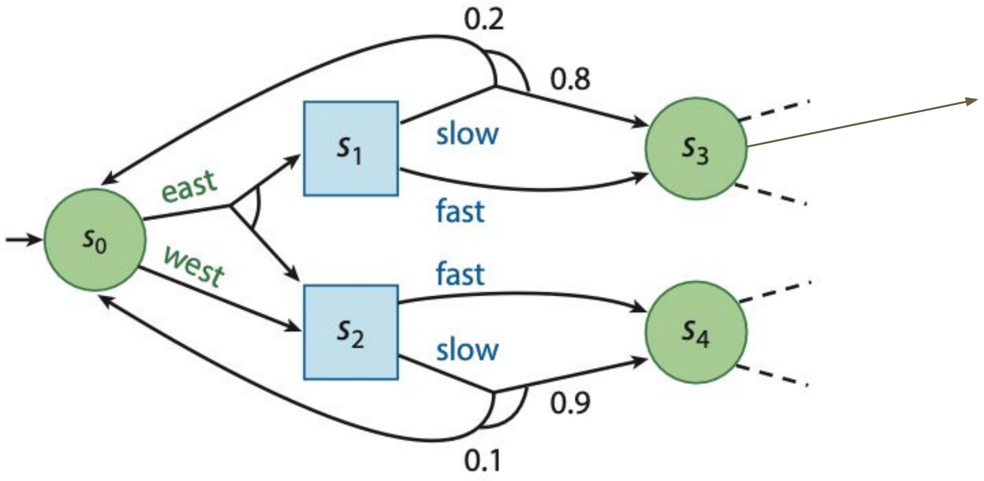
Definition 1 (SMG). A *stochastic multi-player game* (SMG) is a tuple $\mathcal{G} = (\Pi, S, (S_i)_{i\in\Pi}, \overline{s}, A, \delta, L)$, where:

- Π is a finite set of *players*,
- S is a finite set of *states*,
- $(S_i)_{i\in\Pi}$ is a partition of S,
- $\overline{s} \in S$ is an initial state,
- A is a finite set of *actions*,
- $\delta : S\times A \to Dist(S)$ is a (partial) probabilistic transition function,
- $L : S \to 2^{AP}$ is a labelling function mapping states to sets of atomic propositions from a set AP.

Labels: *goal* and *hazard*. L maps states into $2^{|\text{Labels}|}$ configurations

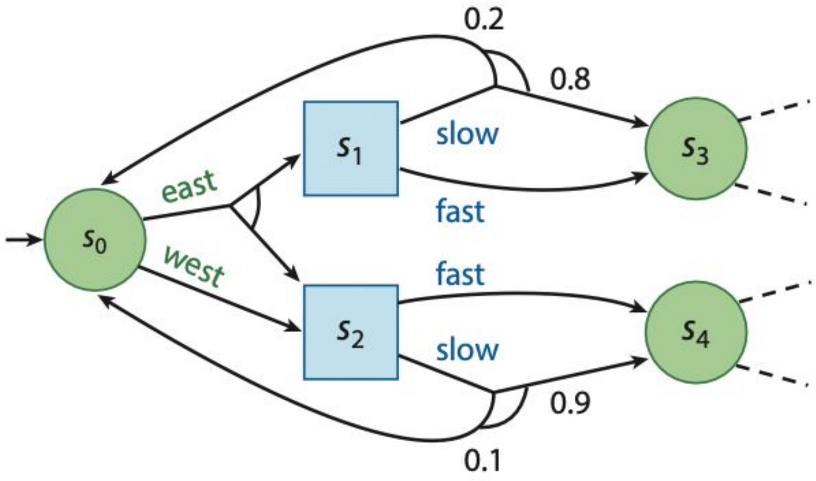
SMG: Turn-based Games [2]

Labels: *goal* and *hazard*. L maps states into $2^{|\text{Labels}|}$ configurations



goal = 0, *hazard* = 0
goal = 0, *hazard* = 1
goal = 1, *hazard* = 0
goal = 1, *hazard* = 1

SMG: Turn-based Games [2]



rPATL Logic

$$\phi ::= \texttt{true} \mid a \mid \neg\phi \mid \phi \wedge \phi \mid \langle\langle C \rangle\rangle \theta$$

$$\theta ::= \mathtt{P}_{\bowtie p}[\psi] \mid \mathtt{R}^{r}_{\bowtie x}[\mathtt{F}^{\star}\phi]$$

$$\psi ::= \mathtt{X}\,\phi \mid \phi\ \mathtt{U}^{\leqslant k}\ \phi \mid \phi\ \mathtt{U}\ \phi$$

