Probabilistic Hyperproperties with Nondeterminism

Erika Ábrahám, RWTH Aachen, Germany
Borzoo Bonakdarpour and Oyendrila Dobe, Michigan State University
Ezio Bartocci, TU Wien, Austria

ATVA 2020
22 October 2020

Trace properties and hyperproperties for non-deterministic systems [Clarkson, Schneider, 2010]

[Clarkson, Finkbeiner, Koleini, Micinski, Rabe, Sánchez, 2014]

- A trace $t = s_0, s_1, \ldots$ is an infinite sequence of states $s_i \in S$.
- A trace property is a set of traces.

Example (LTL property)

I never start working before having a coffee.

$$(\mathcal{G}\ \neg work) \vee ((\neg work)\ \mathcal{U}\ coffee)$$

Trace properties and hyperproperties for non-deterministic systems [Clarkson, Schneider, 2010]

[Clarkson, Finkbeiner, Koleini, Micinski, Rabe, Sánchez, 2014]

- A trace $t = s_0, s_1, \ldots$ is an infinite sequence of states $s_i \in S$.
- A trace property is a set of traces.

Example (LTL property)

I never start working before having a coffee.

$$(\mathcal{G}\ \neg work) \vee ((\neg work)\ \mathcal{U}\ coffee)$$

Classical trace properties cannot express relations between traces.

Trace properties and hyperproperties for non-deterministic systems [Clarkson, Schneider, 2010]

[Clarkson, Finkbeiner, Koleini, Micinski, Rabe, Sánchez, 2014]

- A trace $t = s_0, s_1, \ldots$ is an infinite sequence of states $s_i \in S$.
- A trace property is a set of traces.

Example (LTL property)

I never start working before having a coffee.

$$(\mathcal{G}\ \neg work) \vee ((\neg work)\ \mathcal{U}\ coffee)$$

Classical trace properties cannot express relations between traces.

- A hyperproperty is a set of sets of traces.

Trace properties and hyperproperties for non-deterministic systems [Clarkson, Schneider, 2010]

[Clarkson, Finkbeiner, Koleini, Micinski, Rabe, Sánchez, 2014]

- A trace $t = s_0, s_1, \ldots$ is an infinite sequence of states $s_i \in S$.
- A trace property is a set of traces.

Example (LTL property)

I never start working before having a coffee.

$$(\mathcal{G}\ \neg work) \vee ((\neg work)\ \mathcal{U}\ coffee)$$

Classical trace properties cannot express relations between traces.

- A hyperproperty is a set of sets of traces.

Example (HyperLTL property)

I drink coffee every day at the same time.

$$\forall \pi.\ \forall \pi'.\quad (\mathcal{G}\ (coffee_{\pi} \Leftrightarrow coffee_{\pi'}))$$

Probabilistic hyperproperties

Consider a parallel program (h: high input / l: low output).

$$\boxed{\text{while } h > 0 \text{ do } \{h \leftarrow h - 1\};\ l \leftarrow 2 \quad || \quad l \leftarrow 1}$$

Probabilistic hyperproperties

Consider a parallel program (h: high input / l: low output).

$$\boxed{\text{while } h > 0 \text{ do } \{h \leftarrow h - 1\};\ l \leftarrow 2 \quad || \quad l \leftarrow 1}$$

Assuming a uniform probabilistic scheduler:

Probabilistic hyperproperties

Consider a parallel program (h: high input / l: low output).

$$\boxed{\text{while } h > 0 \text{ do } \{h \leftarrow h - 1\};\ l \leftarrow 2 \quad || \quad l \leftarrow 1}$$

Assuming a uniform probabilistic scheduler:

- $h = 0 \rightarrow \mathbb{P}(l{=}1) = 1/4$ at termination.
- $h = 5 \rightarrow \mathbb{P}(l{=}1) = 1/4096$ at termination.

Probabilistic hyperproperties

Consider a parallel program (h: high input / l: low output).

$$\boxed{\text{while } h > 0 \text{ do } \{h \leftarrow h - 1\};\ l \leftarrow 2 \quad || \quad l \leftarrow 1}$$

Assuming a uniform probabilistic scheduler:

- $h = 0 \rightarrow \mathbb{P}(l{=}1) = 1/4$ at termination.
- $h = 5 \rightarrow \mathbb{P}(l{=}1) = 1/4096$ at termination.

Probabilistic noninterference stipulates that the probability distribution on the final values on publicly observable channels (low outputs) is independent of the initial values of secrets (high inputs).

Probabilistic hyperproperties

Consider a parallel program (h: high input / l: low output).

$$\boxed{\text{while } h > 0 \text{ do } \{h \leftarrow h - 1\};\ l \leftarrow 2 \quad || \quad l \leftarrow 1}$$

Assuming a uniform probabilistic scheduler:

- $h = 0 \rightarrow \mathbb{P}(l{=}1) = 1/4$ at termination.
- $h = 5 \rightarrow \mathbb{P}(l{=}1) = 1/4096$ at termination.

Probabilistic noninterference stipulates that the probability distribution on the final values on publicly observable channels (low outputs) is independent of the initial values of secrets (high inputs).

We need probabilistic hyperproperties to express probabilistic relations between independent executions of a system.

