Strategic Verifier (STV): Towards Practical Verification of Strategic Ability

---

Damian Kurpiewski

14/05/2024

Institute of Computer Science
Polish Academy of Sciences

Faculty of Mathematics and Computer Science
Nicolaus Copernicus University in Toruń

Model Checking of Strategic Abilities

ATL: What Agents Can Achieve

- ATL: Alternating-time Temporal Logic [Alur et al. 1997-2002]
- Temporal logic meets game theory
- Main idea: cooperation modalities

$\langle\langle A \rangle\rangle\Phi$: coalition A has a collective strategy to enforce Φ

$\leadsto$ Φ can include temporal operators: X (next), F (sometime in the future), G (always in the future), U (strong until)

ATL with incomplete information

- Imperfect information ($q \sim_a q'$)

ATL with incomplete information

- Imperfect information ($q \sim_a q'$)
- Imperfect recall - agent memory coded within state of the model

ATL with incomplete information

- Imperfect information ($q \sim_a q'$)
- Imperfect recall - agent memory coded within state of the model
- Uniform strategies - specify same choices for indistinguishable states:
  $q \sim_a q' \implies s_a(q) = s_a(q')$

ATL with incomplete information

- Imperfect information ($q \sim_a q'$)
- Imperfect recall - agent memory coded within state of the model
- Uniform strategies - specify same choices for indistinguishable states: $q \sim_a q' \implies s_a(q) = s_a(q')$
- Fixpoint equivalences **do not hold** anymore

ATL with incomplete information

- Imperfect information ($q \sim_a q'$)
- Imperfect recall - agent memory coded within state of the model
- Uniform strategies - specify same choices for indistinguishable states:
  $q \sim_a q' \implies s_a(q) = s_a(q')$
- Fixpoint equivalences **do not hold** anymore
- Model checking **ATL**$_{\mathrm{ir}}$ is Δ_2^p-complete

Example: Simple Model of Voting and Coercion

- Two agents: the Voter and the Coercer
- Two candidates

Example: Simple Model of Voting and Coercion

- Two agents: the Voter and the Coercer
- Two candidates

- Voter can cast her vote and then interact with the Coercer
- Voter can give (or not) her vote to the Coercer

Example: Simple Model of Voting and Coercion

- Two agents: the Voter and the Coercer
- Two candidates

- Voter can cast her vote and then interact with the Coercer
- Voter can give (or not) her vote to the Coercer

- Coercer can punish (or not) the voter

Example: Simple Model of Voting and Coercion

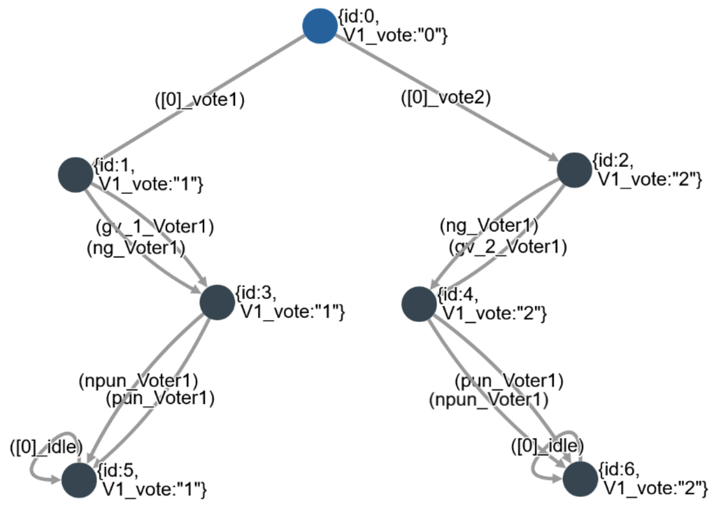
- Two agents: the Voter and the Coercer
- Two candidates

- Voter can cast her vote and then interact with the Coercer
- Voter can give (or not) her vote to the Coercer