SMG: Turn-based Games [2]

$$\phi ::= \texttt{true} \mid a \mid \neg\phi \mid \phi \wedge \phi \mid \langle\langle C \rangle\rangle \theta$$

A Property to satisfy by our model

Coalition of players

Quantitative objective that the set of players C try to satisfy

SMG: Turn-based Games [2]

$$\phi ::= \mathtt{true} \mid a \mid \neg\phi \mid \phi \wedge \phi \mid \langle\langle C \rangle\rangle \theta$$

$$\theta ::= \mathrm{P}_{\bowtie p}[\psi] \mid \mathrm{R}^{r}_{\bowtie x}[\mathrm{F}^{\star}\phi]$$

Objective

The probability of *some event* happening should meet the bound *p*

The expected amount of the reward *r* accumulated until the property ϕ is true meets the bound *x*

SMG: Turn-based Games [2]

$$\phi ::= \texttt{true} \mid a \mid \neg\phi \mid \phi \wedge \phi \mid \langle\langle C \rangle\rangle \theta$$

$$\theta ::= \mathrm{P}_{\bowtie p}[\,\psi\,] \mid \mathrm{R}^{r}_{\bowtie x}[\,\mathrm{F}^{\star}\phi\,]$$

c means to consider the accumulated reward along the whole path

* can take the values of 0, *c*, or Infinity and it indicates how to account for when a property ϕ cannot be reached

SMG: Turn-based Games [2]

$$\phi ::= \mathtt{true} \mid a \mid \neg\phi \mid \phi \wedge \phi \mid \langle\langle C \rangle\rangle \theta$$

$$\theta ::= \mathtt{P}_{\bowtie p}[\,\psi\,] \mid \mathtt{R}^{r}_{\bowtie x}[\,\mathtt{F}^{\star}\phi\,]$$

$$\psi ::= \mathtt{X}\,\phi \mid \phi\ \mathtt{U}^{\leqslant k}\ \phi \mid \phi\ \mathtt{U}\ \phi$$

Path formula

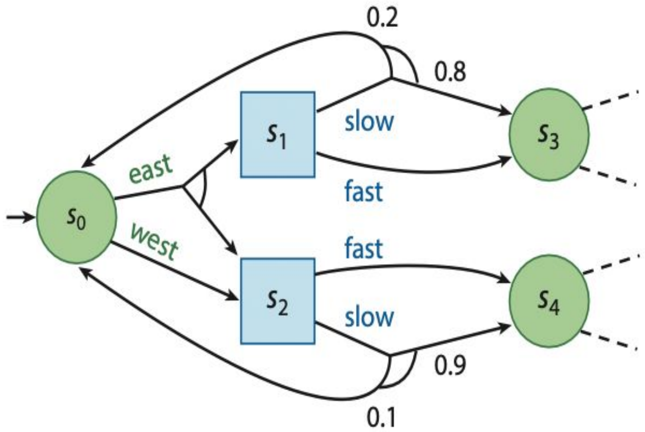
Note: we can derive the following:

$\mathtt{F}\ \phi \equiv \mathtt{true}\ \mathtt{U}\ \phi$ Eventually a property will be satisfied

$\mathtt{G}\ \phi \equiv \neg\mathtt{F}\ \neg\phi$ A property will always remain true

TSG Model Checking

- Note that the verification and strategy synthesis problems are the same since checking for a property reduces to show that there exists a strategy for one coalition of players that satisfies a property for all strategies of another coalition



$$\langle\langle \mathit{human} \rangle\rangle \mathrm{P}_{\geqslant 0.6}[\neg \mathsf{crash}\ \mathtt{U}\ \mathsf{target}]$$

$$\langle\langle \mathit{rbt} \rangle\rangle \mathrm{R}^{r_{\mathrm{steps}}}_{\geqslant 3.2}[\mathsf{F}\ \mathsf{target}]$$

Multi-objective specifications [2]

$$\phi ::= \mathtt{true} \mid a \mid \neg\phi \mid \phi \wedge \phi \mid \langle\langle C \rangle\rangle \theta$$

