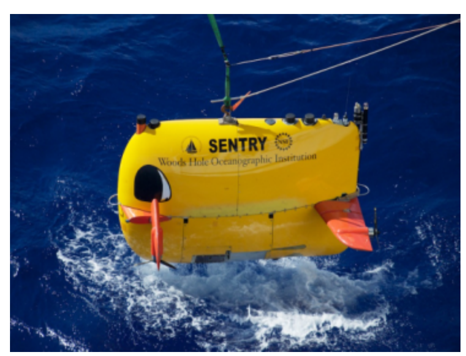
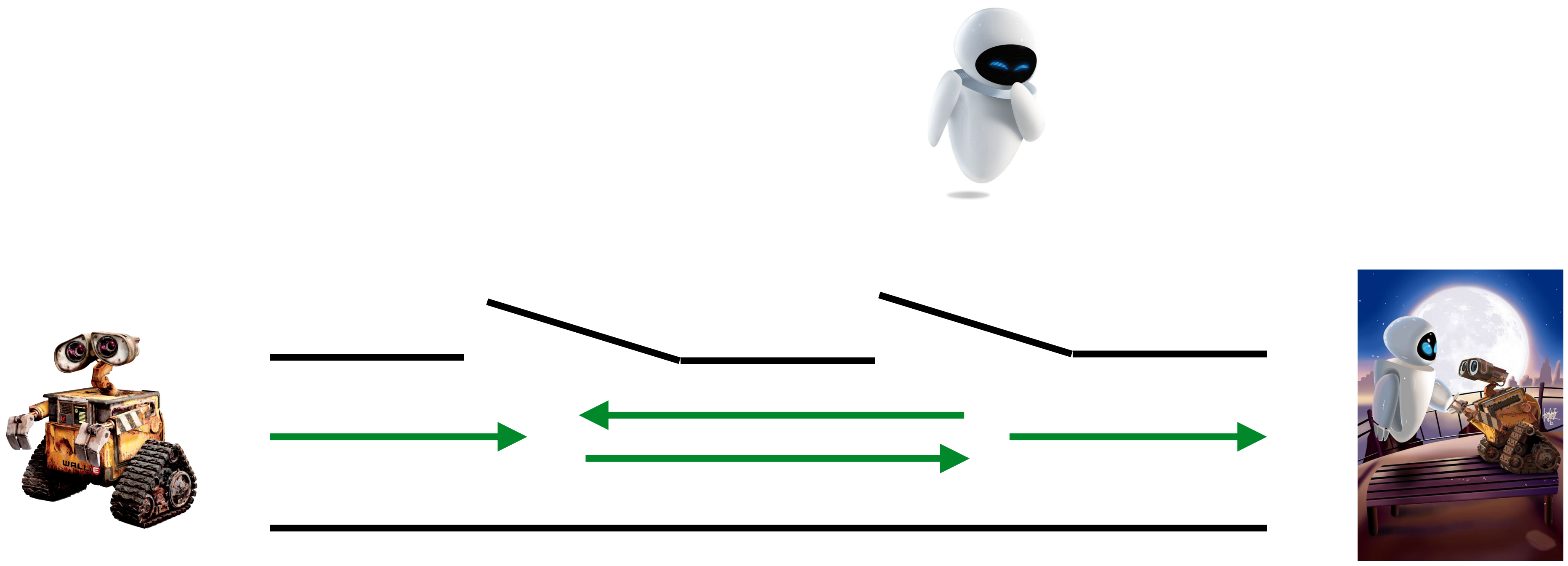
Heuristic Search Planning With Multi-Objective Probabilistic LTL Constraints