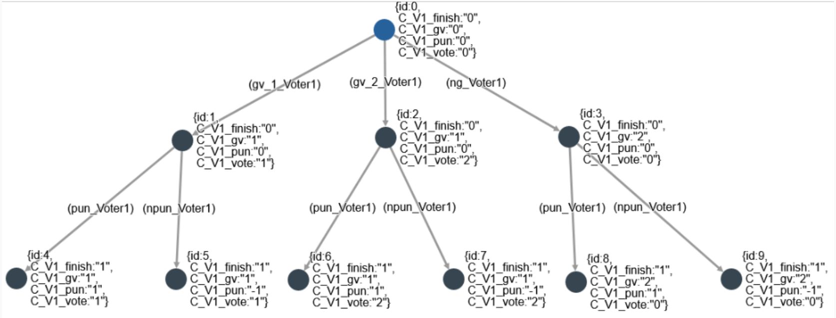
- Coercer can punish (or not) the voter

- Asynchronous semantics with synchronization over actions: vote giving and punishment are **synchronized**

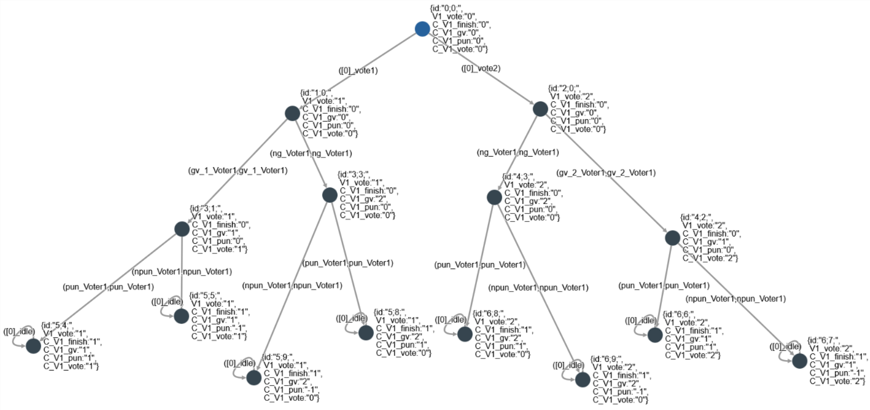
Example: Simple Model of Voting and Coercion
Voter Local Model