HyperPCTL for DTMCs [Ábrahám, Bonakdarpour, 2018]

HyperPCTL: PCTL extended with quantification over initial states

Example (Probabilistic noninterference)

$$\forall \hat{s}.\forall \hat{s}'.\Big(init_{\hat{s}} \land init_{\hat{s}'} \land h_{\hat{s}} \neq h_{\hat{s}'}\Big) \Rightarrow \Big(\mathbb{P}\big(\mathcal{F}(fin_{\hat{s}} \land (l{=}1)_{\hat{s}})\big) = \mathbb{P}\big(\mathcal{F}(fin_{\hat{s}'} \land (l{=}1)_{\hat{s}'})\big)\Big)$$

HyperPCTL for DTMCs [Ábrahám, Bonakdarpour, 2018]

HyperPCTL: PCTL extended with quantification over initial states

Example (Probabilistic noninterference)

$$\forall \hat{s}.\forall \hat{s}'.\Big(\mathit{init}_{\hat{s}} \wedge \mathit{init}_{\hat{s}'} \wedge h_{\hat{s}} \neq h_{\hat{s}'} \Big) \Rightarrow \Big(\mathbb{P}\Big(\mathcal{F}(\mathit{fin}_{\hat{s}} \wedge (l{=}1)_{\hat{s}}) \Big) = \mathbb{P}\Big(\mathcal{F}(\mathit{fin}_{\hat{s}'} \wedge (l{=}1)_{\hat{s}'}) \Big) \Big)$$

But this argues about a fixed scheduler only...

HyperPCTL for DTMCs [Ábrahám, Bonakdarpour, 2018]

HyperPCTL: PCTL extended with quantification over initial states

Example (Probabilistic noninterference)

$$\forall \hat{s}.\forall \hat{s}'.\Big(init_{\hat{s}} \wedge init_{\hat{s}'} \wedge h_{\hat{s}} \neq h_{\hat{s}'}\Big) \Rightarrow$$
$$\Big(\mathbb{P}\Big(\mathcal{F}(fin_{\hat{s}} \wedge (l{=}1)_{\hat{s}})\Big) = \mathbb{P}\Big(\mathcal{F}(fin_{\hat{s}'} \wedge (l{=}1)_{\hat{s}'})\Big)\Big)$$

But this argues about a fixed scheduler only...

Our contribution:
extend HyperPCTL to non-deterministic probabilistic systems!

HyperPCTL syntax

HyperPCTL syntax

HyperPCTL formulas are similar to PCTL formulas BUT

- they quantify ($Q_i \in \{\exists, \forall\}$) over schedulers and initial states:

$$Q_{\hat\sigma_1}\hat\sigma_1 \dots . Q_{\hat\sigma_m}\hat\sigma_m . \quad Q_{\hat s_1}\hat s_1(\hat\sigma_{i_1}) \dots Q_{\hat s_n}\hat s_n(\hat\sigma_{i_n}). \quad \psi$$

- they index atomic propositions:

$$\psi ::= a_{\hat s} \;\Big|\; \psi \wedge \psi \;\Big|\; \neg\psi \;\Big|\; p < c$$

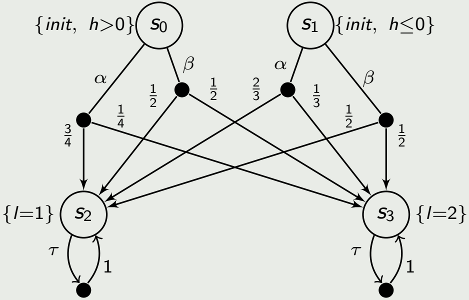
- they support arithmetic computations with probability expressions:

$$p \quad ::= \quad \mathbb{P}(\mathcal{X}\psi) \;\Big|\; \mathbb{P}(\psi\,\mathcal{U}\,\psi) \;\Big|\; \mathbb{P}(\psi\,\mathcal{U}^{[k_1,k_2]}\,\psi) \;\Big|$$

$$c \;\Big|\; p + p \;\Big|\; p - p \;\Big|\; p \cdot p$$

HyperPCTL syntax

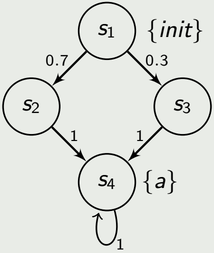
Example



$$\psi = \exists\hat{\sigma}.\ \forall\hat{s}.\ \forall\hat{s}'.\ (init_{\hat{s}} \wedge init_{\hat{s}'}) \ \Rightarrow\ \Big(\mathbb{P}(\mathcal{F}(l{=}1)_{\hat{s}}) = \mathbb{P}(\mathcal{F}(l{=}1)_{\hat{s}'})\Big)$$