Peter Baumgartner, Sylvie Thiébaux, Felipe Trevizan

Data61/CSIRO and Research School of Computer Science, ANU
Australia



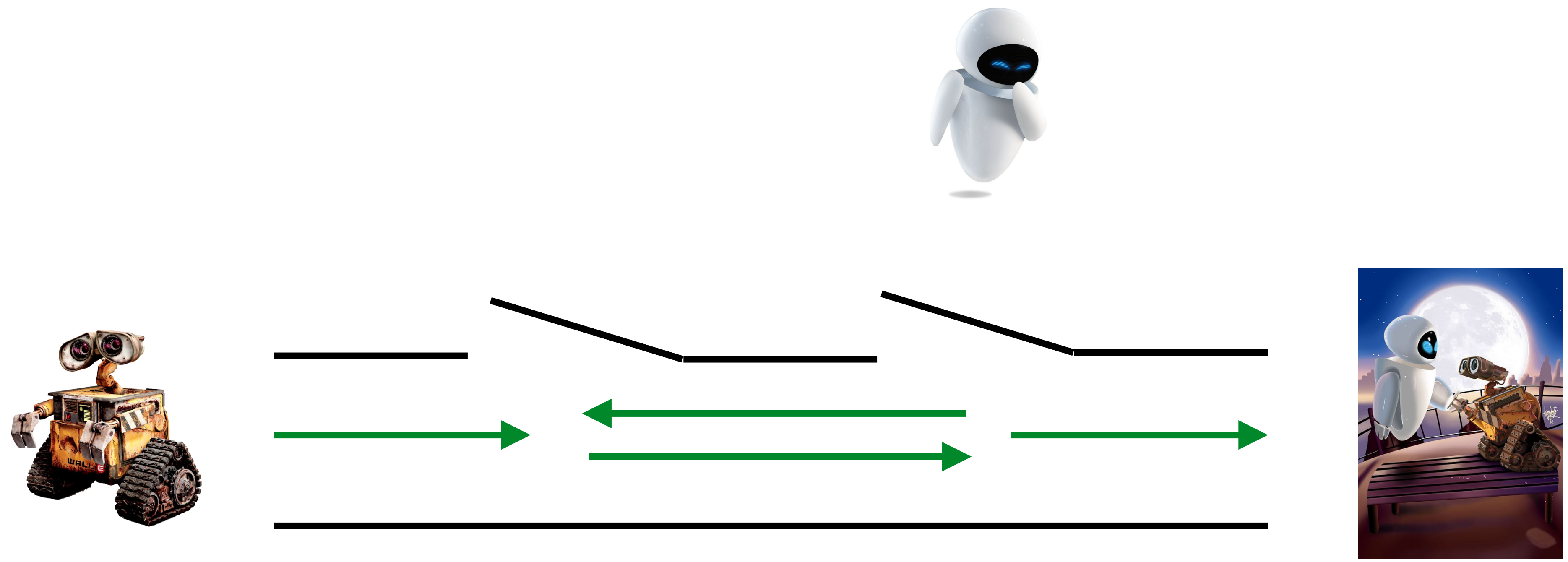
Planning Under Uncertainty



Goal

Actions: move left, move right, enter, get Eve, exit

Planning Under Uncertainty



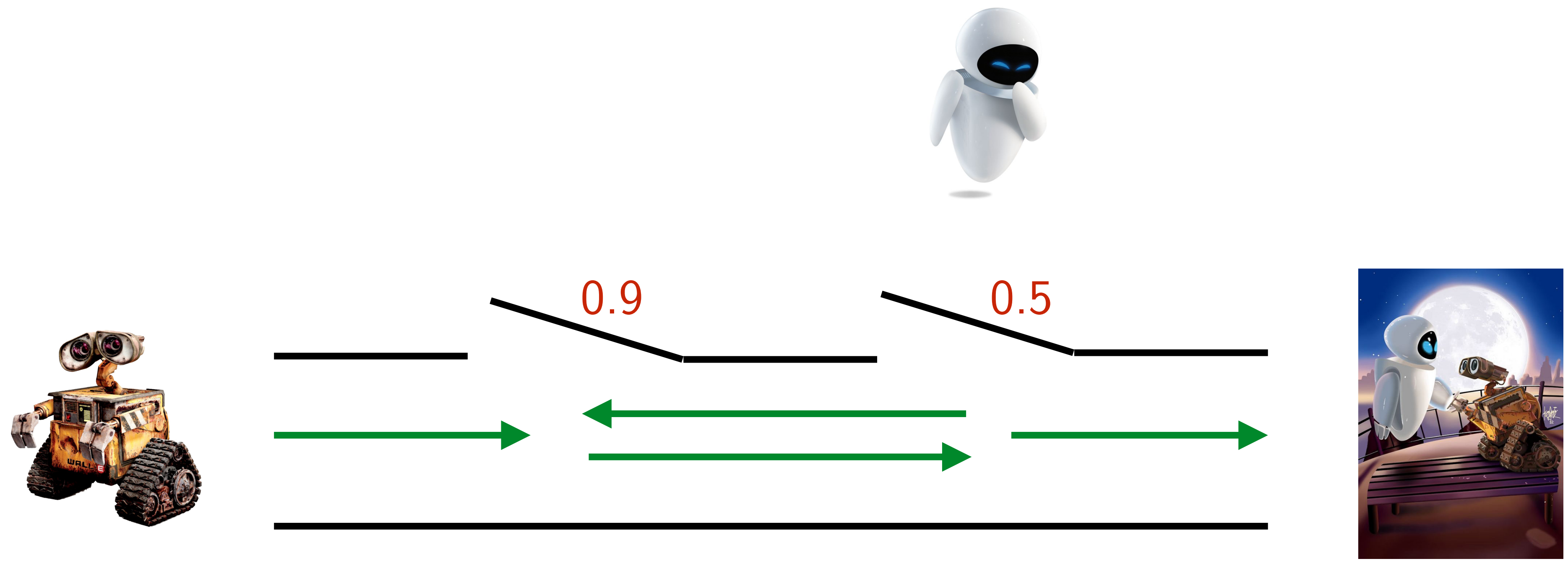
Goal

action $\Longrightarrow$ stochastic environment response

Actions: move left, move right, enter, get Eve, exit

Environment: door possibly jams, …

Planning Under Uncertainty



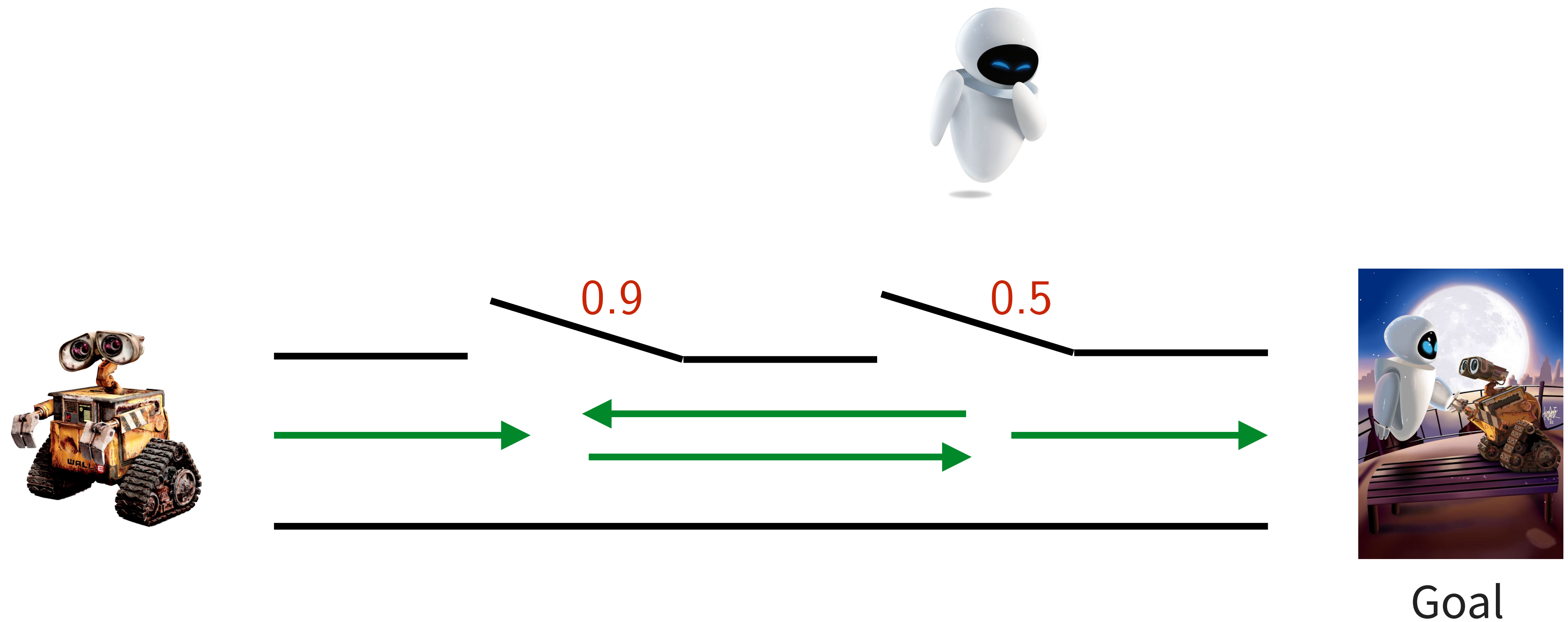
Goal

action $\Longrightarrow$ stochastic environment response

Actions: move left, move right, enter, get Eve, exit

Environment: door possibly jams, …

Planning Under Uncertainty



action $\Longrightarrow$ stochastic environment response

Actions: move left, move right, enter, get Eve, exit

Environment: door possibly jams, …

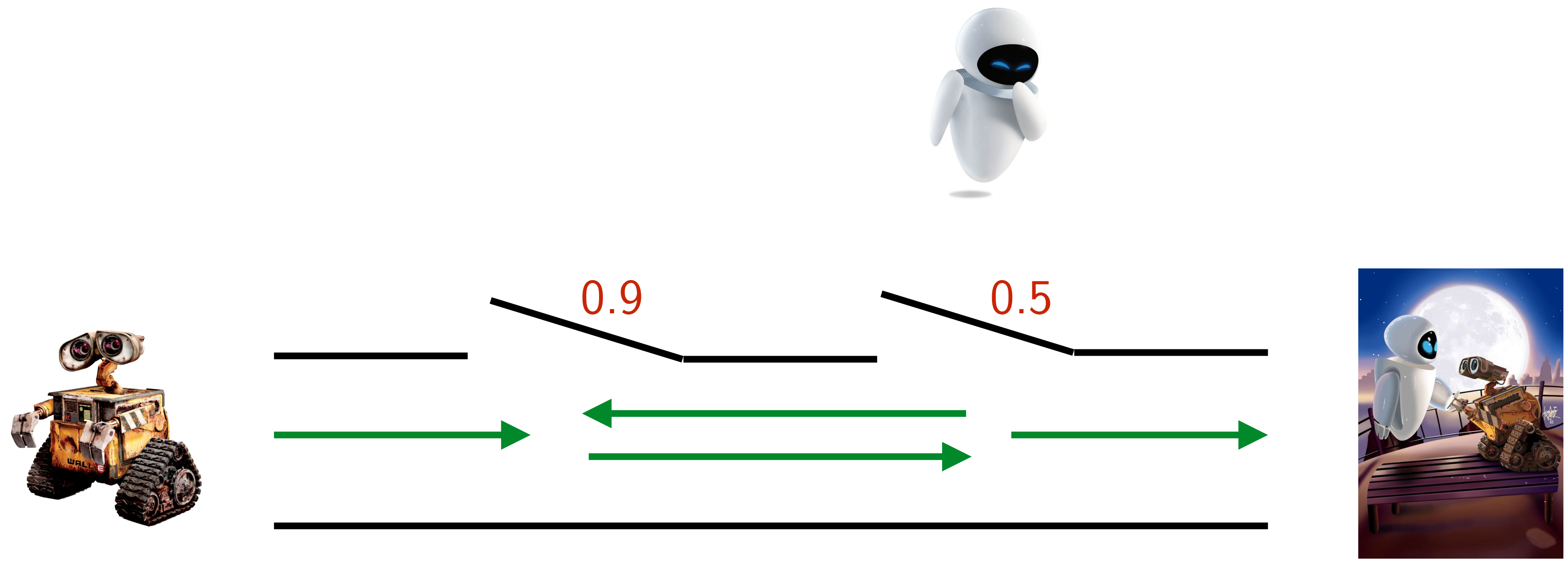
Stochastic Shortest Path Problem (SSP)

Problem: What *action* to take in what *state* to reach the *goal* with *minimal* costs?

Solution: *Stochastic policy*: probability distribution on actions

"When at door 1 enter the room 3 out of 10 times,…"

Planning Under Uncertainty



Goal

action $\Longrightarrow$ stochastic environment response

Actions: move left, move right, enter, get Eve, exit

Environment: door possibly jams, ...

Add constraints for better expressivity (C-SSP)

- well-known: “fuel < 5”

- here: PLTL

Stochastic Shortest Path Problem (SSP)

Problem: What *action* to take in what *state* to reach the *goal* with *minimal* costs?

Solution: *Stochastic policy*: probability distribution on actions

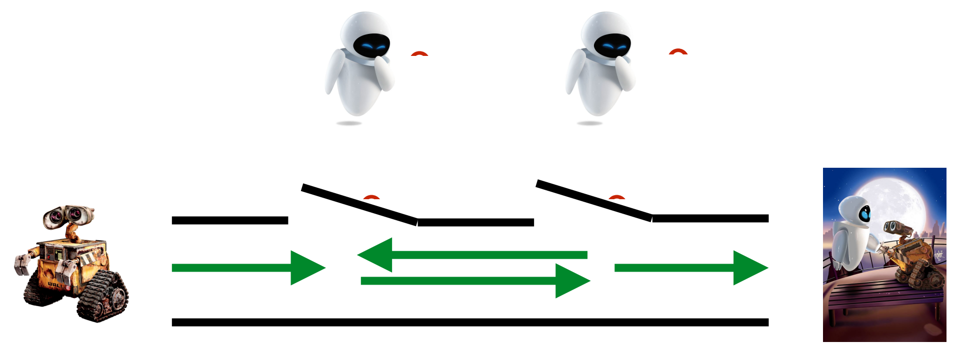
“When at door 1 enter the room 3 out of 10 times, ...”

Multi-Objective Probabilistic LTL (MO-PLTL)

$\psi := \top \mid A \mid \psi \wedge \psi \mid \psi \vee \psi \mid \neg \psi$

$\quad \mid \mathbf{X}\, \psi \mid \psi\, \mathbf{U}\, \psi \mid \mathbf{F}\, \psi \mid \mathbf{G}\, \psi$ (LTL)

$\phi := \mathbf{P}_{>z}\, \psi \mid \mathbf{P}_{\geq z}\, \psi$ (PLTL)



Eve stays in a room until Eve and Wall-E are together

eve_in_a_room **U** together (ψ_1)

Once together, eventually together forever

G (together ⇒ **F G** together) (ψ_2)

Wall-E never visits room1 twice

G (wall-E_room1 ⇒ (wall-E_room1 **U G** ¬wall-E_room1) (ψ_3)

Additional Multi-Objective PLTL Constraint

$\phi = \mathbf{P}_{\geq 0.8}\, \psi_1 \wedge \mathbf{P}_{\geq 1.0}\, \psi_2 \wedge \mathbf{P}_{\geq 0.5}\, \psi_3$ (MO-PLTL)

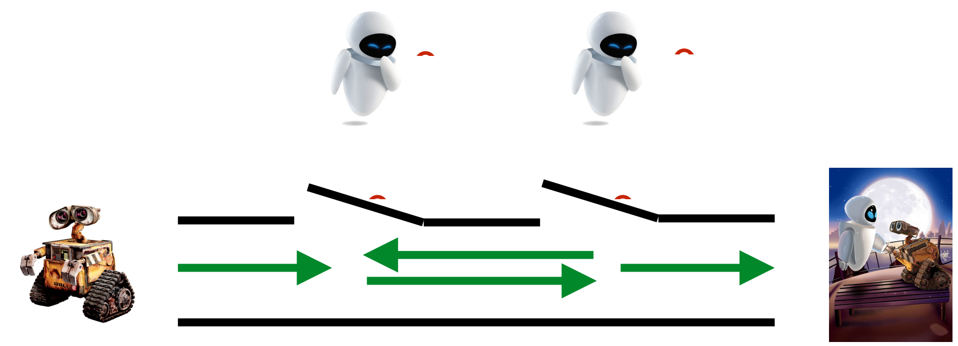
Task: compute a cost-minimal stochastic policy for reaching the goal (with probability 1) such that ϕ is satisfied

Multi-Objective Probabilistic LTL (MO-PLTL)

$\psi := \top \mid A \mid \psi \wedge \psi \mid \psi \vee \psi \mid \neg \psi$

$\quad \mid \mathbf{X}\, \psi \mid \psi\, \mathbf{U}\, \psi \mid \mathbf{F}\, \psi \mid \mathbf{G}\, \psi$ (LTL)

$\phi := \mathbf{P}_{>z}\, \psi \mid \mathbf{P}_{\geq z}\, \psi$ (PLTL)



Eve stays in a room until Eve and Wall-E are together

eve_in_a_room **U** together (ψ_1)

Once together, eventually together forever

G (together ⇒ **F G** together) (ψ_2)

Wall-E never visits room1 twice

G (wall-E_room1 ⇒ (wall-E_room1 **U G** ¬wall-E_room1) (ψ_3)

Not as used in "optimisation"

Additional Multi-Objective PLTL Constraint

$\boldsymbol{\phi} = \mathbf{P}_{\geq 0.8}\, \psi_1 \wedge \mathbf{P}_{\geq 1.0}\, \psi_2 \wedge \mathbf{P}_{\geq 0.5}\, \psi_3$ (MO-PLTL)

Task: compute a cost-minimal stochastic policy for reaching the goal (with probability 1) such that $\boldsymbol{\phi}$ is satisfied

Solving MO-PLTL

Methods Based on Probabilistic Verification

- State of the art method, implemented in PRISM probabilistic model checker
- Needs infinite runs

  (1) add self-loop at Goal

  (2) add Goal constraint : $\boldsymbol{\phi} = \mathbf{P}_1 \, \psi_1 \wedge \cdots \wedge \mathbf{P}_k \, \psi_k \wedge \mathbf{P}_{\geq 1} \, \mathbf{F}$ Goal

- Compute cross-product automaton

  $\mathbf{A} = \mathrm{DRA}(\psi_1) \times \cdots \times \mathrm{DRA}(\psi_k) \times \mathrm{DRA}(\mathbf{F}\ \mathrm{Goal}) \times \mathbf{S}$ (**S** is given state transition system, MDP).