Example: Simple Model of Voting and Coercion
Coercer Local Model

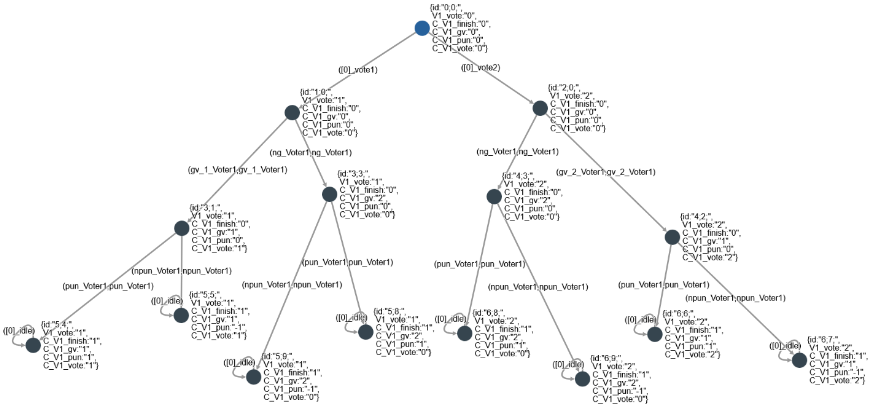


Example: Simple Model of Voting and Coercion
Global Model



Example: Simple Model of Voting and Coercion

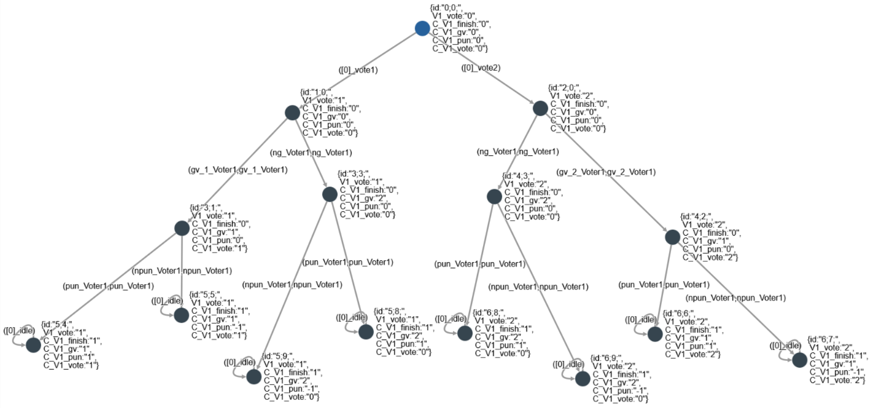
Example Formula



$\langle\!\langle \mathit{coercer} \rangle\!\rangle \mathrm{G}(\mathrm{finish}_1 \wedge \mathrm{vote}_{1,1} \implies \neg \mathrm{pun}_1)$:

“The Coercer can coerce Voter to vote for the first candidate”

Example: Simple Model of Voting and Coercion
Example Formula

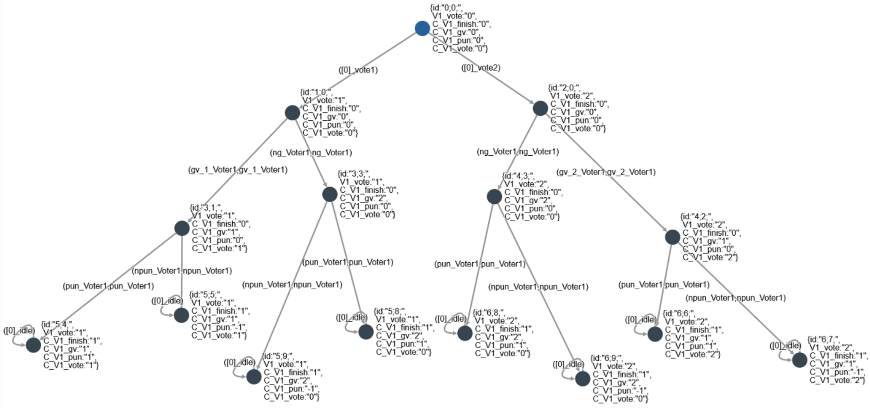


$\langle\langle \mathit{coercer} \rangle\rangle \mathrm{G}(\mathrm{finish}_1 \wedge \mathrm{vote}_{1,1} \implies \neg \mathrm{pun}_1)$:

"The Coercer can coerce Voter to vote for the first candidate"

FALSE

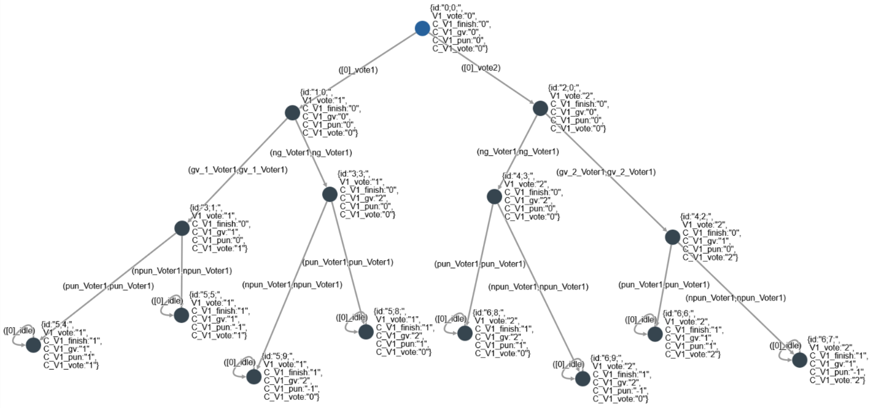
Example: Simple Model of Voting and Coercion
Example Formula



$\langle\langle \mathit{coercer} \rangle\rangle \mathrm{G}(\mathrm{finish}_1 \wedge \mathrm{vote}_{1,1} \implies K_C \mathrm{vote}_{1,1})$:

“The Coercer knows when the Voter has voted for the first candidate”