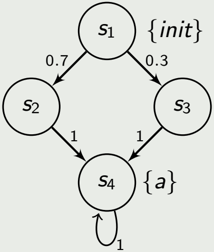
DTMCs and their self-composition

Example



DTMCs and their self-composition

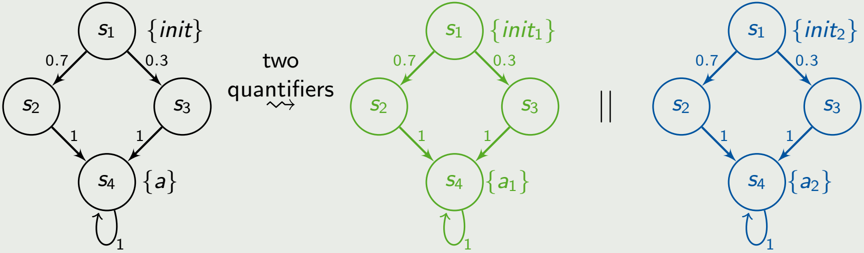
Example



two quantifiers $\rightsquigarrow$

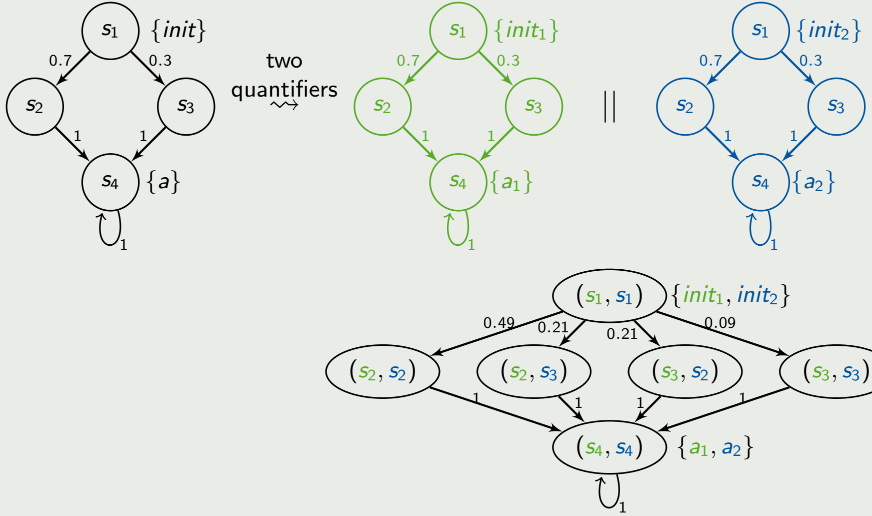
DTMCs and their self-composition

Example



DTMCs and their self-composition

Example



HyperPCTL semantics

$$\mathcal{M} \models \varphi \qquad \textit{iff} \qquad \underbrace{\mathcal{M}}_{MDP}, \underbrace{()}_{\textit{schedulers of initial states}}, \underbrace{()}_{\textit{initial states}} \models \varphi$$

$$\mathcal{M}, \vec{\sigma}, \vec{r} \models \forall \hat{\sigma}.\varphi \qquad \textit{iff} \quad \forall \sigma \in \Sigma^{\mathcal{M}}.\ \mathcal{M}, \vec{\sigma}, \vec{r} \models \varphi[\hat{\sigma} \rightsquigarrow \sigma]$$

$\dots$

$$\mathcal{M}, \vec{\sigma}, \vec{r} \models \forall \hat{s}(\sigma).\varphi \quad \textit{iff} \quad \forall s_{n+1} \in S.\ \mathcal{M}, \vec{\sigma} \circ \sigma, \vec{r} \circ (init(s_{n+1}), s_{n+1}) \models \varphi[\hat{s} \rightsquigarrow s_{n+1}]$$

$\dots$

$$[\![\mathbb{P}(\varphi_{path})]\!]_{\mathcal{M},\vec{\sigma},\vec{r}} \quad = \quad Pr^{\mathcal{M}^{\vec{\sigma}}}\left(\{\pi \in Paths^{\vec{r}}(\mathcal{M}^{\vec{\sigma}}) \mid \mathcal{M}, \vec{\sigma}, \pi \models \varphi_{path}\}\right)$$

$\dots$

$$\mathcal{M}, \vec{\sigma}, \pi \models \varphi_1 \mathcal{U} \varphi_2 \quad \textit{iff} \quad \exists j \geq 0. \Big(\mathcal{M}, \vec{\sigma}, \vec{r}_j \models \varphi_2 \wedge \forall i \in [0, j).\ \mathcal{M}, \vec{\sigma}, \vec{r}_i \models \varphi_1\Big)$$

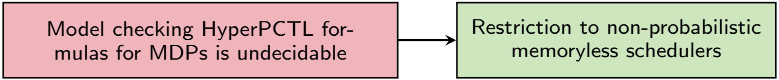
HyperPCTL semantics vs PCTL semantics

- For DTMCs, the two semantics are equivalent.
- For MDPs,
  - PCTL evaluates a probability constraint $\mathbb{P}(\varphi) < c$ to true if it holds under all schedulers,
  - HyperPCTL supports free quantifier type and positioning.

Theorem

The HyperPCTL model checking problem for MDPs is undecidable.

Restricted HyperPCTL Model Checking



Model checking HyperPCTL formulas for MDPs is undecidable

Restriction to non-probabilistic memoryless schedulers

Restricted HyperPCTL Model Checking

Model checking HyperPCTL formulas for MDPs is undecidable → Restriction to non-probabilistic memoryless schedulers

We propose an SMT-based technique for solving the model checking problem, such that

$$\boxed{\begin{array}{c}\mathcal{M} = (S, Act, \mathrm{P}, AP, L) \text{ satisfies} \\ Q\hat{\sigma}.Q_1\hat{s}_1(\hat{\sigma}).\ldots Q_n\hat{s}_n(\hat{\sigma}).\varphi^{nq}\end{array}}$$

iff

$$\boxed{\text{SMT encoding is satisfied}}$$

We explain the simplified case of having a single scheduler quantifier for understanding of the basic ideas.

SMT Encoding Algorithm for Existential scheduler quantifier

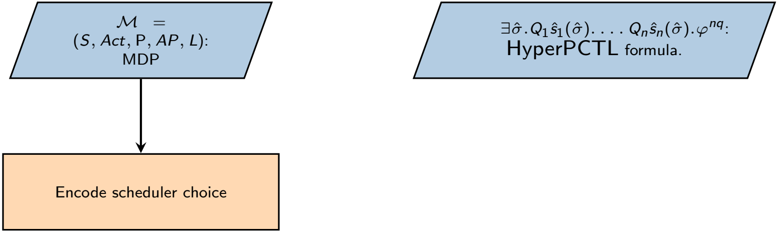
$\mathcal{M} = (S, Act, \mathrm{P}, AP, L)$: MDP

SMT Encoding Algorithm for Existential scheduler quantifier

$\mathcal{M} = (S, Act, \mathrm{P}, AP, L)$: MDP

$\exists\hat{\sigma}.Q_1\hat{s}_1(\hat{\sigma}).\dots.Q_n\hat{s}_n(\hat{\sigma}).\varphi^{nq}$: HyperPCTL formula.