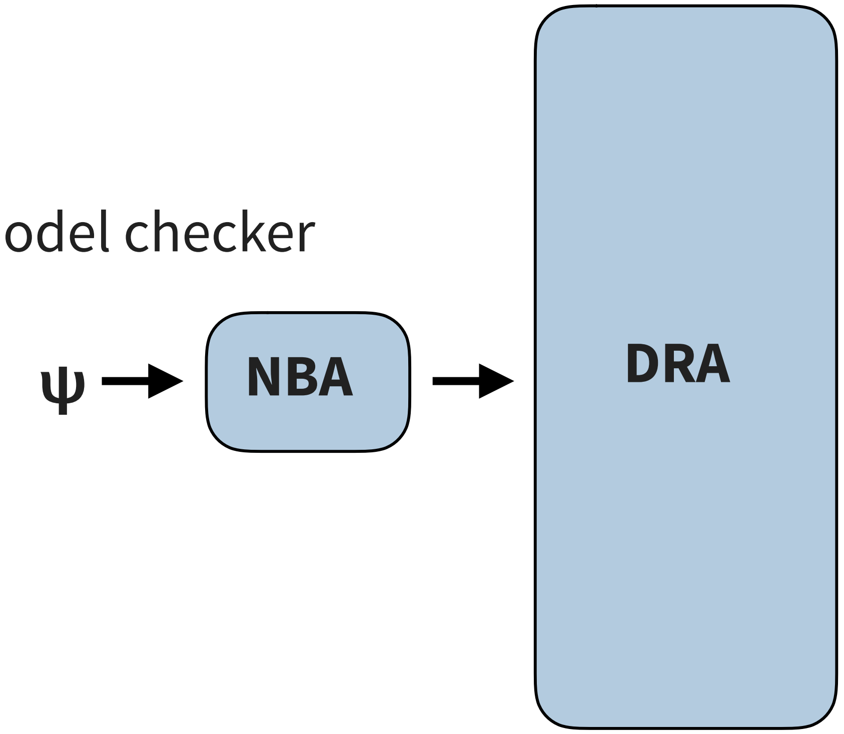
- Obtain policy for $\boldsymbol{\phi}$ as a solution of a certain linear program obtained from **A**

Complexity

- $|\mathrm{DRA}(\psi)|$ is double exponential in $|\psi|$
- |**S**| is usually huge for planning problems - cannot afford to generate in full
- Upfront DRA-computation/crossproduct is problematic even for small examples
- The verification/synthesis problem is 2EXPTIME complete
- Complicated algorithms (see also [deGiacomo&Vardi IJCAI2013, IJCAI2015])

Solving MO-PLTL

Methods Based on Probabilistic Verification



- State of the art method, implemented in PRISM probabilistic model checker
- Needs infinite runs

  (1) add self-loop at Goal

  (2) add Goal constraint : $\boldsymbol{\phi} = \mathbf{P}_1\, \psi_1 \wedge \cdots \wedge \mathbf{P}_k\, \psi_k \wedge \mathbf{P}_{\geq 1}\, \mathbf{F}\ \mathrm{Goal}$

- Compute cross-product automaton

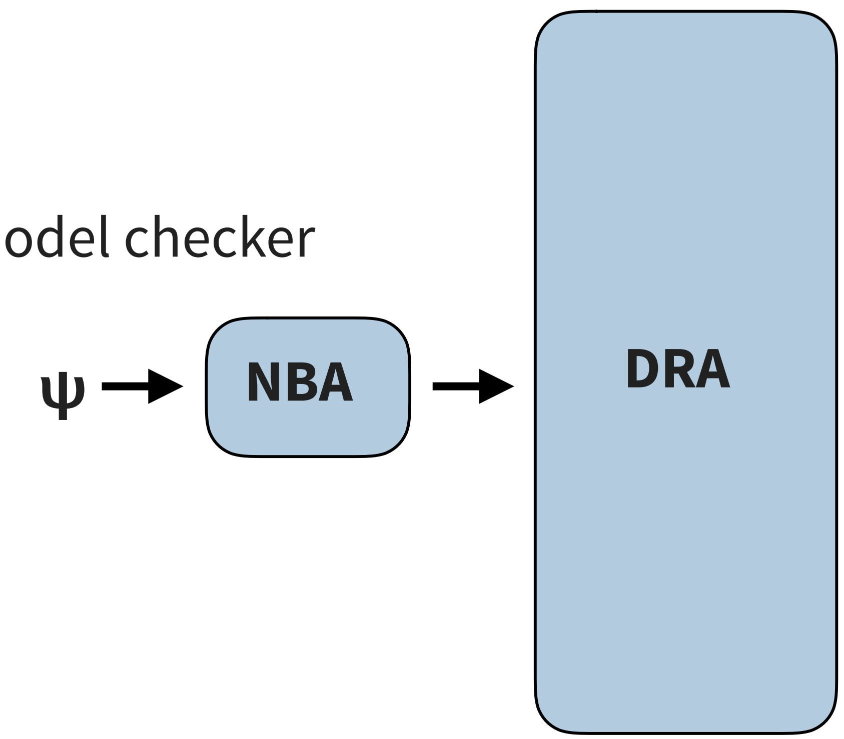
  $\mathbf{A} = \mathrm{DRA}(\psi_1) \times \cdots \times \mathrm{DRA}(\psi_k) \times \mathrm{DRA}(\mathbf{F}\ \mathrm{Goal}) \times \mathbf{S}$ (**S** is given state transition system, MDP).

- Obtain policy for $\boldsymbol{\phi}$ as a solution of a certain linear program obtained from **A**

Complexity

- $|\mathrm{DRA}(\psi)|$ is double exponential in $|\psi|$
- $|\mathbf{S}|$ is usually huge for planning problems - cannot afford to generate in full
- Upfront DRA-computation/crossproduct is problematic even for small examples
- The verification/synthesis problem is 2EXPTIME complete
- Complicated algorithms (see also [deGiacomo&Vardi IJCAI2013, IJCAI2015])

Solving MO-PLTL



Methods Based on Probabilistic Verification

- State of the art method, implemented in PRISM probabilistic model checker
- Needs infinite runs

  (1) add self-loop at Goal

  (2) add Goal constraint : $\boldsymbol{\phi} = \mathbf{P}_1\, \psi_1 \wedge \cdots \wedge \mathbf{P}_k\, \psi_k \wedge \mathbf{P}_{\geq 1}\, \mathbf{F}\ \mathrm{Goal}$

- Compute cross-product automaton

  $\mathbf{A} = \mathrm{DRA}(\psi_1) \times \cdots \times \mathrm{DRA}(\psi_k) \times \mathrm{DRA}(\mathbf{F}\ \mathrm{Goal}) \times \mathbf{S}$ (**S** is given state transition system, MDP).

- Obtain policy for $\boldsymbol{\phi}$ as a solution of a certain linear program obtained from **A**

Complexity

- $|\mathrm{DRA}(\psi)|$ is double exponential in $|\psi|$
- $|\mathbf{S}|$ is usually huge for planning problems - cannot afford to generate in full
- Upfront DRA-computation/crossproduct is problematic even for small examples
- The verification/synthesis problem is 2EXPTIME complete
- Complicated algorithms (see also [deGiacomo&Vardi IJCAI2013, IJCAI2015])

We have a specific problem - all BSCCs are self-loops at goals - and can do better

Contributions

	Verification Based	Our Method
General	Yes	No (Requires Goal)
Approach	Automata (DRA)	(1) Formula progression, Tseitin (2) NBA
State Space	Upfront	On-the-fly
Complexity	Double exponential in ϕ	Single exponential in ϕ for (1)
Heuristics	No	Yes (i^2Dual)

Baier&McIlraith ICAPS 2006: non-stochastic planning w/ LTL, heuristics, NFA, by compilation

Contributions

	Verification Based	Our Method
General	Yes	No (Requires Goal)
Approach	Automata (DRA)	(1) Formula progression, Tseitin (2) NBA
State Space	Upfront	On-the-fly
Complexity	Double exponential in ϕ	Single exponential in ϕ for (1)
Heuristics	No	Yes (i^2Dual)

Baier&McIlraith ICAPS 2006: non-stochastic planning w/ LTL, heuristics, NFA, by compilation

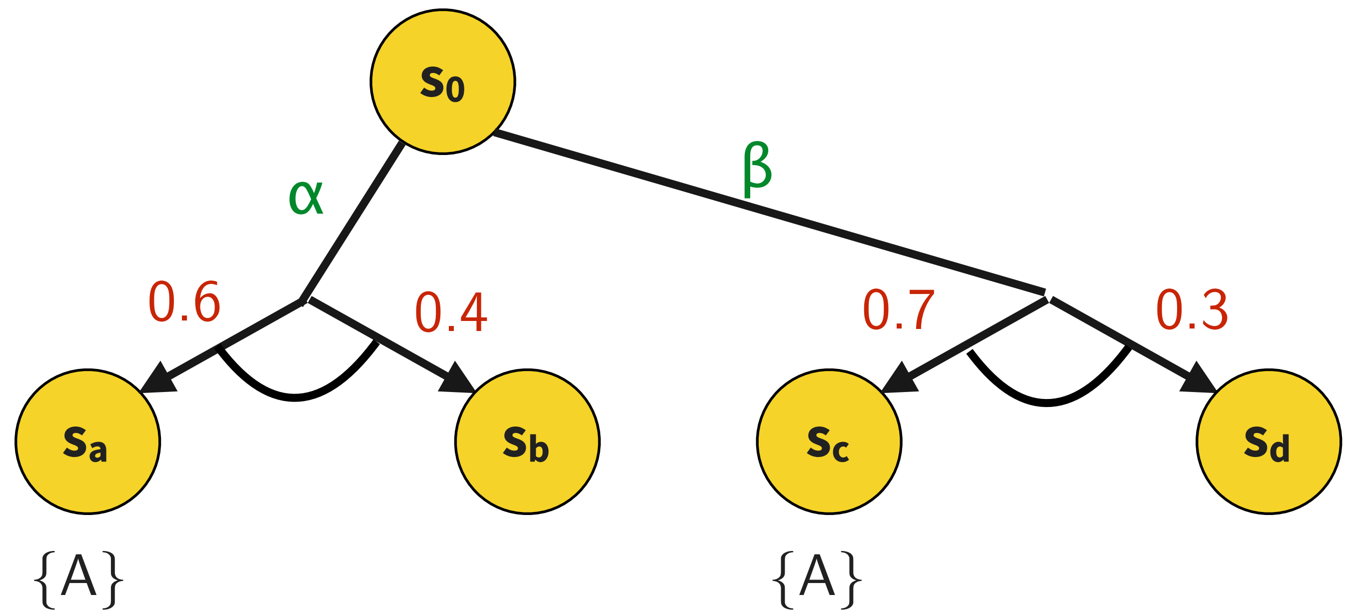
Rest of this talk: approach, complexity, heuristics, experiments

How to Check a Policy π for Satisfying a PLTL Formula

Given policy π =

$\mathbf{s_0}$: [$\alpha \rightarrow 0.6$, $\beta \rightarrow 0.4$]

It follows $\mathbf{s_0} \vDash \mathbf{P}_{>0.6}\ \mathbf{F}\ A$



Proof

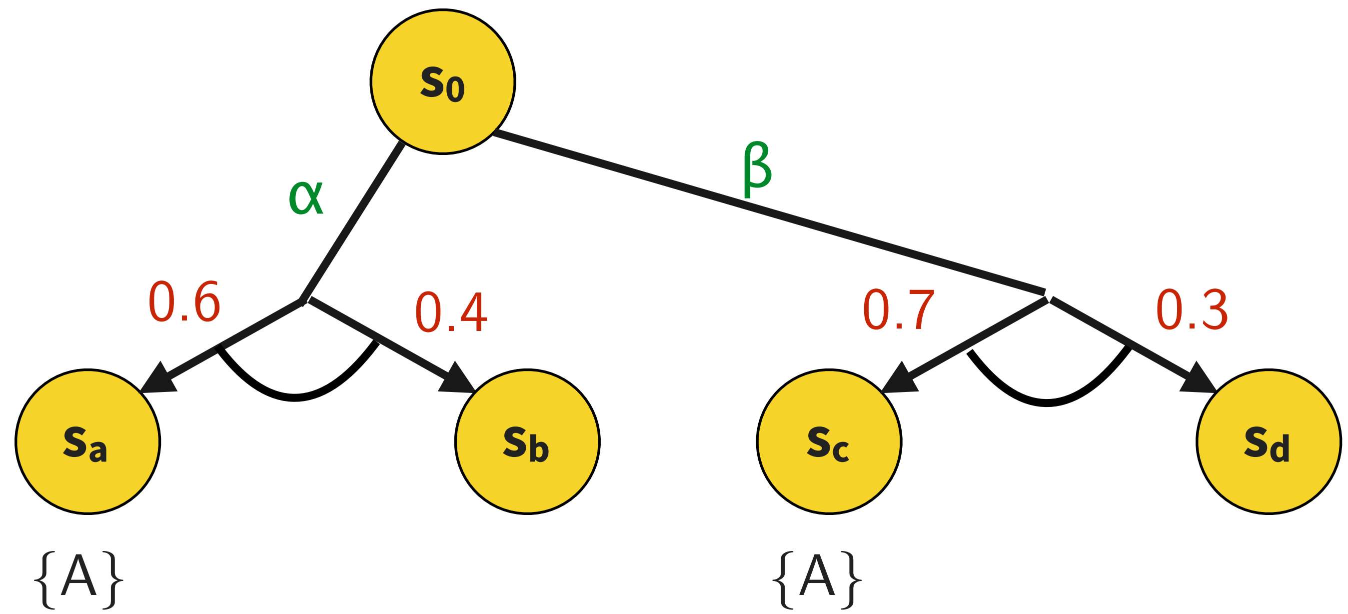
$\mathbf{s_0} \vDash \mathbf{P}_{>0.6}\ \mathbf{F}\ A$ *The probability of all paths from* $\mathbf{s_0}$ *satisfying* **F** A *is* > 0.6