Example: Simple Model of Voting and Coercion
Example Formula



$\langle\langle \mathit{coercer} \rangle\rangle \mathrm{G}(\mathrm{finish}_1 \wedge \mathrm{vote}_{1,1} \implies K_C \mathrm{vote}_{1,1})$:

"The Coercer knows when the Voter has voted for the first candidate"

FALSE

Simple Specification Language

Simple Voting Model

```
Agent Voter1:
LOCAL: [V1_vote]
PERSISTENT: [V1_vote]
INITIAL: []
init q0
vote1: q0 -> q1 [V1_vote:=1]
vote2: q0 -> q1 [V1_vote:=2]
shared[2] gv_1_Voter1[gv_1_Voter1]: q1 [V1_vote==1] -> q2
shared[2] gv_2_Voter1[gv_1_Voter2]: q1 [V1_vote==2] -> q2
shared[2] ng_Voter1[ng_Voter1]: q1 -> q2
shared[2] pun_Voter1[pn_Voter1]: q2 -> q3
shared[2] npun_Voter1[pn_Voter1]: q2 -> q3
idle: q3 -> q3

FORMULA: <<Coercer>>[](C_V1_finish==0 ||
          (V1_vote==1 && &K_Coercer(V1_vote==1)) )
```

Agent

Initial configuration

Shared transition

Local name

Local transition

Guard

State (template)

Proposition variable

Formula

Tool

STV - Strategic Verifier

- Explicit-state model checking.

STV - Strategic Verifier

- Explicit-state model checking.
- User-defined input.

STV - Strategic Verifier

- Explicit-state model checking.
- User-defined input.
- Web-based graphical interface.

STV - Strategic Verifier

- Explicit-state model checking.
- User-defined input.
- Web-based graphical interface.
- Model-checking algorithms: fixpoint-approximations, depth-first strategy synthesis and on-the-fly strategy synthesis.

STV - Strategic Verifier

- Explicit-state model checking.
- User-defined input.
- Web-based graphical interface.
- Model-checking algorithms: fixpoint-approximations, depth-first strategy synthesis and on-the-fly strategy synthesis.
- Reduction methods: partial-order reductions and assume-guarantee reasoning.

STV - Strategic Verifier

- Explicit-state model checking.
- User-defined input.
- Web-based graphical interface.
- Model-checking algorithms: fixpoint-approximations, depth-first strategy synthesis and on-the-fly strategy synthesis.
- Reduction methods: partial-order reductions and assume-guarantee reasoning.
- Asynchronous semantics with: action-oriented synchronization and data-oriented synchronization.

STV - Strategic Verifier

- Explicit-state model checking.
- User-defined input.
- Web-based graphical interface.
- Model-checking algorithms: fixpoint-approximations, depth-first strategy synthesis and on-the-fly strategy synthesis.
- Reduction methods: partial-order reductions and assume-guarantee reasoning.
- Asynchronous semantics with: action-oriented synchronization and data-oriented synchronization.
- Properties: reachability and safety.

STV - Strategic Verifier

- Explicit-state model checking.
- User-defined input.
- Web-based graphical interface.
- Model-checking algorithms: fixpoint-approximations, depth-first strategy synthesis and on-the-fly strategy synthesis.
- Reduction methods: partial-order reductions and assume-guarantee reasoning.
- Asynchronous semantics with: action-oriented synchronization and data-oriented synchronization.
- Properties: reachability and safety.
- Epistemic operators: knowledge and Hartley uncertainty.