SMT Encoding Algorithm for Existential scheduler quantifier

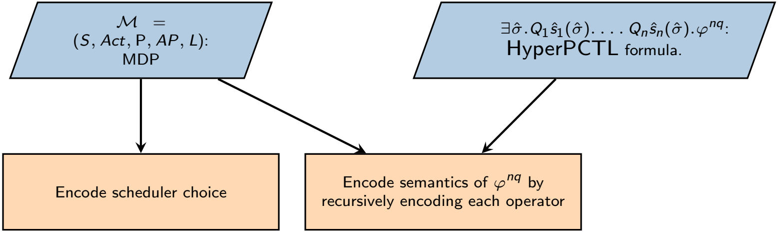


$\mathcal{M} = (S, Act, \mathrm{P}, AP, L)$: MDP

Encode scheduler choice

$\exists\hat{\sigma}.Q_1\hat{s}_1(\hat{\sigma}).\ldots Q_n\hat{s}_n(\hat{\sigma}).\varphi^{nq}$: HyperPCTL formula.

SMT Encoding Algorithm for Existential scheduler quantifier



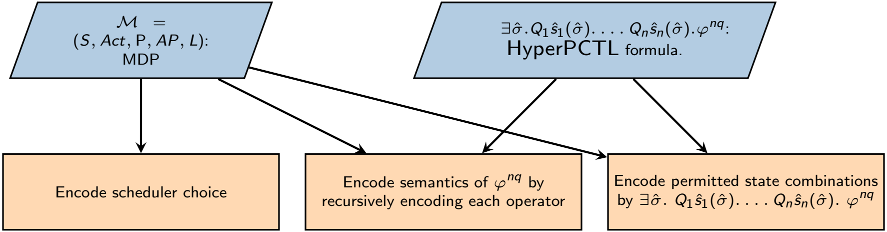
$\mathcal{M} = (S, Act, \mathrm{P}, AP, L)$: MDP

$\exists\hat{\sigma}.Q_1\hat{s}_1(\hat{\sigma}).\ldots Q_n\hat{s}_n(\hat{\sigma}).\varphi^{nq}$: HyperPCTL formula.

Encode scheduler choice

Encode semantics of φ^{nq} by recursively encoding each operator

SMT Encoding Algorithm for Existential scheduler quantifier



$\mathcal{M} = (S, Act, \mathrm{P}, AP, L)$: MDP

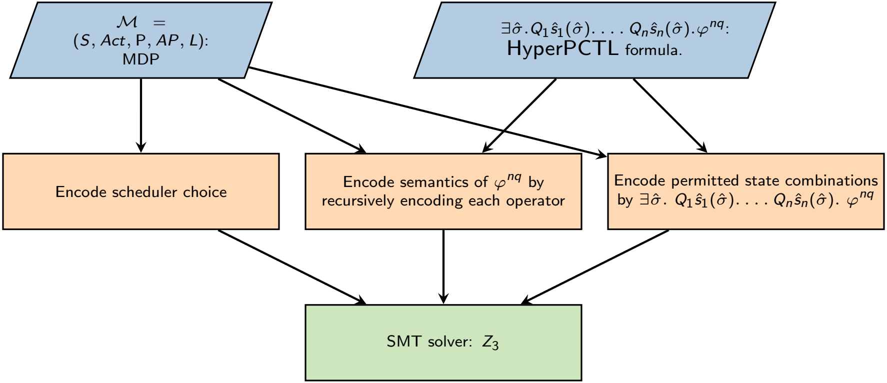
$\exists\hat{\sigma}.Q_1\hat{s}_1(\hat{\sigma}).\ldots Q_n\hat{s}_n(\hat{\sigma}).\varphi^{nq}$: HyperPCTL formula.

Encode scheduler choice

Encode semantics of φ^{nq} by recursively encoding each operator

Encode permitted state combinations by $\exists\hat{\sigma}.\ Q_1\hat{s}_1(\hat{\sigma}).\ldots Q_n\hat{s}_n(\hat{\sigma}).\ \varphi^{nq}$

SMT Encoding Algorithm for Existential scheduler quantifier



$\mathcal{M} = (S, Act, \mathrm{P}, AP, L)$: MDP

$\exists\hat{\sigma}.Q_1\hat{s}_1(\hat{\sigma}).\dots Q_n\hat{s}_n(\hat{\sigma}).\varphi^{nq}$: HyperPCTL formula.

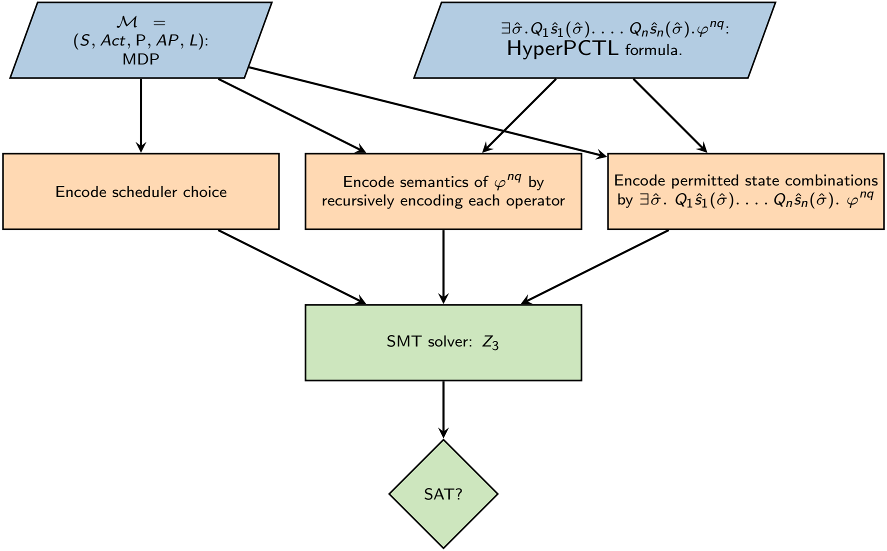
Encode scheduler choice

Encode semantics of φ^{nq} by recursively encoding each operator

Encode permitted state combinations by $\exists\hat{\sigma}.\ Q_1\hat{s}_1(\hat{\sigma}).\dots Q_n\hat{s}_n(\hat{\sigma}).\ \varphi^{nq}$