How to Check a Policy π for Satisfying a PLTL Formula

Given policy π =

$\mathbf{s_0}$: [$\alpha \rightarrow 0.6$, $\beta \rightarrow 0.4$]

It follows $\mathbf{s_0} \models \mathbf{P}_{>0.6}\ \mathbf{F}\ A$



Proof

$\mathbf{s_0} \models \mathbf{P}_{>0.6}\ \mathbf{F}\ A$ *The probability of all paths from* $\mathbf{s_0}$ *satisfying* $\mathbf{F}$ A *is* > 0.6

iff

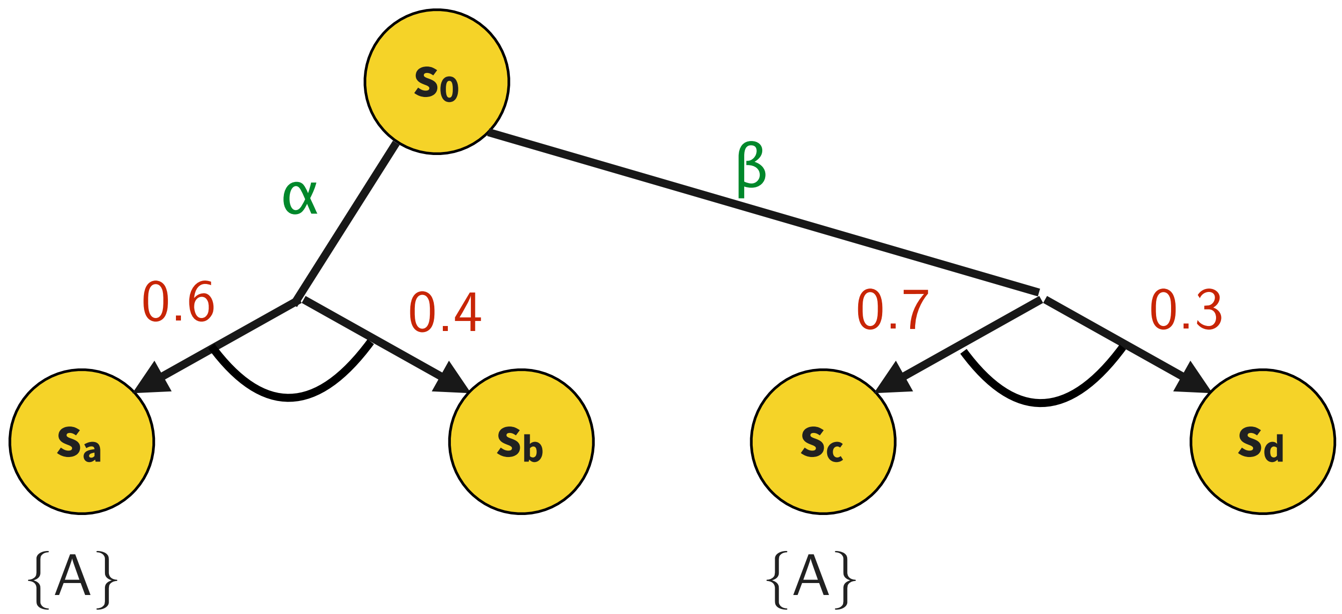
$\Pr\{p \mid p \text{ is a path from } \mathbf{s_0} \text{ and } p \models \mathbf{F}\ A\} > 0.6$

How to Check a Policy π for Satisfying a PLTL Formula

Given policy π =

$\mathbf{s_0}$: [$\alpha \to 0.6$, $\beta \to 0.4$]

It follows $\mathbf{s_0} \models \mathbf{P}_{>0.6}\ \mathbf{F}\ \mathrm{A}$



Proof

$\mathbf{s_0} \models \mathbf{P}_{>0.6}\ \mathbf{F}\ \mathrm{A}$ *The probability of all paths from* $\mathbf{s_0}$ *satisfying* **F** A *is* > 0.6

iff

$\Pr\{p \mid p \text{ is a path from } \mathbf{s_0} \text{ and } p \models \mathbf{F}\ \mathrm{A}\} > 0.6$

Non-probabilistic LTL

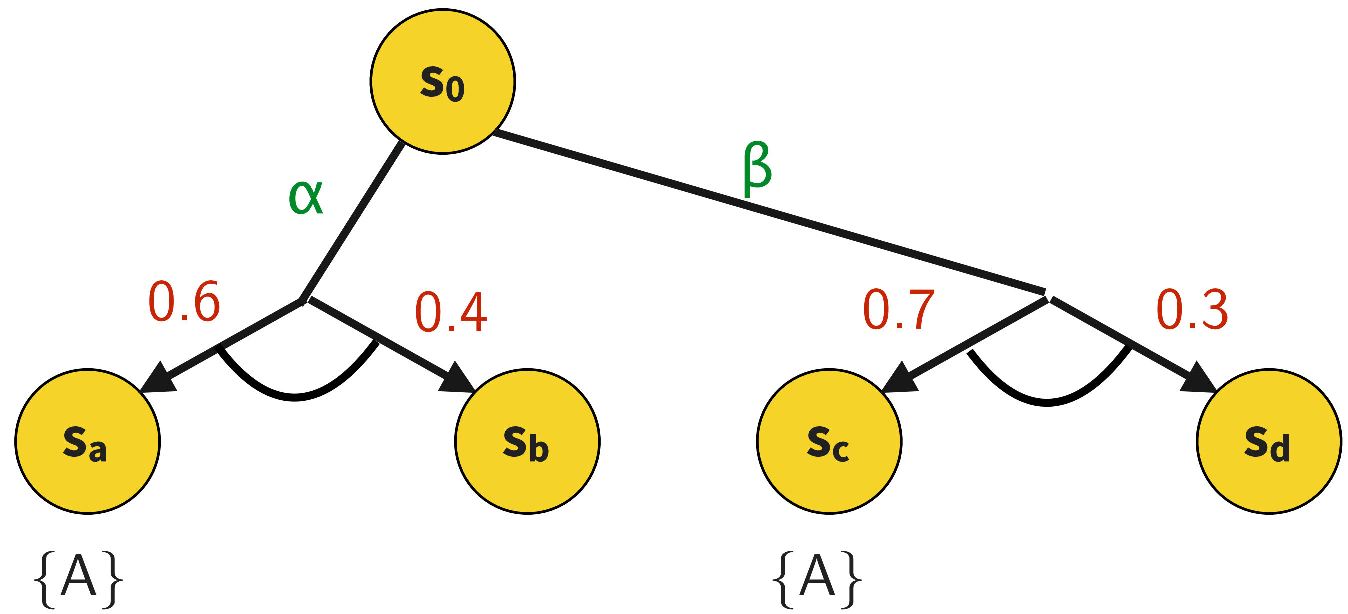
Ignore finiteness of paths on this slide

How to Check a Policy π for Satisfying a PLTL Formula

Given policy π =

$\mathbf{s_0}$: [α → 0.6, β → 0.4]

It follows $\mathbf{s_0} \vDash \mathbf{P}_{>0.6}\ \mathbf{F}\ A$



Proof

$\mathbf{s_0} \vDash \mathbf{P}_{>0.6}\ \mathbf{F}\ A$

The probability of all paths from $\mathbf{s_0}$ *satisfying* **F** A *is* > 0.6

iff

$\Pr\{p \mid p \text{ is a path from } \mathbf{s_0} \text{ and } p \vDash \mathbf{F}\ A\} > 0.6$

iff

$\Pr\{\mathbf{s_0 s_a},\ \mathbf{s_0 s_c}\} > 0.6$

Non-probabilistic LTL

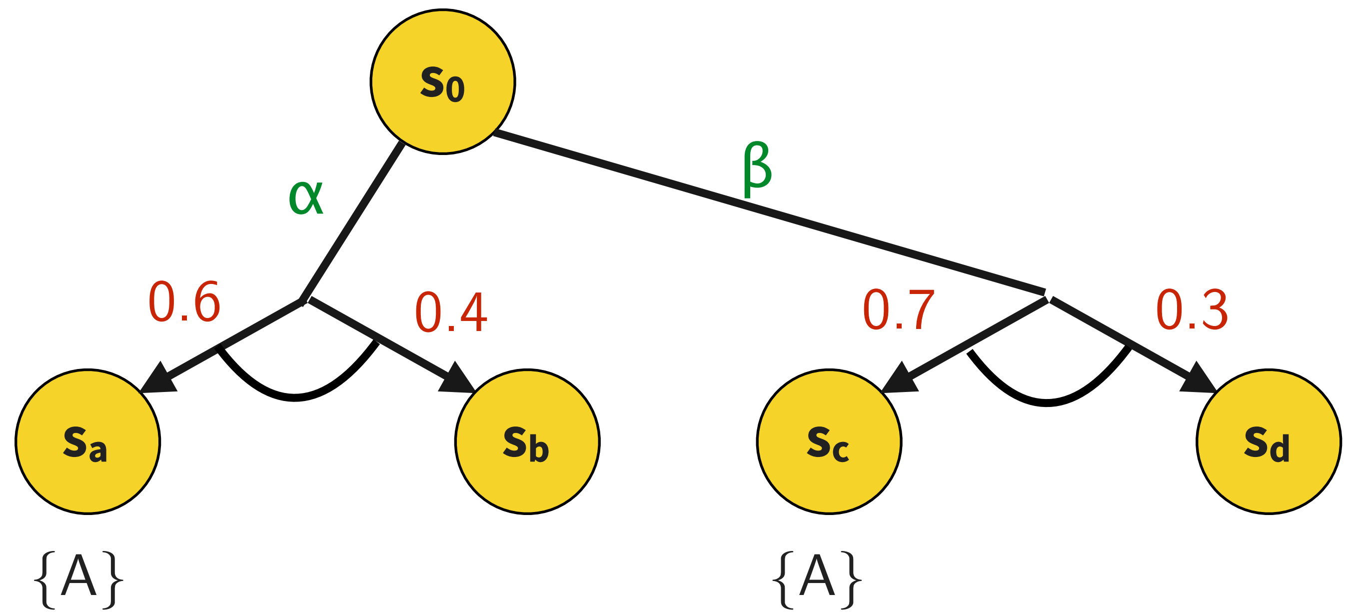
Ignore finiteness of paths on this slide