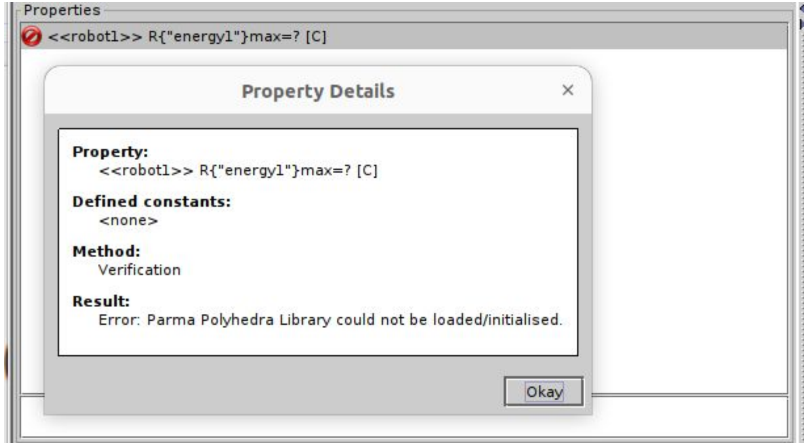
$$\theta ::= \mathtt{R}^{r}_{\bowtie x}[\,\mathtt{C}\,] \mid \mathtt{R}^{r}_{\bowtie x}[\,\mathtt{S}\,] \mid \mathtt{R}^{r/r}_{\bowtie x}[\,\mathtt{S}\,] \mid \mathtt{P}_{\geqslant 1}[\,\psi\,] \mid \neg\theta \mid \theta \wedge \theta$$

$$\psi ::= \mathtt{R}^{r}_{\bowtie x}[\,\mathtt{S}\,] \mid \mathtt{R}^{r/r}_{\bowtie x}[\,\mathtt{S}\,]$$

$\mathtt{R}^{r}_{\bowtie x}[\,\mathtt{C}\,]$	Total reward
$\mathtt{R}^{r}_{\bowtie x}[\,\mathtt{S}\,]$	Long-run average reward
$\mathtt{R}^{r/r}_{\bowtie x}[\,\mathtt{S}\,]$	Long-run *ratio* of 2 rewards
$\mathtt{P}_{\geqslant 1}[\,\psi\,]$	Almost sure satisfaction for total and long-run mean rewards

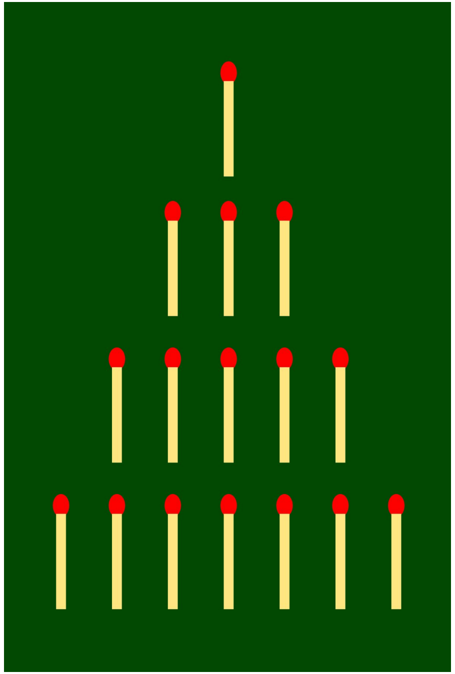
Examples of multi-objective specifications

$$\langle\langle\{robot_1\}\rangle\rangle(\mathrm{R}^{energy_1}_{\leqslant e_1}[\mathrm{C}] \wedge \mathrm{R}^{tasks_1}_{\geqslant t_1}[\mathrm{C}])$$



Robot 1 has a strategy that allows it to complete on average *t1* tasks while using less than *e1* energy

Example: Verifying the game of NIM



- The game consists of a **predetermined number of piles** (here shown as rows)
- Each pile/row consists a **variable number of objects** chosen at the start (here shown as matches)
- **Players take turns** removing at least 1 match from a pile (but they can take any number they want until completely removing the pile)
- **Wins who removes the last match**

NIM Optimal Playing Strategy [4]

- Given any initial configuration it is possible to determine who will win
- The proof is based on representing for each pile its number of matches as a binary number
- We then sum the binary number without accounting for carry-overs → This is called the *nim sum*
- A configuration is said to be **safe if the nim sum is 0** otherwise its **unsafe**
- The winning strategy is to always leave the opponent in a **safe configuration**
- Note: we can always reach a safe state but starting from a safe state we can only reach an unsafe state
- So, if we start from an unsafe configuration we can win