SMT solver: Z_3

SMT Encoding Algorithm for Existential scheduler quantifier



$\mathcal{M} = (S, Act, \mathrm{P}, AP, L)$: MDP

$\exists\hat{\sigma}.Q_1\hat{s}_1(\hat{\sigma}).\dots Q_n\hat{s}_n(\hat{\sigma}).\varphi^{nq}$: HyperPCTL formula.

Encode scheduler choice

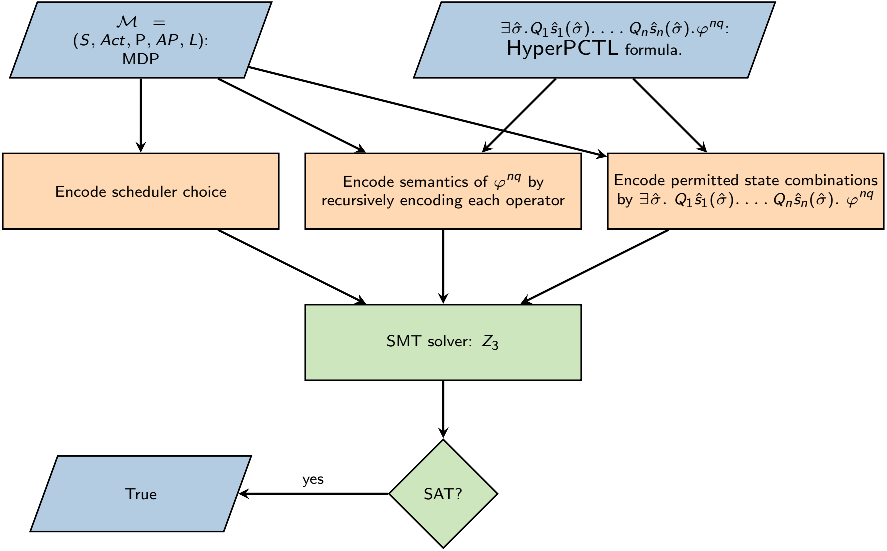
Encode semantics of φ^{nq} by recursively encoding each operator

Encode permitted state combinations by $\exists\hat{\sigma}.\ Q_1\hat{s}_1(\hat{\sigma}).\dots Q_n\hat{s}_n(\hat{\sigma}).\ \varphi^{nq}$

SMT solver: Z_3

SAT?

SMT Encoding Algorithm for Existential scheduler quantifier



$\mathcal{M} = (S, Act, \mathrm{P}, AP, L)$: MDP

$\exists\hat{\sigma}.Q_1\hat{s}_1(\hat{\sigma}).\dots Q_n\hat{s}_n(\hat{\sigma}).\varphi^{nq}$: HyperPCTL formula.

Encode scheduler choice

Encode semantics of φ^{nq} by recursively encoding each operator

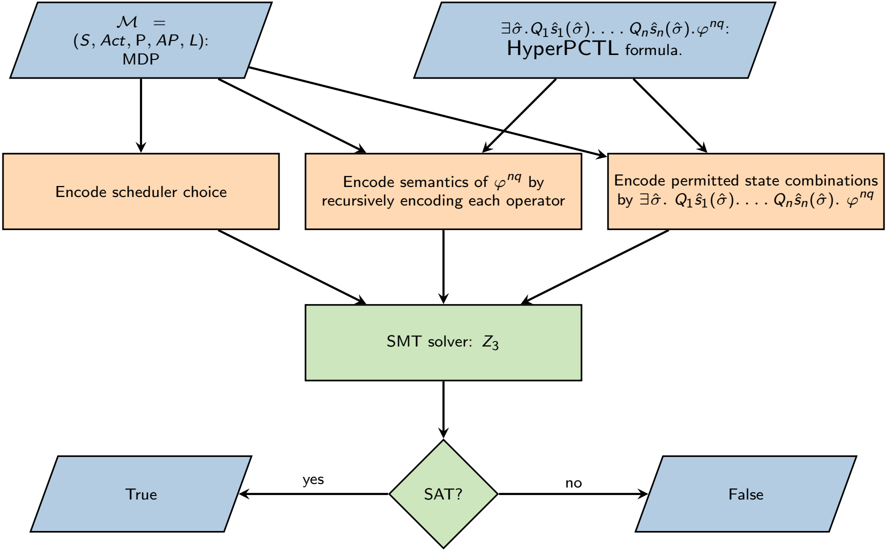
Encode permitted state combinations by $\exists\hat{\sigma}.\ Q_1\hat{s}_1(\hat{\sigma}).\dots Q_n\hat{s}_n(\hat{\sigma}).\ \varphi^{nq}$

SMT solver: Z_3

SAT?

True

SMT Encoding Algorithm for Existential scheduler quantifier



$\mathcal{M} = (S, Act, \mathrm{P}, AP, L)$: MDP

$\exists\hat{\sigma}.Q_1\hat{s}_1(\hat{\sigma}).\dots Q_n\hat{s}_n(\hat{\sigma}).\varphi^{nq}$: HyperPCTL formula.

Encode scheduler choice

Encode semantics of φ^{nq} by recursively encoding each operator

Encode permitted state combinations by $\exists\hat{\sigma}.\ Q_1\hat{s}_1(\hat{\sigma}).\dots Q_n\hat{s}_n(\hat{\sigma}).\ \varphi^{nq}$

SMT solver: Z_3

SAT?