How to Check a Policy π for Satisfying a PLTL Formula

Given policy π =

$\mathbf{s_0}$: $[\alpha \to 0.6,\ \beta \to 0.4\ ]$

It follows $\mathbf{s_0} \models \mathbf{P}_{>0.6}\ \mathbf{F}\ A$



Proof

$\mathbf{s_0} \models \mathbf{P}_{>0.6}\ \mathbf{F}\ A$ *The probability of all paths from* $\mathbf{s_0}$ *satisfying* **F** A *is* > 0.6

iff

$\Pr\{p \mid p \text{ is a path from } \mathbf{s_0} \text{ and } p \models \mathbf{F}\ A\} > 0.6$

iff

Non-probabilistic LTL

Ignore finiteness of paths on this slide

$\Pr\{\mathbf{s_0 s_a},\ \mathbf{s_0 s_c}\} > 0.6$

iff

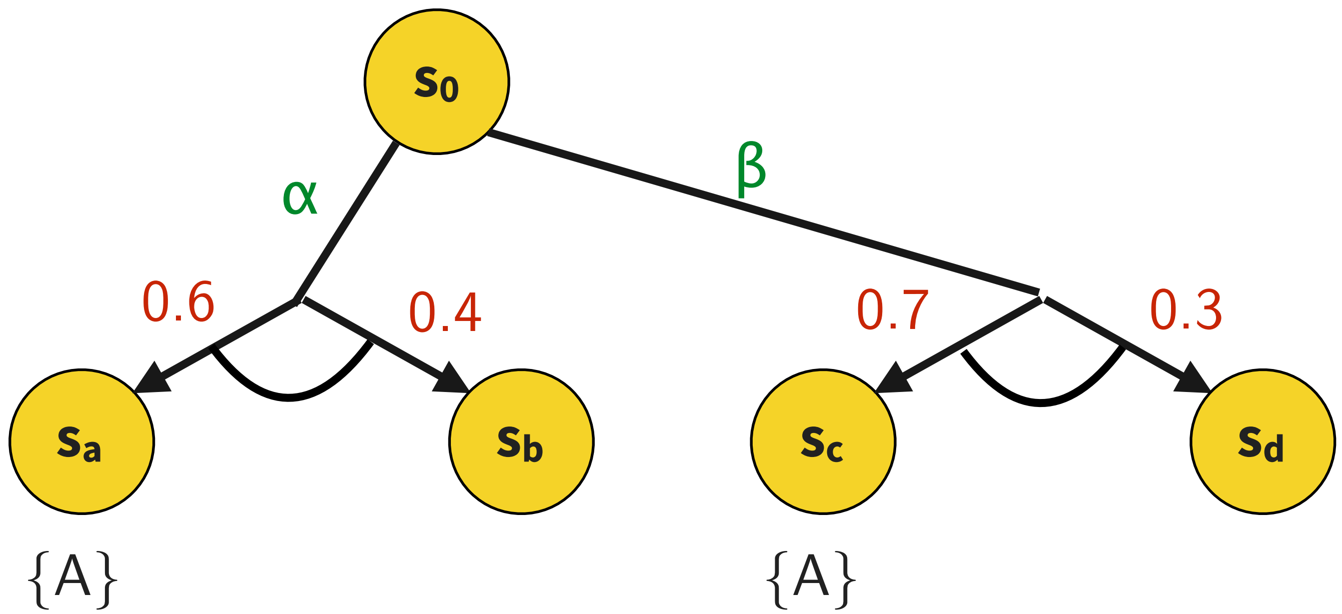
$0.6 \cdot 0.6 + 0.4 \cdot 0.7 = 0.64 > 0.6$

How to Synthesize Policy π for Satisfying a PLTL Formula

Given policy π =

$\mathbf{s_0}$: $[\alpha \rightarrow 0.6,\ \beta \rightarrow 0.4\ ]$

It follows $\mathbf{s_0} \vDash \mathbf{P}_{>0.6}\ \mathbf{F}\ \mathrm{A}$



Proof

$\mathbf{s_0} \vDash \mathbf{P}_{>0.6}\ \mathbf{F}\ \mathrm{A}$ *The probability of all paths from* $\mathbf{s_0}$ *satisfying* **F** A *is* > 0.6

iff

$\Pr\{p \mid p \text{ is a path from } \mathbf{s_0} \text{ and } p \vDash \mathbf{F}\ \mathrm{A}\} > 0.6$

iff

Non-probabilistic LTL

Ignore finiteness of paths on this slide

$\Pr\{\mathbf{s_0 s_a},\ \mathbf{s_0 s_c}\} > 0.6$

iff

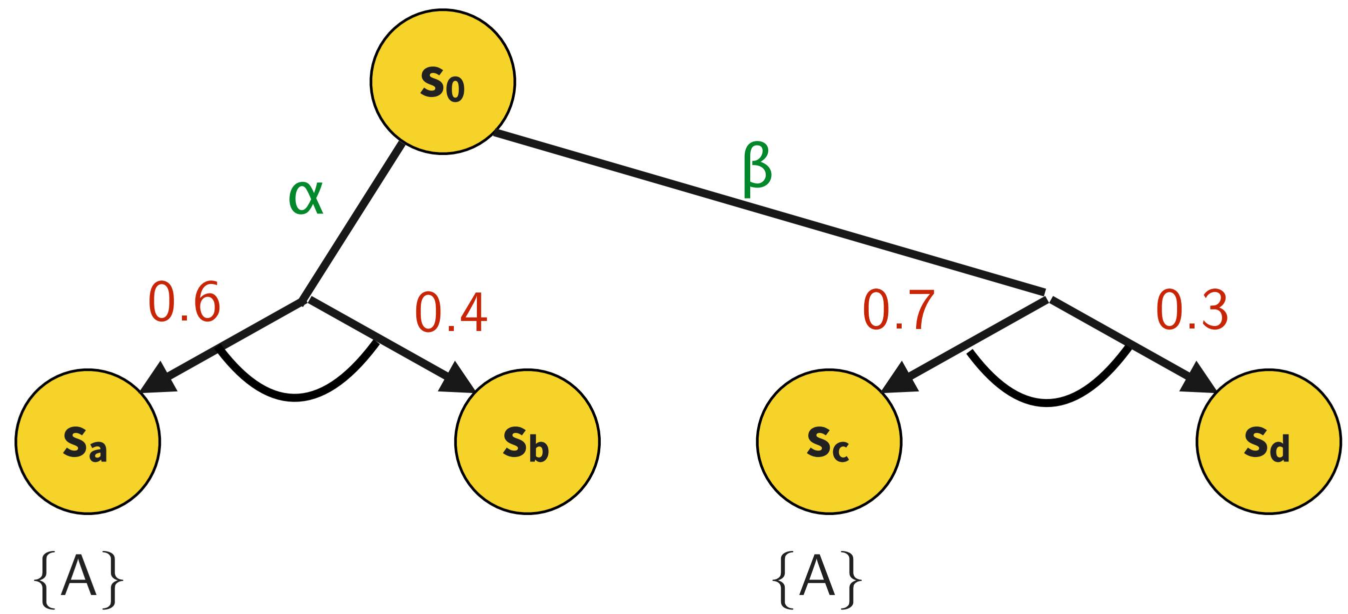
$0.6 \cdot 0.6 + 0.4 \cdot 0.7 = 0.64 > 0.6$

How to Synthesize Policy π for Satisfying a PLTL Formula

Find policy π =

s_0: [$\alpha \rightarrow 0.6$, $\beta \rightarrow 0.4$]

It follows $s_0 \models \mathbf{P}_{>0.6}\ \mathbf{F}\ A$



Proof

$s_0 \models \mathbf{P}_{>0.6}\ \mathbf{F}\ A$ *The probability of all paths from* s_0 *satisfying* **F** A *is* > 0.6

iff

$\Pr\{p \mid p \text{ is a path from } s_0 \text{ and } p \models \mathbf{F}\ A\} > 0.6$

iff

Non-probabilistic LTL

Ignore finiteness of paths on this slide

$\Pr\{s_0 s_a, s_0 s_c\} > 0.6$

iff

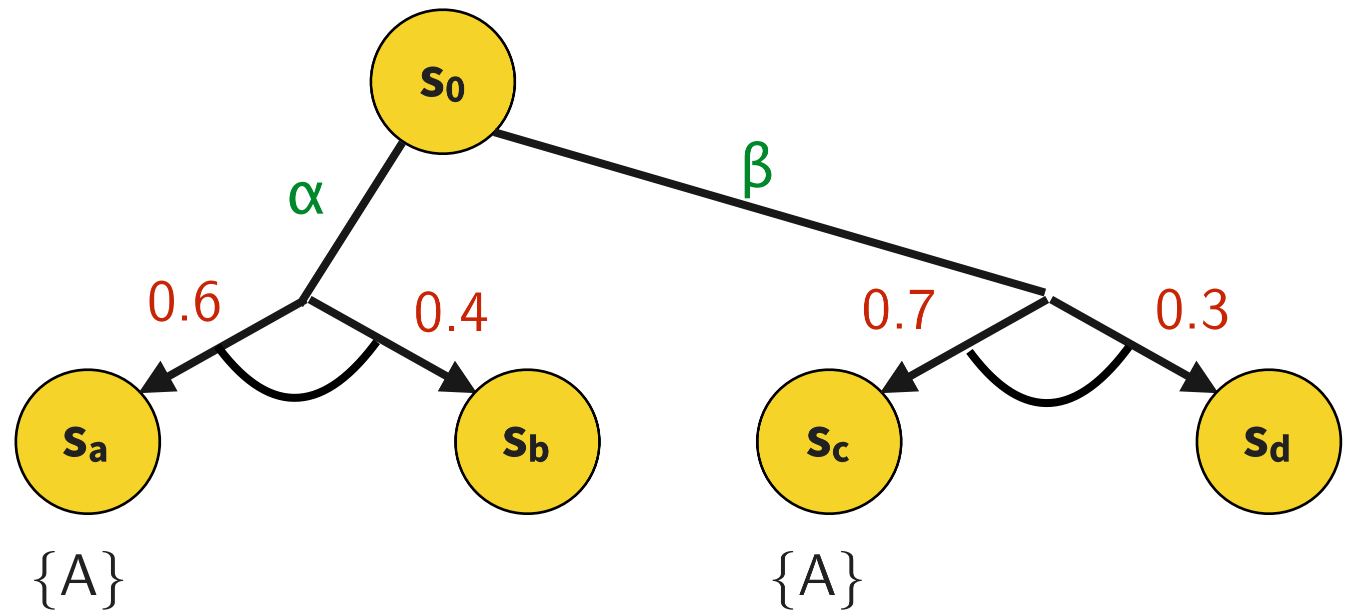
$0.6 \cdot 0.6 + 0.4 \cdot 0.7 = 0.64 > 0.6$

How to Synthesize Policy π for Satisfying a PLTL Formula

Find policy π =

$\mathbf{s_0}$: $[\alpha \to \quad, \beta \to \quad]$

It follows $\mathbf{s_0} \models \mathbf{P}_{>0.6}\ \mathbf{F}\ \mathrm{A}$



Proof

$\mathbf{s_0} \models \mathbf{P}_{>0.6}\ \mathbf{F}\ \mathrm{A}$ *The probability of all paths from* $\mathbf{s_0}$ *satisfying* **F** A *is* > 0.6

iff

$\Pr\{p \mid p \text{ is a path from } \mathbf{s_0} \text{ and } p \models \mathbf{F}\ \mathrm{A}\} > 0.6$

iff

$\Pr\{\mathbf{s_0 s_a}, \mathbf{s_0 s_c}\} > 0.6$

iff

$0.6 \cdot 0.6 + 0.4 \cdot 0.7 = 0.64 > 0.6$

Non-probabilistic LTL

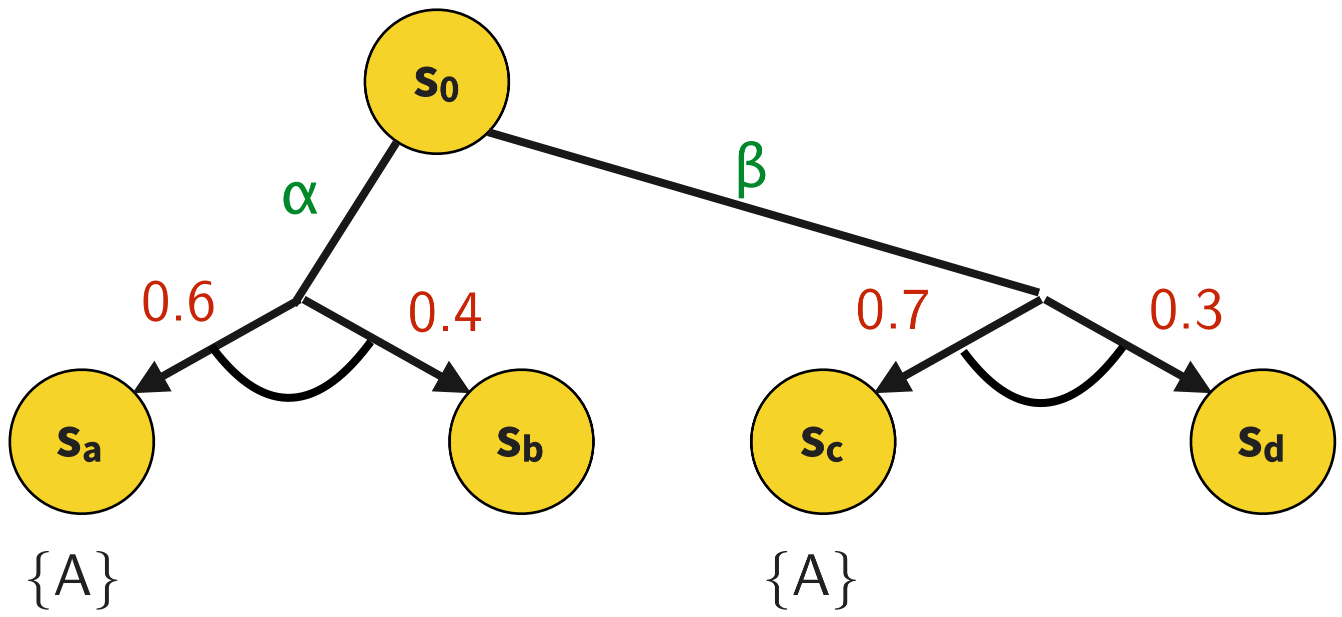
Ignore finiteness of paths on this slide