Approximate Verification of Strategic Ability

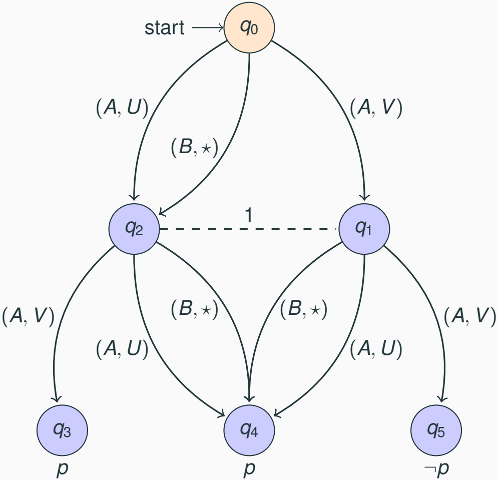
$$M \models_{ir} \varphi : \textbf{DIFFICULT!}$$

$$M \models LB(\varphi) \Rightarrow M \models_{ir} \varphi \Rightarrow M \models UB(\varphi)$$

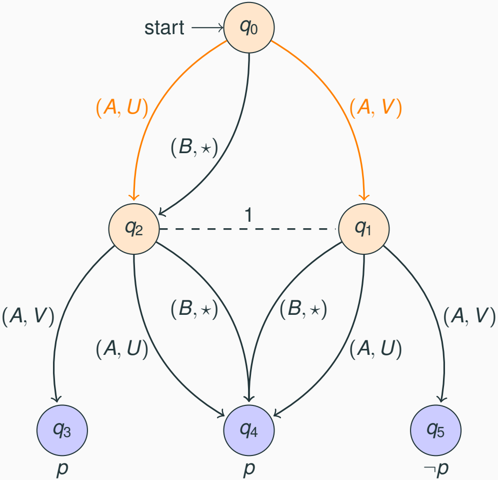
Alternating Epistemic
Mu-Calculus

Perfect Information

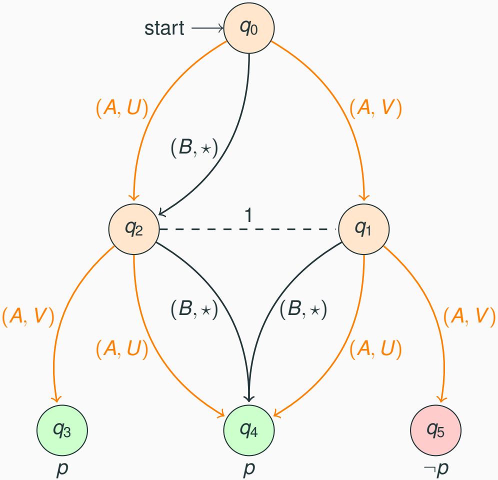
DFS Strategy Synthesis



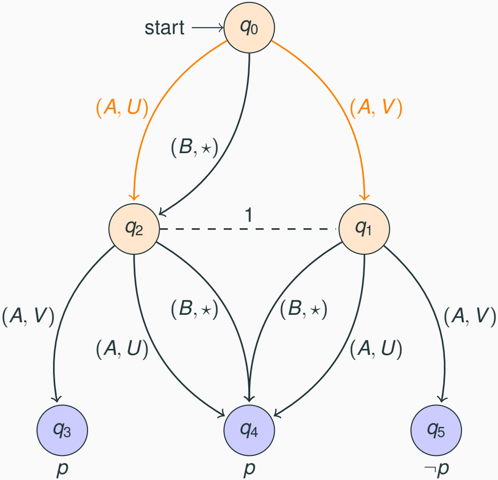
DFS Strategy Synthesis



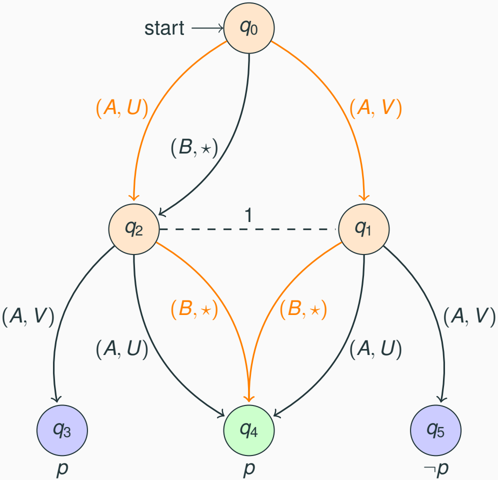
DFS Strategy Synthesis



DFS Strategy Synthesis



DFS Strategy Synthesis



Partial-order Reductions

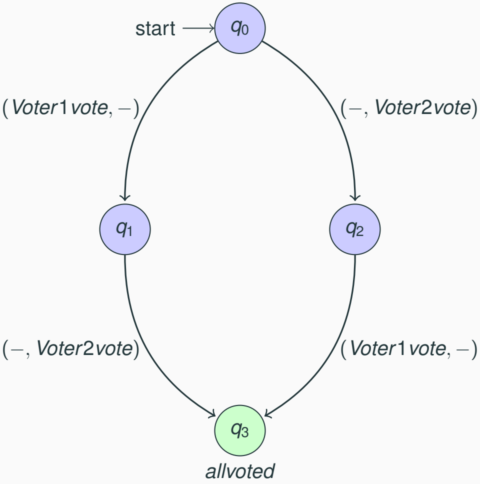
POR

POR is a method of generating reduced state spaces, preserving some temporal formula ϕ, that exploits:

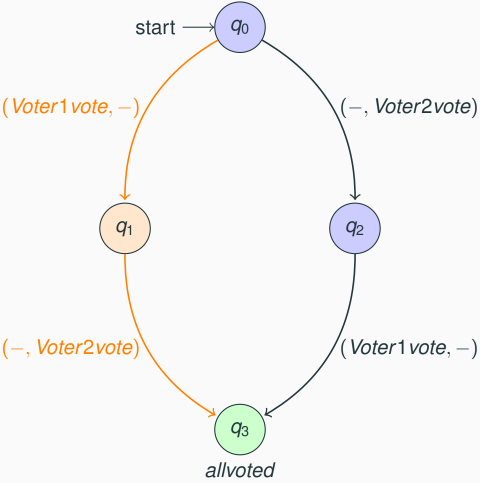
- Independency of actions, restricted to the pairs of actions such that one of them is invisible, i.e., does not change valuations of the atomic propositions used in ϕ,
- Infinite sequences of global locations that differ in the ordering of independent actions only are called ϕ-equivalent,
- ϕ does not distinguish between ϕ-equivalent sequences,

A reduced state space contains for each infinite sequence at least one ϕ-equivalent, but as few as possible.

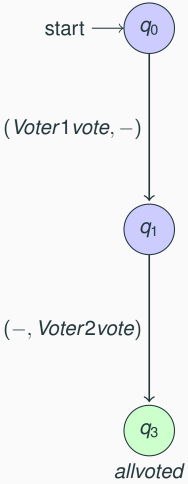
POR example


start
q_0
$(Voter1vote, -)$
$(-, Voter2vote)$
q_1
q_2
$(-, Voter2vote)$
$(Voter1vote, -)$
q_3
$allvoted$

POR example


start
q_0
$(Voter1vote, -)$
$(-, Voter2vote)$
q_1
q_2
$(-, Voter2vote)$
$(Voter1vote, -)$
q_3
$allvoted$

POR example



Assume-guarantee Verification

An assumption $A = (M, F)$ is a module augmented with a finite set of accepting states $F \subseteq Q$.

Composition of modules M guaranties an assumption A (which operates on subset of M's variables) if A accepts all traces derived by M (modulo stattering).

Assume-guarantee Verification

An assumption $A = (M, F)$ is a module augmented with a finite set of accepting states $F \subseteq Q$.

Composition of modules M guaranties an assumption A (which operates on subset of M's variables) if A accepts all traces derived by M (modulo stattering).

Automated assumptions

1. Design the model and create a specification file.
2. Split the agents into assumption groups.
3. Each assumption group should specify the coalition and the formula. Environment group should not specify the formula.
4. Use STV to automatically generate specification files for each assumption group.
5. Verify each model in the tool.

Conclusions

Conclusions

- Modal logics for MAS are characterized by high computational complexity.
- Verification of strategic properties in scenarios with imperfect information is **difficult**.
- Much complexity of model checking for strategic abilities is due to the size of the model of the system.
- STV addresses the challenge by implementing various **reduction** and **model-checking** methods which shows very promising performance.
- STV supports user-friendly modelling of MAS, and automated reduction and verification methods.

THANK YOU!