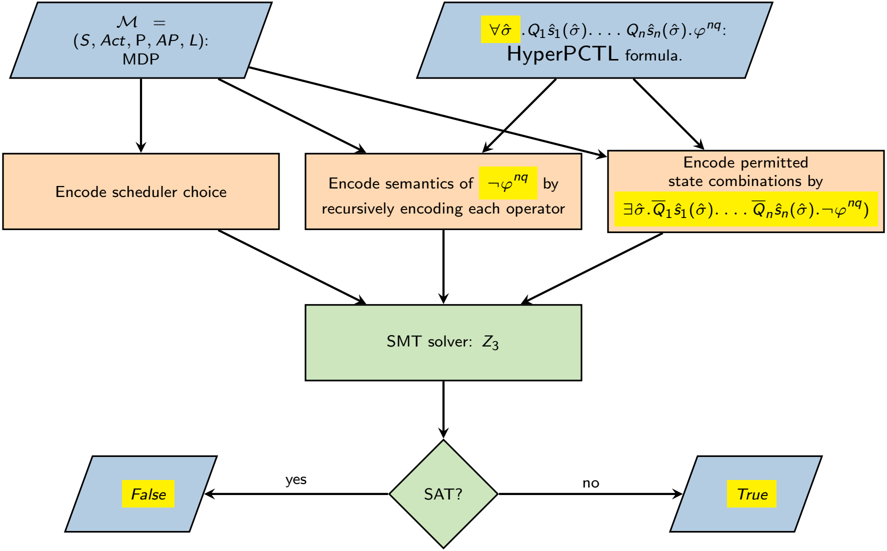
yes

no

True

False

SMT Encoding Algorithm for Universal scheduler quantifier



$\mathcal{M} = (S, Act, \mathrm{P}, AP, L)$: MDP

$\forall\hat{\sigma}.Q_1\hat{s}_1(\hat{\sigma}).\ldots Q_n\hat{s}_n(\hat{\sigma}).\varphi^{nq}$: HyperPCTL formula.

Encode scheduler choice

Encode semantics of $\neg\varphi^{nq}$ by recursively encoding each operator

Encode permitted state combinations by $\exists\hat{\sigma}.\overline{Q}_1\hat{s}_1(\hat{\sigma}).\ldots\overline{Q}_n\hat{s}_n(\hat{\sigma}).\neg\varphi^{nq})$

SMT solver: Z_3

SAT?

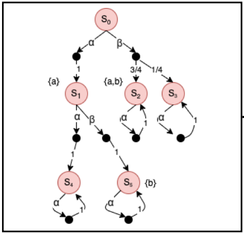
yes

False

no

True

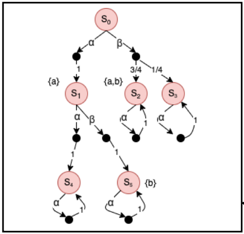
Elaborating on each sub-method used: scheduler choice



Encode scheduler choice

$$(\sigma_0 = \alpha \vee \sigma_0 = \beta) \wedge (\sigma_1 = \alpha \vee \sigma_1 = \beta) \\ \wedge (\sigma_2 = \alpha) \wedge (\sigma_3 = \alpha) \wedge \\ (\sigma_4 = \alpha) \wedge (\sigma_5 = \alpha)$$

Elaborating on each sub-method used: State Quantifier encoding

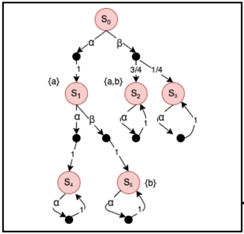


$$\exists \hat{\sigma}.\forall \hat{s}(\hat{\sigma}).\exists \hat{s}'(\hat{\sigma}).\Big(init_{\hat{s}} \;\wedge\; init_{\hat{s}'}\Big) \;\Rightarrow\; \Big(\mathbb{P}(\mathcal{F} a_{\hat{s}}) = \mathbb{P}(\mathcal{F} a_{\hat{s}'})\Big)$$

Encode permitted state combinations

$$\begin{gathered}
(holds_{s0,s0,\varphi^{nq}} \vee holds_{s0,s1,\varphi^{nq}} \vee holds_{s0,s2,\varphi^{nq}} \vee holds_{s0,s3,\varphi^{nq}} \vee holds_{s0,s4,\varphi^{nq}} \vee holds_{s0,s5,\varphi^{nq}})\wedge \\
(holds_{s1,s0,\varphi^{nq}} \vee holds_{s1,s1,\varphi^{nq}} \vee holds_{s1,s2,\varphi^{nq}} \vee holds_{s1,s3,\varphi^{nq}} \vee holds_{s1,s4,\varphi^{nq}} \vee holds_{s1,s5,\varphi^{nq}})\wedge \\
(holds_{s2,s0,\varphi^{nq}} \vee \ldots holds_{s2,s5,\varphi^{nq}}) \wedge (holds_{s3,s0,\varphi^{nq}} \vee \cdots \vee holds_{s3,s5,\varphi^{nq}}) \wedge \\
(holds_{s4,s0,\varphi^{nq}} \vee \cdots \vee holds_{s4,s5,\varphi^{nq}}) \wedge (holds_{s5,s0,\varphi^{nq}} \vee \cdots \vee holds_{s5,s5,\varphi^{nq}})
\end{gathered}$$

Elaborating on each sub-method used: Semantics encoding

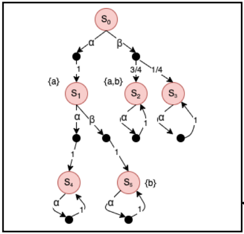


$$\left(init_{\hat{s}} \quad \wedge \quad init_{\hat{s}'}\right) \quad \Rightarrow \quad \left(\mathbb{P}(\mathcal{F} a_{\hat{s}}) \;=\; \mathbb{P}(\mathcal{F} a_{\hat{s}'})\right)$$