How to **Synthesize** Policy π for Satisfying a PLTL Formula

Find policy π =

$\mathbf{s_0}$: [$\alpha \rightarrow$, $\beta \rightarrow$]

such that $\mathbf{s_0} \models \mathbf{P}_{>0.6}\ \mathbf{F}\ A$



Proof

$\mathbf{s_0} \models \mathbf{P}_{>0.6}\ \mathbf{F}\ A$ *The probability of all paths from* $\mathbf{s_0}$ *satisfying* **F** A *is* > 0.6

iff

$\Pr\{p \mid p \text{ is a path from } \mathbf{s_0} \text{ and } p \models \mathbf{F}\ A\} > 0.6$

iff

$\Pr\{\mathbf{s_0 s_a},\ \mathbf{s_0 s_c}\} > 0.6$

Non-probabilistic LTL

Ignore finiteness of paths on this slide

iff

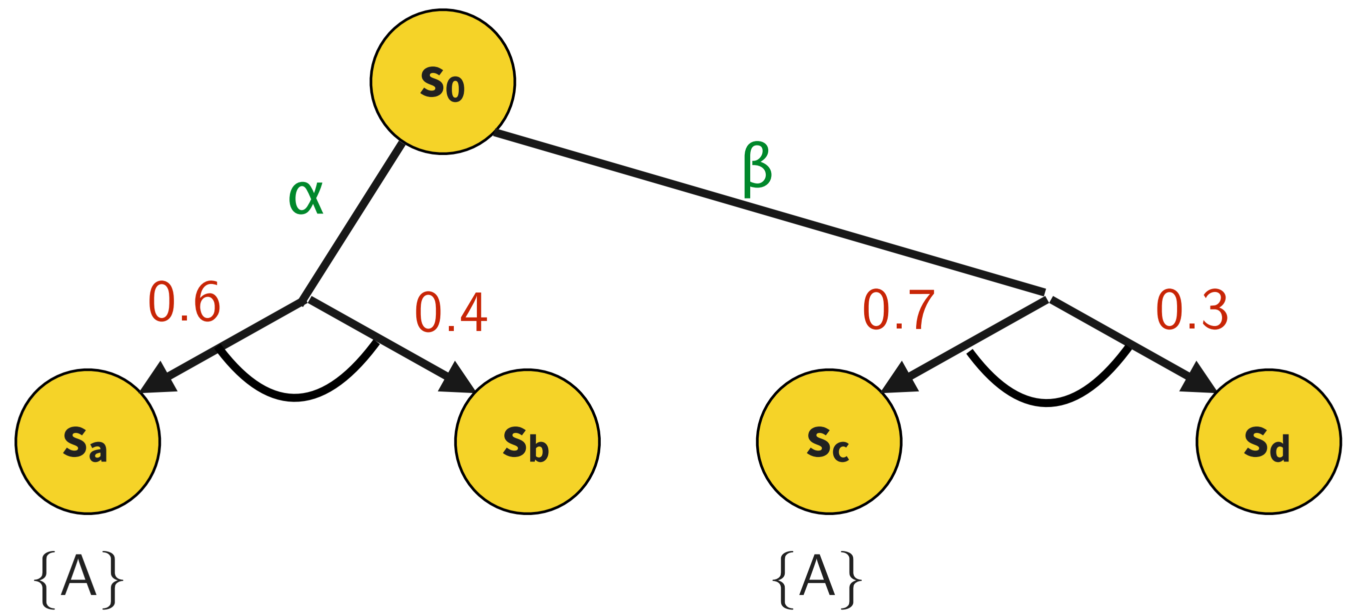
$0.6 \cdot 0.6 + 0.4 \cdot 0.7 = 0.64 > 0.6$

How to **Synthesize** Policy π for Satisfying a PLTL Formula

Find policy π =

$\mathbf{s_0}$: [$\alpha \rightarrow$, $\beta \rightarrow$]

such that $\mathbf{s_0} \vDash \mathbf{P}_{>0.6}\ \mathbf{F}\ \mathsf{A}$



Proof

$\mathbf{s_0} \vDash \mathbf{P}_{>0.6}\ \mathbf{F}\ \mathsf{A}$

The probability of all paths from $\mathbf{s_0}$ *satisfying* **F** A *is* > 0.6

iff

$\mathsf{Pr}\{\mathsf{p} \mid \mathsf{p}\ \text{is a path from}\ \mathbf{s_0}\ \text{and}\ \mathsf{p} \vDash \mathbf{F}\ \mathsf{A}\} > 0.6$

iff

Non-probabilistic LTL

Ignore finiteness of paths on this slide

$\mathsf{Pr}\{\mathbf{s_0 s_a},\ \mathbf{s_0 s_c}\} > 0.6$

iff

$0.6 \quad \cdot 0.6 + \quad 0.4 \quad \cdot 0.7 = 0.64 > 0.6$

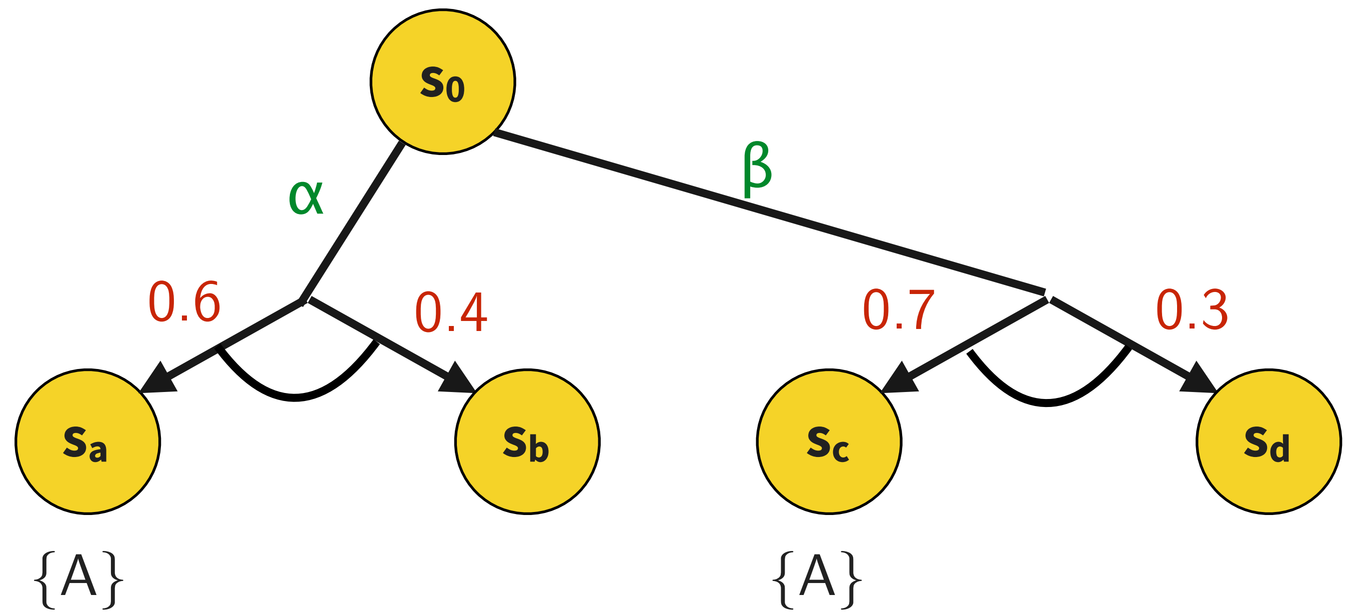
→ **Quantify over action probabilities and compute solution**

How to **Synthesize** Policy π for Satisfying a PLTL Formula

Find policy π =

s_0: [$\alpha \rightarrow$, $\beta \rightarrow$]

such that $s_0 \vDash \mathbf{P}_{>0.6}\ \mathbf{F}\ A$



Proof

$s_0 \vDash \mathbf{P}_{>0.6}\ \mathbf{F}\ A$

The probability of all paths from s_0 *satisfying* **F** A *is* > 0.6

iff

$\Pr\{p \mid p \text{ is a path from } s_0 \text{ and } p \vDash \mathbf{F}\ A\} > 0.6$

iff

$\Pr\{s_0 s_a,\ s_0 s_c\} > 0.6$

Non-probabilistic LTL

Ignore finiteness of paths on this slide

iff

$\pi(\alpha \mid s_0)\ 0.6 + \pi(\beta \mid s_0)\ 0.7 \quad > 0.6$

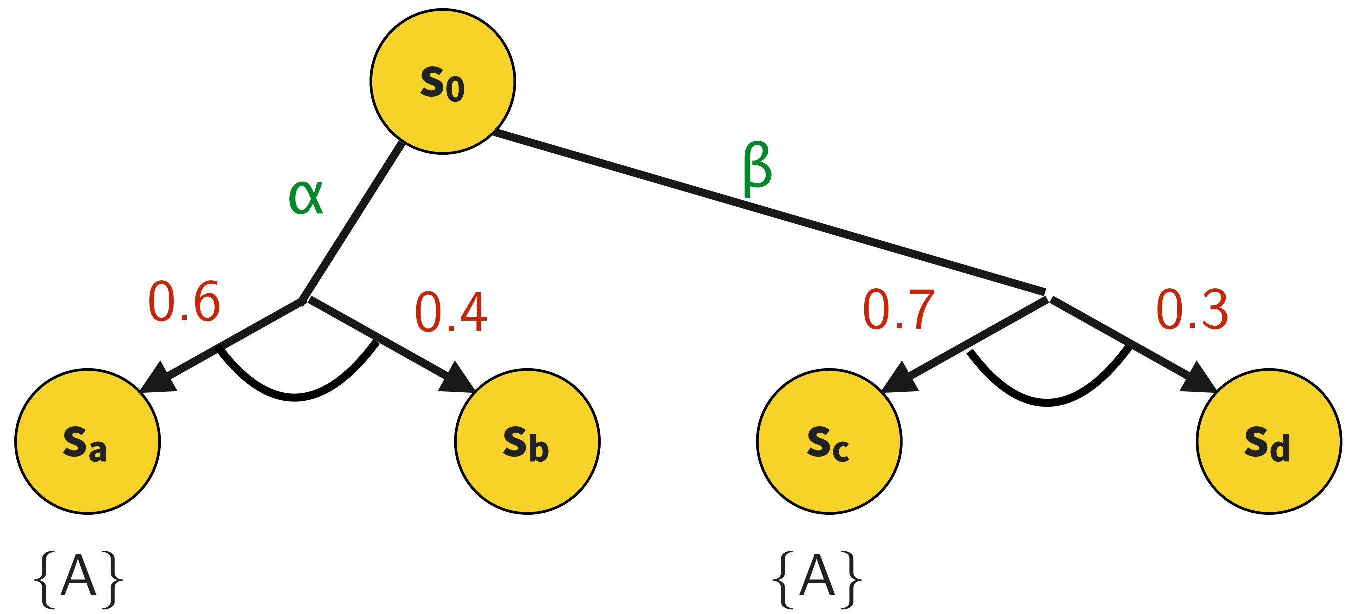
→ **Quantify over action probabilities and compute solution**