Modelling by iteratively checking properties

```
smg

player player1 ply1, [remove_pile1], [pass_turn1] endplayer
player player2 ply2, [remove_pile2], [pass_turn2] endplayer

module ply1

        pile_chosen1 : [0..3] init 0;

        [remove_pile1] turn = 1 & pile1 > 0 & (pile_chosen1 = 0 | pile_chosen1=1) -> (pile_chosen1'=1);
        [pass_turn1] pile_chosen1 > 0 -> (pile_chosen1'=0);
endmodule


module ply2
        pile_chosen2 : [0..3] init 0;

        [remove_pile2] turn = 2 & pile1 > 0 & (pile_chosen2 = 0 | pile_chosen2=1) -> (pile_chosen2'=1);
        [pass_turn2] pile_chosen2 > 0 -> (pile_chosen2'=0);

endmodule


module NIM
        pile1 : [0..3] init 3;
        pile2 : [0..3] init 0;
        pile3 : [0..3] init 0;
        turn : [1..2] init 1;
        winner : [0..2] init 0;

        [remove_pile1] pile1 > 0 -> (pile1'=pile1-1);
        [pass_turn1] true -> (turn'=2);

        [remove_pile2] pile1 > 0 -> (pile1'=pile1-1);
        [pass_turn2] true -> (turn'=1);

        // Wins who remove the last element
        [ ] pile1 = 0 & pile2 = 0 & pile3 = 0 -> (winner' = turn);
endmodule

```

Property Details

Property:
<<player1>> Pmax=? [F winner=1]

Defined constants:
<none>

Method:
Verification

Result:
1.0 (value in the initial state)

Okay

Property Details

Property:
<<player2>> Pmax=? [F winner = 2]

Defined constants:
<none>

Method:
Verification

Result:
0.0 (value in the initial state)

Okay

Property Details

Property:
<<player1,player2>> Pmax=? [F winner=2]

Defined constants:
<none>

Method:
Verification

Result:
1.0 (value in the initial state)

Okay

```
smg

player player1 ply1, [remove_pile1_1], [remove_pile2_1], [remove_pile3_1], [pass_turn1] endplayer
player player2 ply2, [remove_pile1_2], [remove_pile2_2], [remove_pile3_2], [pass_turn2] endplayer

module ply1
	pile_chosen1 : [0..3] init 0;

	[remove_pile1_1] turn = 1 & pile1 > 0 & (pile_chosen1 = 0 | pile_chosen1=1) -> (pile_chosen1'=1);
	[remove_pile2_1] turn = 1 & pile2 > 0 & (pile_chosen1 = 0 | pile_chosen1=2) -> (pile_chosen1'=2);
	[remove_pile3_1] turn = 1 & pile3 > 0 & (pile_chosen1 = 0 | pile_chosen1=3) -> (pile_chosen1'=3);
	[pass_turn1] pile_chosen1 > 0 -> (pile_chosen1'=0);
endmodule


module ply2
	pile_chosen2 : [0..3] init 0;

	[remove_pile1_2] turn = 2 & pile1 > 0 & (pile_chosen2 = 0 | pile_chosen2=1) -> (pile_chosen2'=1);
	[remove_pile2_2] turn = 2 & pile2 > 0 & (pile_chosen2 = 0 | pile_chosen2=2) -> (pile_chosen2'=2);
	[remove_pile3_2] turn = 2 & pile3 > 0 & (pile_chosen2 = 0 | pile_chosen2=3) -> (pile_chosen2'=3);
	[pass_turn2] pile_chosen2 > 0 -> (pile_chosen2'=0);
endmodule


module NIM
	pile1 : [0..10] init 3;
	pile2 : [0..10] init 1;
	pile3 : [0..10] init 7;
	turn : [1..2] init 1;
	winner : [0..2] init 0;

	[remove_pile1_1] pile1 > 0 -> (pile1'=pile1-1);
	[remove_pile1_2] pile1 > 0 -> (pile1'=pile1-1);

	[remove_pile2_1] pile2 > 0 -> (pile2'=pile2-1);
	[remove_pile2_2] pile2 > 0 -> (pile2'=pile2-1);

	[remove_pile3_1] pile3 > 0 -> (pile3'=pile3-1);
	[remove_pile3_2] pile3 > 0 -> (pile3'=pile3-1);

	[pass_turn1] true -> (turn'=2);
	[pass_turn2] true -> (turn'=1);

	// Wins who remove the last element
	[ ] pile1 = 0 & pile2 = 0 & pile3 = 0 -> (winner' = turn);
endmodule
```

Property Details

Property:
<<player1>> Pmax=? [F winner=1]

Defined constants:
<none>

Method:
Verification

Result:
1.0 (value in the initial state)

Okay

Property Details

Property:
<<player2>> Pmax=? [F winner = 2]

Defined constants:
<none>

Method:
Verification

Result:
0.0 (value in the initial state)

Okay

Property Details

Property:
<<player1,player2>> Pmax=? [F winner=2]