Encode sematics

$$Encoding(init_{\hat{s}}) = (holds_{s0,s0,init_{\hat{s}}} \wedge \cdots \wedge holds_{s0,s5,init_{\hat{s}}}) \wedge (\neg holds_{s1,s0,init_{\hat{s}}} \wedge \neg holds_{s1,s1,init_{\hat{s}}} \wedge \cdots \wedge \neg holds_{s5,s5,init_{\hat{s}}})$$

Elaborating on each sub-method used: Semantics encoding



$$\left(init_{\hat{s}} \quad \wedge \quad init_{\hat{s}'}\right) \quad \Rightarrow \quad \left(\mathbb{P}(\mathcal{F} a_{\hat{s}}) \;=\; \mathbb{P}(\mathcal{F} a_{\hat{s}'})\right)$$

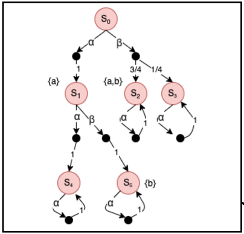
Encode sematics

$$Encoding(init_{\hat{s}}) = (holds_{s0,s0,init_{\hat{s}}} \wedge \cdots \wedge holds_{s0,s5,init_{\hat{s}}}) \wedge (\neg holds_{s1,s0,init_{\hat{s}}} \wedge \neg holds_{s1,s1,init_{\hat{s}}} \wedge \cdots \wedge \neg holds_{s5,s5,init_{\hat{s}}})$$

$$Encoding(init_{\hat{s}} \;\wedge\; init_{\hat{s}'}) \;\; for \;\; state(s0,s0) \;=\; (holds_{s0,s0,(init_{\hat{s}} \,\wedge\, init_{\hat{s}'})} \;\;\wedge\;\; holds_{s0,s0,init_{\hat{s}}} \wedge holds_{s0,s0,init_{\hat{s}'}}) \vee$$

$$(\neg holds_{s0,s0,(init_{\hat{s}} \,\wedge\, init_{\hat{s}'})} \;\;\wedge\;\; (\neg holds_{s0,s0,init_{\hat{s}}} \;\;\vee\;\; \neg holds_{s0,s0,init_{\hat{s}'}}))$$

Elaborating on each sub-method used: Semantics encoding



$$\left(init_{\hat{s}} \quad \wedge \quad init_{\hat{s}'}\right) \quad \Rightarrow \quad \left(\mathbb{P}(\mathcal{F} a_{\hat{s}}) \;=\; \mathbb{P}(\mathcal{F} a_{\hat{s}'})\right)$$

Encode sematics

$$Encoding(init_{\hat{s}}) = (holds_{s0,s0,init_{\hat{s}}} \wedge \cdots \wedge holds_{s0,s5,init_{\hat{s}}}) \wedge (\neg holds_{s1,s0,init_{\hat{s}}} \wedge \neg holds_{s1,s1,init_{\hat{s}}} \wedge \cdots \wedge \neg holds_{s5,s5,init_{\hat{s}}})$$

$$Encoding(init_{\hat{s}} \;\wedge\; init_{\hat{s}'}) \;\; for \;\; state(s0, s0) \;=\; (holds_{s0,s0,(init_{\hat{s}} \wedge init_{\hat{s}'})} \;\wedge\; holds_{s0,s0,init_{\hat{s}}} \wedge holds_{s0,s0,init_{\hat{s}'}}) \vee$$
$$(\neg holds_{s0,s0,(init_{\hat{s}} \wedge init_{\hat{s}'})} \;\wedge\; (\neg holds_{s0,s0,init_{\hat{s}}} \;\vee\; \neg holds_{s0,s0,init_{\hat{s}'}}))$$

$$Encoding(\mathbb{P}(\mathcal{F} a_{\hat{s}})) \;\; for \;\; state(s0, s0) \;\; action(\alpha_{\hat{s}}.\alpha_{\hat{s}'}) = (holds_{s0,s0,a_{\hat{s}}} \Rightarrow prob_{s0,s0,\mathbb{P}(\mathcal{F} a_{\hat{s}})} = 1) \wedge (prob_{s0,s0,\mathbb{P}(\mathcal{F} a_{\hat{s}})} \geq 0)$$
$$\wedge(\neg holds_{s0,s0,a_{\hat{s}}} \wedge (\sigma_0 = \alpha \wedge \sigma_0 = \alpha)) \Rightarrow (prob_{s0,s0,\mathbb{P}(\mathcal{F} a_{\hat{s}})} = (1 * 1 * prob_{s1,s1,\mathbb{P}(\mathcal{F} a_{\hat{s}})})$$
$$\wedge(prob_{s0,s0,\mathbb{P}(\mathcal{F} a_{\hat{s}})} > 0 \Rightarrow (holds_{s1,s1,a_{\hat{s}}} \vee d_{s0,s0,a_{\hat{s}}} > d_{s1,s1,a_{\hat{s}}})))$$

Side-channel timing leaks

- It allows an attacker to infer the value of a secret by observing execution time of a function.
- In this example, a is an integer representing the plaintext and b is the integer encryption key.
- This algorithm should satisfy the following property:

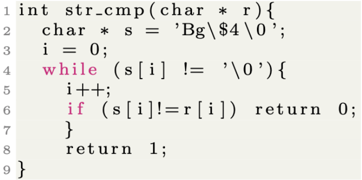
```
void mexp(){
  c = 0; d = 1; i = k;
  while (i >= 0){
    i = i-1; c = c*2;
    d = (d*d) % n;
    if (b(i) = 1)
      c = c+1;
      d = (d*a) % n;
    }
}
/************/
t = new Thread(mexp());
j = 0; m = 2 * k;
while (j < m & !t.stop) j++;
/************/
```