How to **Synthesize** Policy π for Satisfying a PLTL Formula

Find policy π =

s_0: [α → , β →]

such that $s_0 \models \mathbf{P}_{>0.6}\ \mathbf{F}\ A$



Proof

$s_0 \models \mathbf{P}_{>0.6}\ \mathbf{F}\ A$

The probability of all paths from s_0 *satisfying* **F** A *is* > 0.6

iff

$\Pr\{p \mid p \text{ is a path from } s_0 \text{ and } p \models \mathbf{F}\ A\} > 0.6$

Non-probabilistic LTL

Ignore finiteness of paths on this slide

iff

$\Pr\{s_0 s_a,\ s_0 s_c\} > 0.6$

iff

$\pi(\alpha \mid s_0)\ 0.6 + \pi(\beta \mid s_0)\ 0.7 \quad > 0.6$

$\pi(\alpha \mid s_0) + \pi(\beta \mid s_0) = 1$

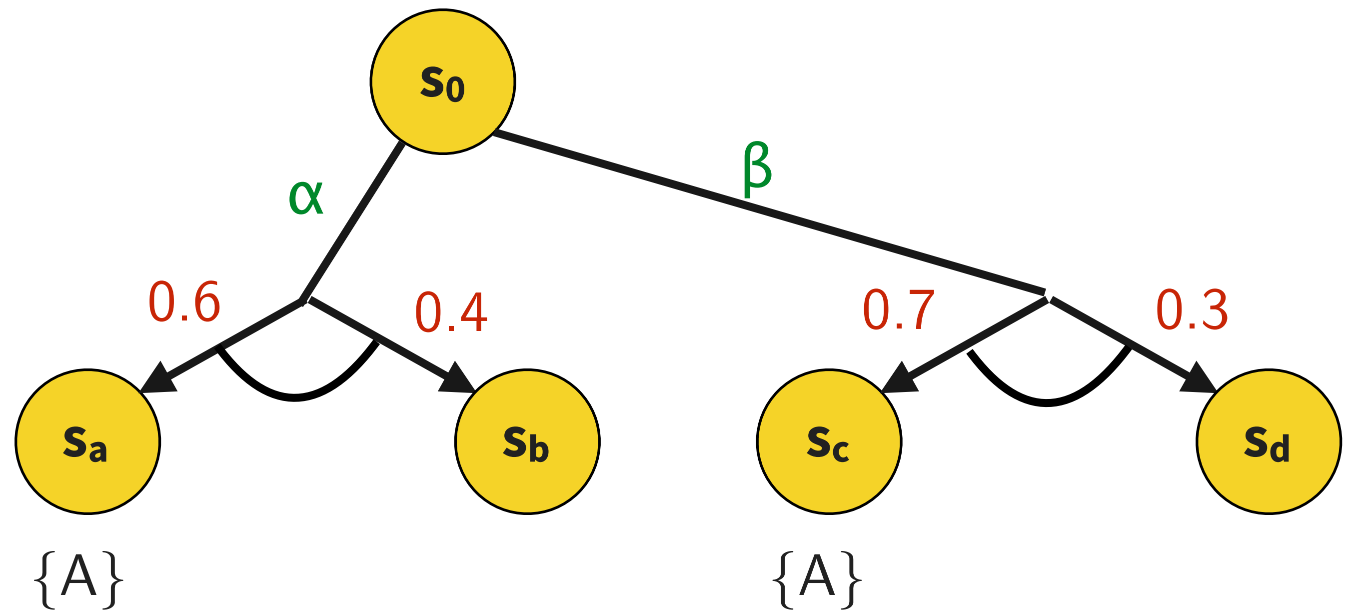
→ **Quantify over action probabilities and compute solution**

How to Synthesize Policy π for Satisfying a PLTL Formula

Find policy π =

s_0: [α → **0.6** β → **0.4**]

such that $s_0 \models \mathbf{P}_{>0.6}\ \mathbf{F}\ A$



Proof

$s_0 \models \mathbf{P}_{>0.6}\ \mathbf{F}\ A$ *The probability of all paths from* s_0 *satisfying* **F** A *is* > 0.6

iff

$\Pr\{p \mid p \text{ is a path from } s_0 \text{ and } p \models \mathbf{F}\ A\} > 0.6$

iff

Non-probabilistic LTL

Ignore finiteness of paths on this slide

$\Pr\{s_0 s_a,\ s_0 s_c\} > 0.6$

iff

$\pi(\alpha \mid s_0)\ 0.6 + \pi(\beta \mid s_0)\ 0.7 \quad > 0.6$

$\pi(\alpha \mid s_0) + \pi(\beta \mid s_0) = 1$

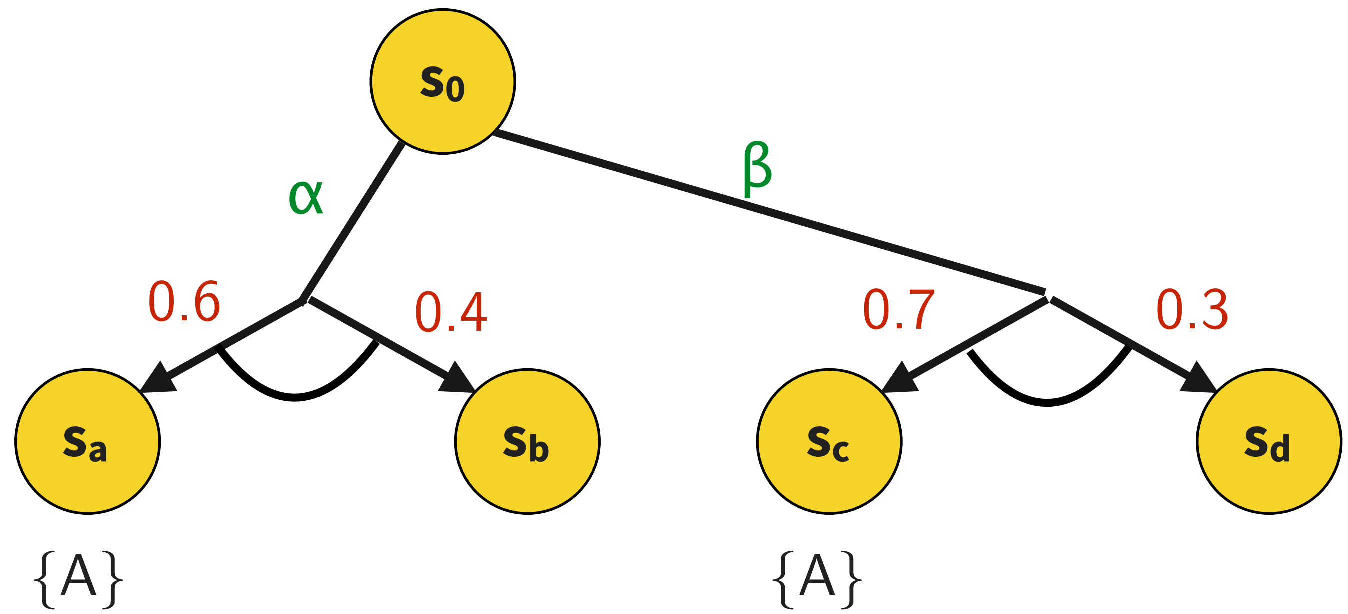
→ **Quantify over action probabilities and compute solution**

How to **Synthesize** Policy π for Satisfying a PLTL Formula

Find policy π =

s_0: [$\alpha \to$ **0.6** $\beta \to$ **0.4**]

such that $s_0 \models \mathbf{P}_{>0.6}\ \mathbf{F}\ A$



Proof

$s_0 \models \mathbf{P}_{>0.6}\ \mathbf{F}\ A$

The probability of all paths from s_0 *satisfying* **F** A *is* > 0.6

iff

$\Pr\{p \mid p \text{ is a path from } s_0 \text{ and } p \models \mathbf{F}\ A\} > 0.6$

iff

Non-probabilistic LTL

Ignore finiteness of paths on this slide

$\Pr\{s_0 s_a,\ s_0 s_c\} > 0.6$

iff

$\pi(\alpha \mid s_0)\ 0.6 + \pi(\beta \mid s_0)\ 0.7 > 0.6$

→ **Quantify over action probabilities and compute solution**

$\pi(\alpha \mid s_0) + \pi(\beta \mid s_0) = 1$

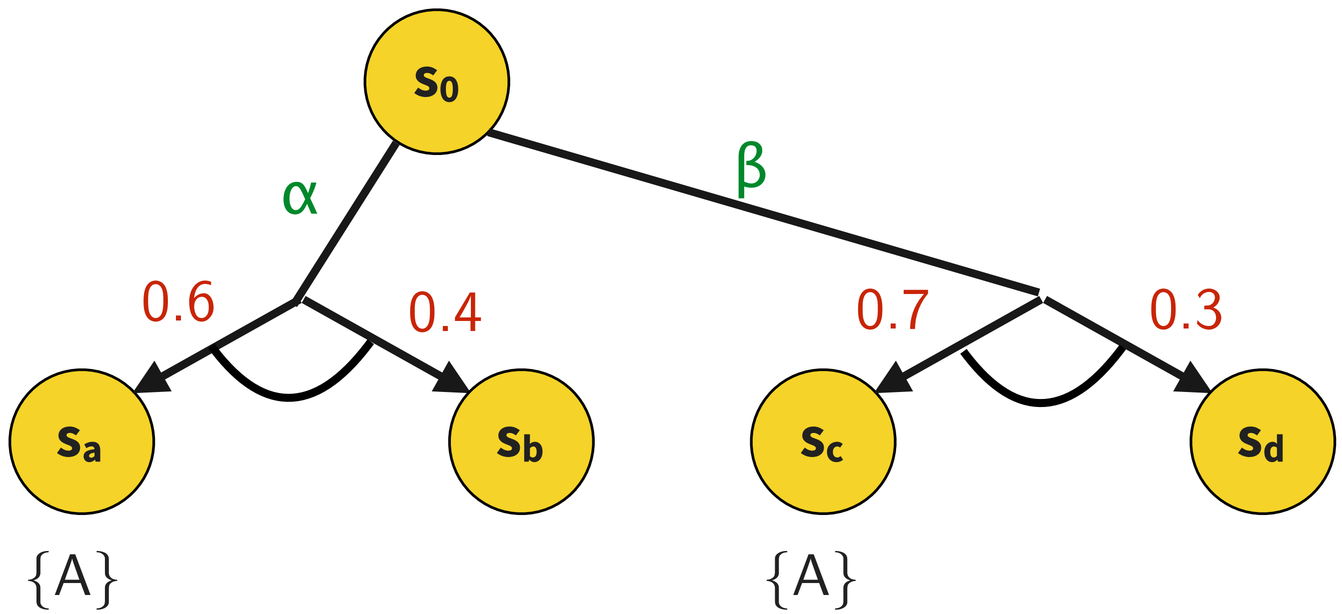
Non-linear program in general - we use dual-space LPs instead