Defined constants:
<none>

Method:
Verification

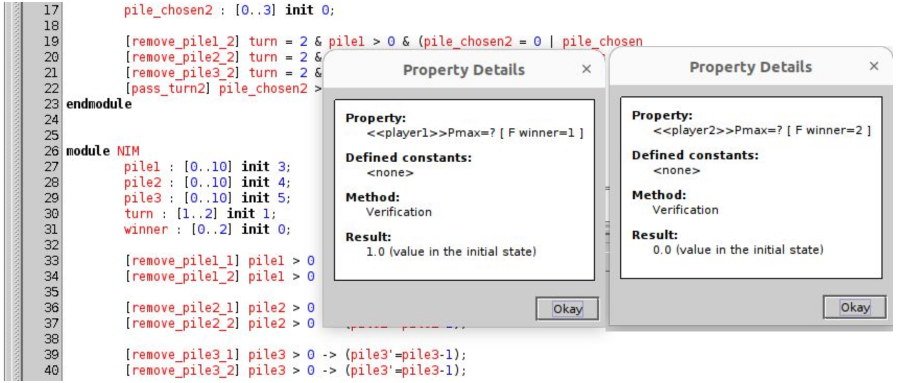
Result:
1.0 (value in the initial state)

Okay

Sanity check with Optimal Strategy [4]

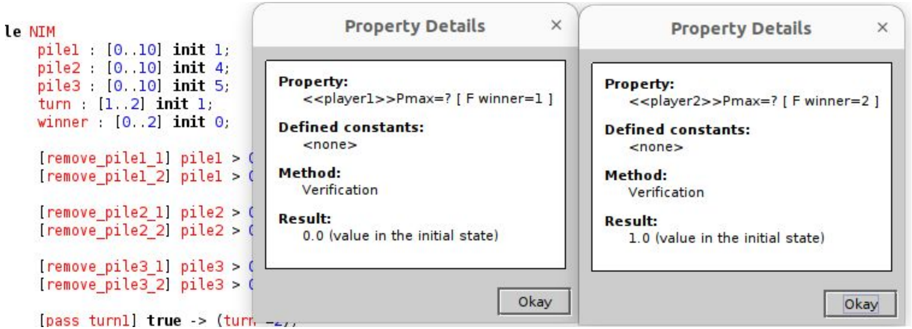
Unsafe configuration → Player 1 can win

```
Pila 1: 3 elementi       o o o
Pila 2: 4 elementi      o o o o
Pila 3: 5 elementi     o o o o o

3₁₀ = 011₂
4₁₀ = 100₂
5₁₀ = 101₂

  0 1 1
+ 1 0 0
+ 1 0 1
-------
  0 1 0
```