Figure: Modular Exponentiation

$$\forall \hat{\sigma}_1 . \forall \hat{\sigma}_2 . \forall \hat{s}(\hat{\sigma}_1) . \forall \hat{s}'(\hat{\sigma}_2) . \Big(init_{\hat{s}} \wedge init_{\hat{s}'} \Big) \Rightarrow \bigwedge_{l=0}^{m} \Big(\mathbb{P}(\mathcal{F}(j = l)_{\hat{s}}) = \mathbb{P}(\mathcal{F}(j = l)_{\hat{s}'}) \Big)$$

Probabilistic-scheduling Side-channel timing attack

- It allows an attacker to infer the value of a secret by observing execution time of a function.
- We want to ensure an attacker thread cannot be used to infer the number of correct bits we have in the user input.



```
int str_cmp(char * r){
  char * s = 'Bg\$4\0';
  i = 0;
  while (s[i] != '\0'){
    i++;
    if (s[i]!=r[i]) return 0;
    }
    return 1;
}
```

Figure: String comparison

Scheduler-specific observational determinism

Assume two threads:

$$th_1: \text{ while } h > 0 \text{ do } \{h := h - 1\};\ l := 2$$

$$th_2: \ l := 1$$

- Attacker should not be able to choose a specific scheduler to control set of traces generated.
- Observational determinism should be followed across all schedulers, according to the formula,

$$\forall \hat{\sigma}.\forall \hat{s}(\hat{\sigma}).\forall \hat{s}'(\hat{\sigma}).\big(h_{\hat{s}} \oplus h_{\hat{s}'}\big) \ \Rightarrow\ \mathbb{P}\mathcal{G}\Big(\bigwedge_{l \in L}\big((\mathbb{P}\mathcal{X}\, l_{\hat{s}}) = (\mathbb{P}\mathcal{X}\, l_{\hat{s}'})\big)\Big) = 1.$$

Probabilistic conformance

- Check if implementation conforms with given specification.
- Synthesize a protocol that simulates the 6-sided die behavior only by repeatedly tossing a fair coin.
- Given all the possible coin-implementations, our goal is to check if there exists a scheduler that gives us the DTMC from the given MDP using,

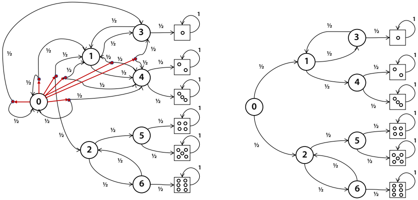


Figure: Knuth-Yao protocol for simulating fair dice using only fair coins

$$\exists\hat{\sigma}.\forall\hat{s}(\hat{\sigma}).\exists\hat{s}'(\hat{\sigma}).\Big(init_{\hat{s}} \wedge init_{\hat{s}'}\Big) \Rightarrow \bigwedge_{l=1}^{6}\Big(\mathbb{P}(\mathcal{F}(die = l)_{\hat{s}}) = \mathbb{P}(\mathcal{F}(die = l)_{\hat{s}'})\Big)$$

Evaluation

Case study		Running time (s)			#SMT variables	#subformulas	#states	#transitions
		SMT encoding	SMT solving	Total				
TA	$m = 2$	5.43	0.31	5.74	8088	50654	24	46
	$m = 4$	114	20	134	50460	368062	60	136
	$m = 6$	1721	865	2586	175728	1381118	112	274
PW	$m = 2$	5.14	0.3	8.14	8088	43432	24	46
	$m = 4$	207	40	247	68670	397852	70	146
	$m = 6$	3980	1099	5079	274540	1641200	140	302
TS	$h = (0, 1)$	0.83	0.07	0.9	1379	7913	7	13
	$h = (0, 15)$	60	1607	1667	34335	251737	35	83
	$h = (4, 8)$	11.86	17.02	28.88	12369	87097	21	48
	$h = (8, 15)$	60	1606	1666	34335	251737	35	83
PC	s=(0)	277	1996	2273	21220	1859004	20	158
	s=(0,1)	822	5808	6630	21220	5349205	20	280
	s=(0,1,2)	1690	58095	59785	21220	11006581	20	404

Table: Experimental results. **TA:** Timing attack. **PW:** Password leakage. **TS:** Thread scheduling. **PC:** Probabilistic conformance.

Here, for **TA**, m refers to 2*number of bits in the encryption key, for **PW**, m refers to 2*length of user password, for **PC**, we have included all possible transitions from the mentioned states.

Summary

Summary

Extended the temporal logic HyperPCTL for DTMCs to the context of MDPs by allowing formulas to quantify over schedulers.

Summary

Extended the temporal logic HyperPCTL for DTMCs to the context of MDPs by allowing formulas to quantify over schedulers.

This additional expressive power leads to undecidability of the HyperPCTL model checking on MDPs.

Summary

Extended the temporal logic HyperPCTL for DTMCs to the context of MDPs by allowing formulas to quantify over schedulers.

This additional expressive power leads to undecidability of the HyperPCTL model checking on MDPs.

Presented a SMT-based model checking algorithm which is NP-complete (coNP-complete for universal quantifier) in the state set size of the input MDP.

Future work

Future work

Optimize the current algorithm for faster results.

Future work

Optimize the current algorithm for faster results.

Design less accurate model checking algorithms that scale better alongside providing probabilistic guarantees about the correctness of verification.

Future work

Optimize the current algorithm for faster results.

Design less accurate model checking algorithms that scale better alongside providing probabilistic guarantees about the correctness of verification.

Counter-example guided techniques to manage large state space.