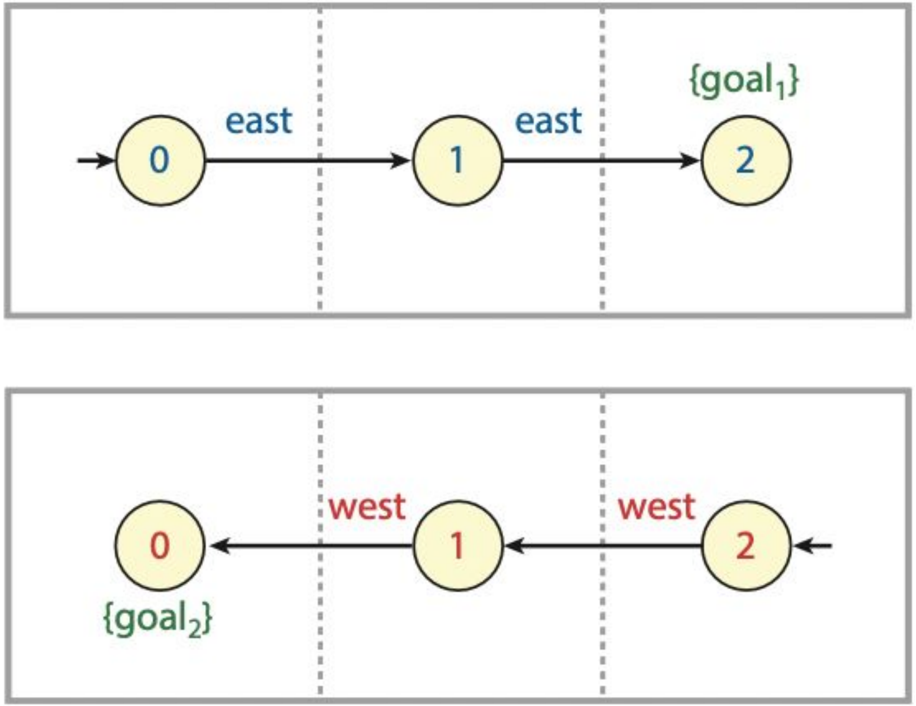


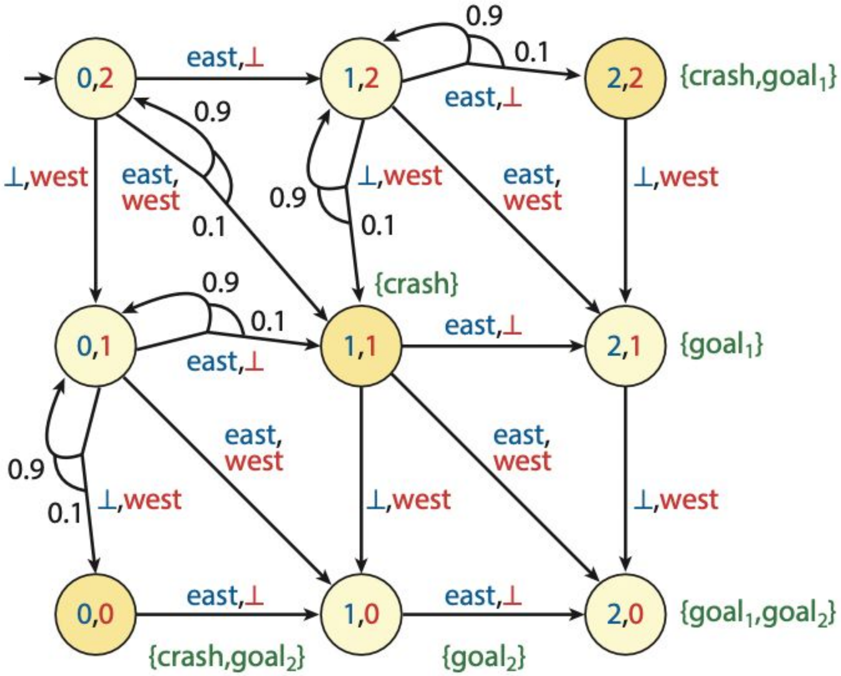
Safe configuration → Player 1 cannot win

```
Pila 1: 1 elemento         o
Pila 2: 4 elementi      o o o o
Pila 3: 5 elementi     o o o o o

1₁₀ = 001₂
4₁₀ = 100₂
5₁₀ = 101₂

    0 0 1
+   1 0 0
+   1 0 1
--------
    0 0 0
```



Concurrent Stochastic Games [1]





Property specification for CSG

$$\Phi \;:=\; \langle\langle C\rangle\rangle \mathrm{P}_{\bowtie p}[\psi] \;\mid\; \langle\langle C\rangle\rangle \mathrm{R}^{r}_{\bowtie q}[\rho] \;\mid\; \langle\langle C_1{:}\cdots{:}C_m\rangle\rangle_{\mathrm{opt}\bowtie q}(\theta),$$
$$\theta \;:=\; \mathrm{P}[\psi]+\cdots+\mathrm{P}[\psi] \;\mid\; \mathrm{R}^{r}[\rho]+\cdots+\mathrm{R}^{r}[\rho],$$

Non-zero sum formula

- *m* sets of coalitions of players
- Each coalition *i* will try to satisfy the objective *i*
- A **non-zero sum** formula is satisfied if:
  There exists an optimal strategy such that:
  1. no coalition C_i can deviate from the optimal strategy to improve their objective
  2. There is no other strategy which improves the sum of the local objectives
  3. The sum of the local objectives satisfies the bound *q*

social welfare–optimal Nash equilibrium (SWNE)

Examples of property specification for CSG

What is the maximum probability that *robot1* will eventually reach its goal without crashing? (no matter the behaviour of the second robot)

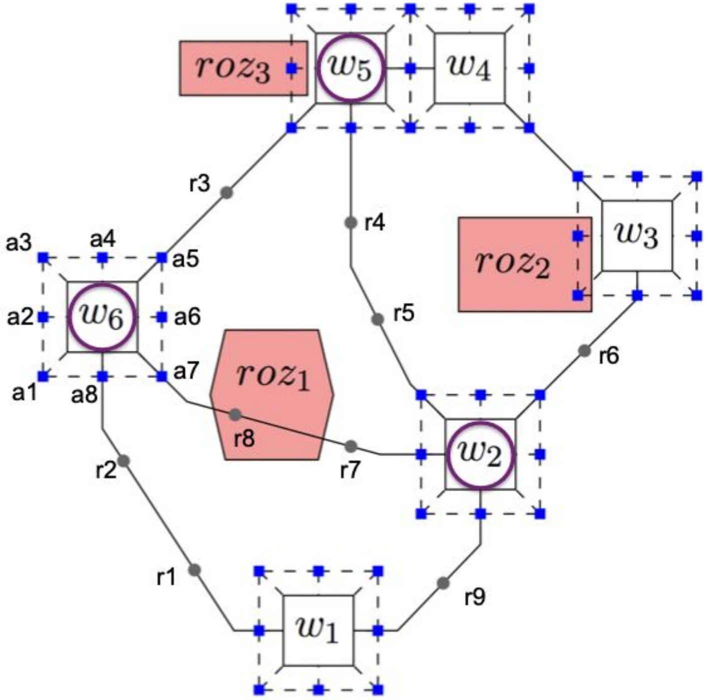
$$\langle\langle rbt_1 \rangle\rangle \mathrm{P}_{\max=?}[\neg \mathsf{crash}\ \mathtt{U}\ \mathsf{goal}_1]$$

Can the two robots collaborate such that both will reach their goals while the *robot2* does not crash and completes it goal in at most 10 steps?

$$\langle\langle rbt_1{:}rbt_2 \rangle\rangle_{\max\geqslant 2}(\mathtt{P}[\mathtt{F}\ \mathsf{goal}_1]+\mathtt{P}[\neg \mathsf{crash}\ \mathtt{U}^{\leqslant 10}\mathsf{goal}_2])$$

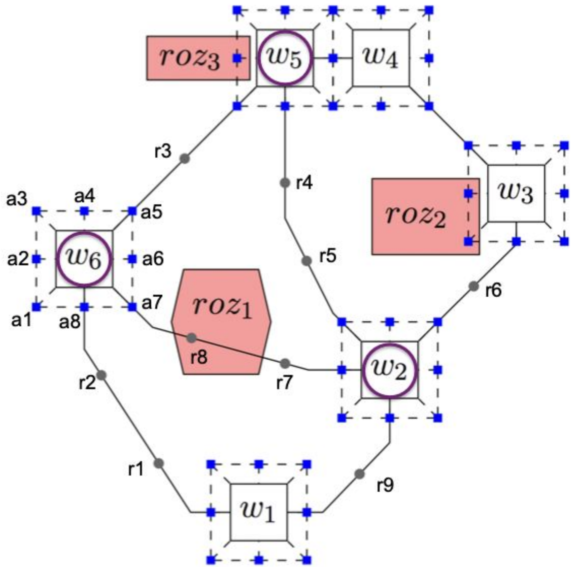
Case study: UAV Modelling [3]

- Study a road network surveillance by an unmanned aerial vehicle (UAV)
- 2 players: the human operator and the UAV
- Goal: Construct an optimal controller for the UAV under any possible operator behaviour
- The human operator can still decide where to go in states: w2, w5 and w6



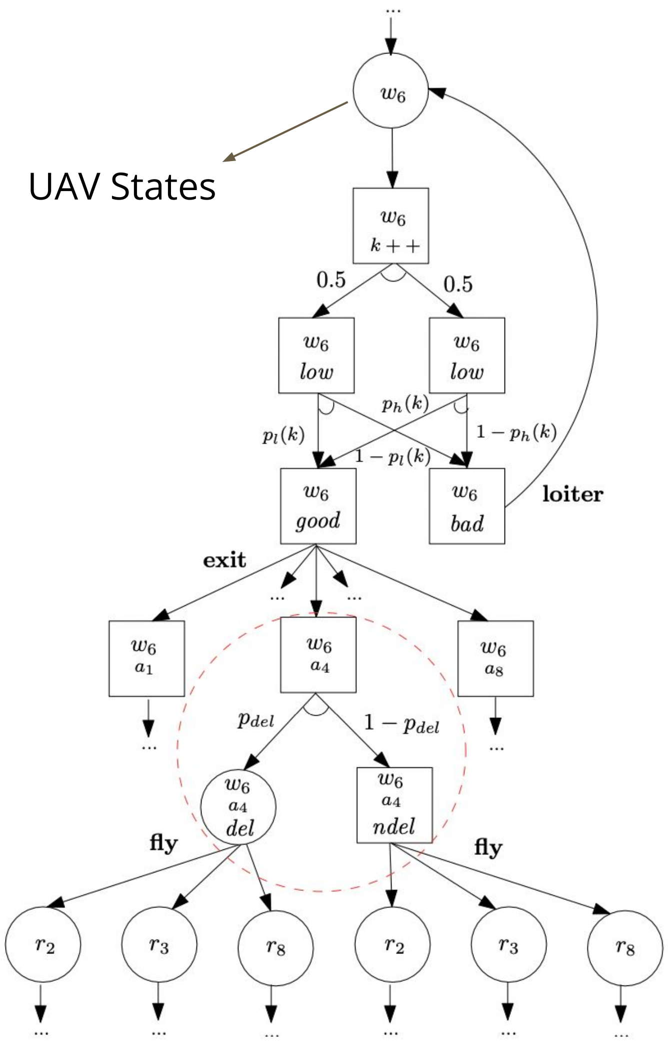
Problem: We sometimes not want a perfect adversarial

In case of verifying a property for the UAV, the adversarial operator could ask the UAV to fly in the loop w2, w6, w5, w2, w6, ... forever

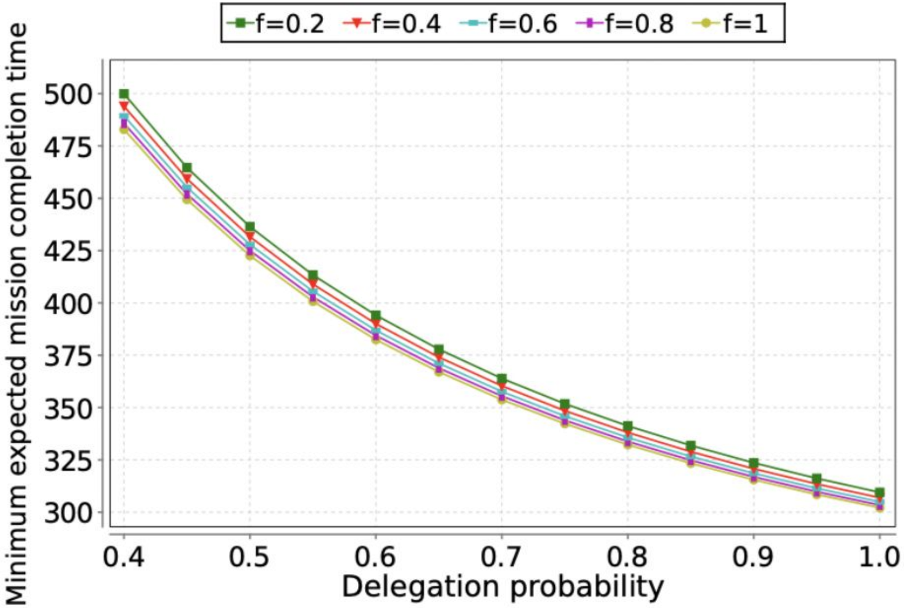


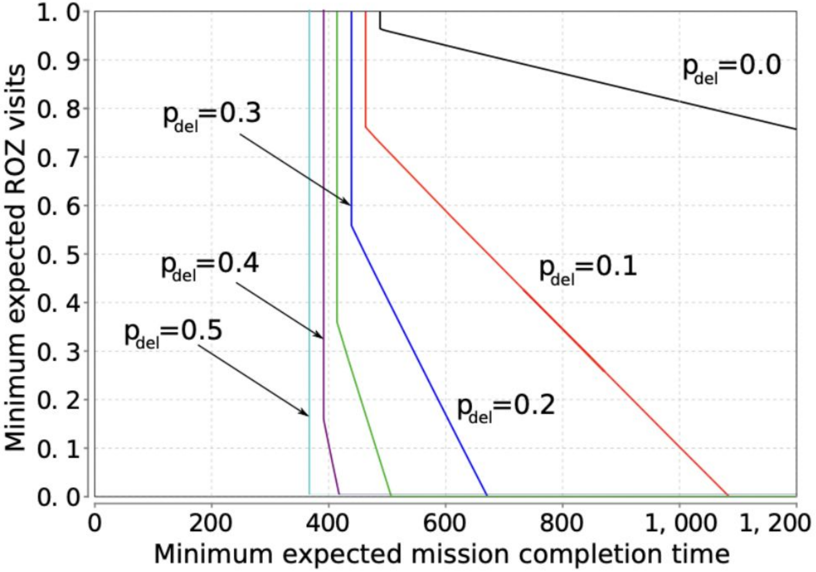
Solution: Allow for delegation

- At checkpoints where the human can choose where to go we only allow this to happen with a probability p_{del}
- This solution will prevent from infinite adversarial paths to emerge allowing for realistic strategies



Examples of model checking





Conclusions

- Statistical model checking is a powerful tool which can help us design correct models and also explore different trade-off by studying the influence of model parameters
- However we need to be very careful into designing the correct model!
- Also, it’s easier to use when we already know or at least have a general idea of the results we’ll obtain since they provide a reality check
- Unfortunately the tool is *not* easy to install and use properly at first

References

[1] M. Kwiatkowska et al, Probabilistic Model Checking and Autonomy, 2022

[2] M. Kwiatkowska et al, PRISM-games: Verification and Strategy Synthesis for Stochastic Multi-player Games with Multiple Objectives, 2018

[3] L. Feng, Controller Synthesis for Autonomous Systems Interacting With Human Operators, 2015

[4] https://it.wikipedia.org/wiki/Nim

Thank you for your attention!

You can find this presentation and the code at: https://github.com/Pier297/PRISM-Games