How to **Synthesize** Policy π for Satisfying a PLTL Formula

Find policy π =

s_0: [$\alpha \to$ **0.6** $\beta \to$ **0.4**]

such that $s_0 \models \mathbf{P}_{>0.6}\ \mathbf{F}\ A$



Proof

$s_0 \models \mathbf{P}_{>0.6}\ \mathbf{F}\ A$ *The probability of all paths from* s_0 *satisfying* **F** A *is* > 0.6

iff

$\Pr\{p \mid p \text{ is a path from } s_0 \text{ and } p \models \mathbf{F}\ A\} > 0.6$

iff

$\Pr\{s_0 s_a,\ s_0 s_c\} > 0.6$

iff

$\pi(\alpha \mid s_0)\ 0.6 + \pi(\beta \mid s_0)\ 0.7 > 0.6$

$\pi(\alpha \mid s_0) + \pi(\beta \mid s_0) = 1$

→ **Quantify over action probabilities and compute solution**

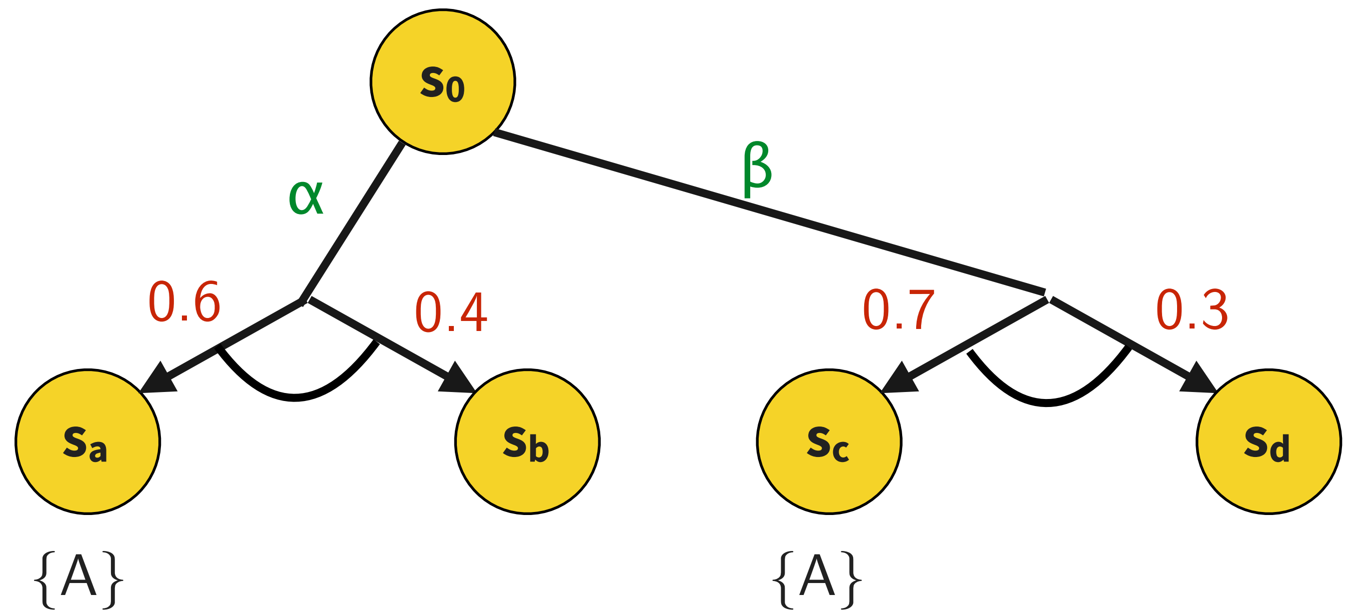
Non-linear program in general - we use dual-space LPs instead

How to **Synthesize** Policy π for Satisfying a PLTL Formula

Find policy π =

s_0: [α → **0.6** β → **0.4**]

such that $s_0 \models \mathbf{P}_{>0.6}\ \mathbf{F}\ A$



Proof

$s_0 \models \mathbf{P}_{>0.6}\ \mathbf{F}\ A$

The probability of all paths from s_0 *satisfying* **F** A *is* > 0.6

iff

$\Pr\{p \mid p \text{ is a path from } s_0 \text{ and } p \models \mathbf{F}\ A\} > 0.6$

Contributions

(1) Formula progression, or

(2) NBA mode

iff

$\Pr\{s_0 s_a,\ s_0 s_c\} > 0.6$

iff

$\pi(\alpha \mid s_0)\ 0.6 + \pi(\beta \mid s_0)\ 0.7 > 0.6$

→ **Quantify over action probabilities and compute solution**

$\pi(\alpha \mid s_0) + \pi(\beta \mid s_0) = 1$

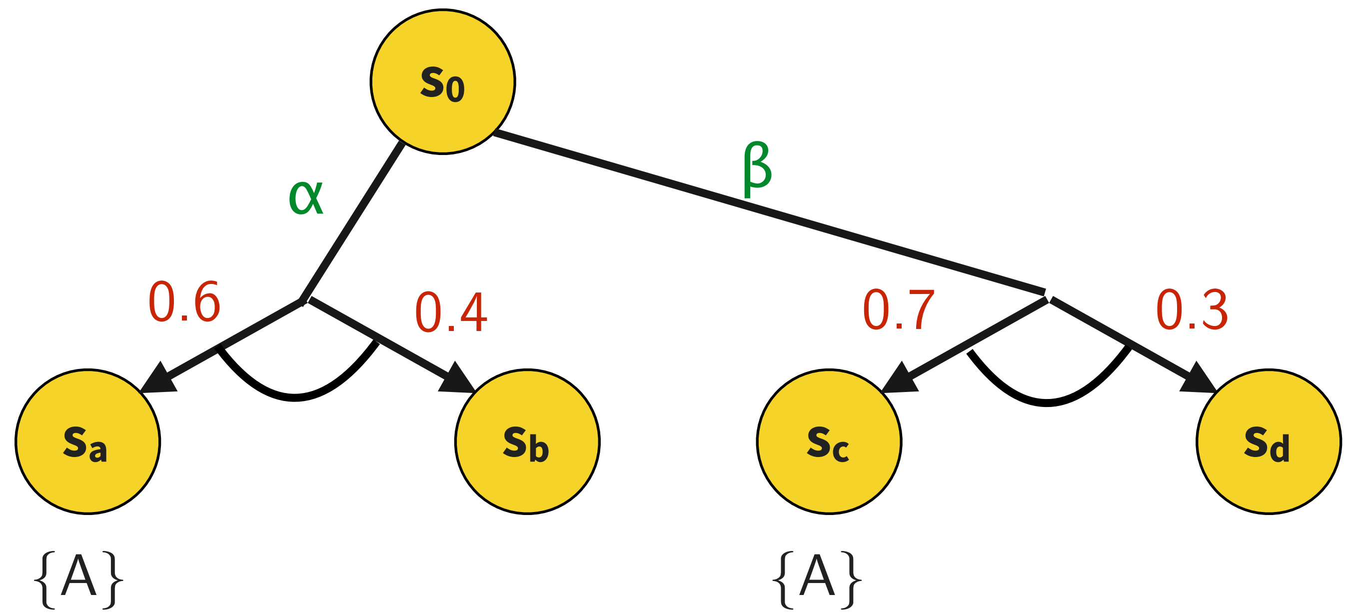
Non-linear program in general - we use dual-space LPs instead

How to **Synthesize** Policy π for Satisfying a PLTL Formula

Find policy π =

s_0: $[\alpha \to \mathbf{0.6}\ \ \beta \to \mathbf{0.4}\ ]$

such that $s_0 \models \mathbf{P}_{>0.6}\ \mathbf{F}\ A$



Proof

$s_0 \models \mathbf{P}_{>0.6}\ \mathbf{F}\ A$

The probability of all paths from $\mathbf{s_0}$ *satisfying* **F** A *is* > 0.6

iff

$\Pr\{p \mid p \text{ is a path from } s_0 \text{ and } p \models \mathbf{F}\ A\} > 0.6$

Contributions

Next

(1) Formula progression, or

~~(2) NBA mode~~

iff

$\Pr\{s_0 s_a,\ s_0 s_c\} > 0.6$

iff

$\pi(\alpha \mid s_0)\ 0.6 + \pi(\beta \mid s_0)\ 0.7 \quad > 0.6$

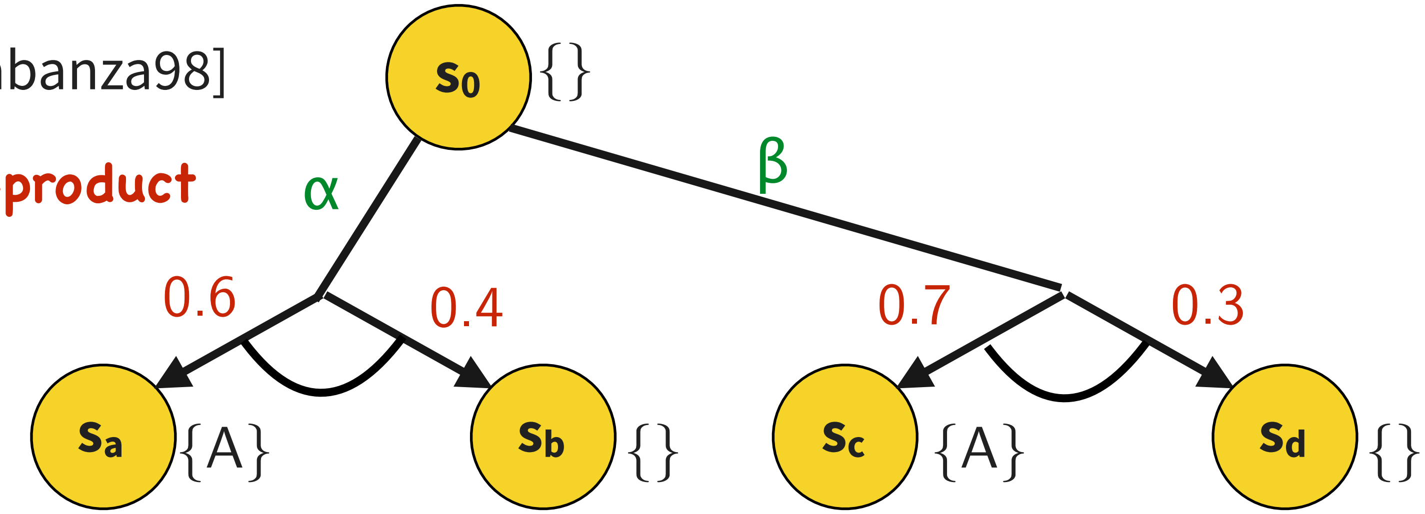
$\pi(\alpha \mid s_0) + \pi(\beta \mid s_0) = 1$

→ **Quantify over action probabilities and compute solution**

Non-linear program in general - we use dual-space LPs instead

Formula Progression [Bachus&Kabanza98]

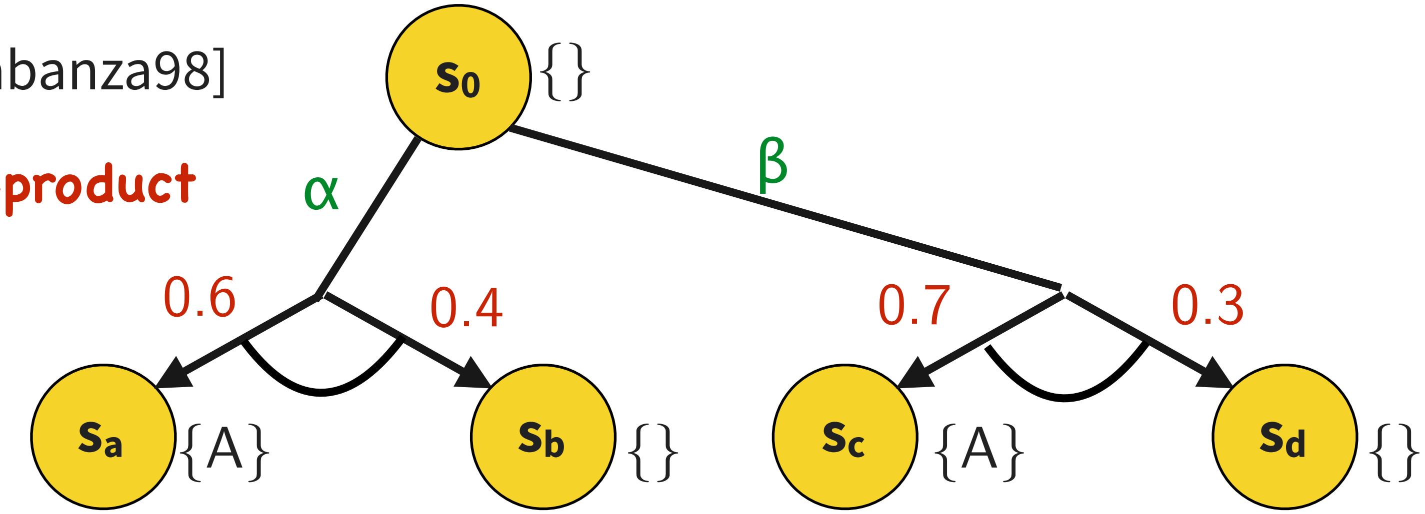
Why? On-the-fly instead of upfront cross-product



Formula Progression [Bachus&Kabanza98]

Why? On-the-fly instead of upfront cross-product

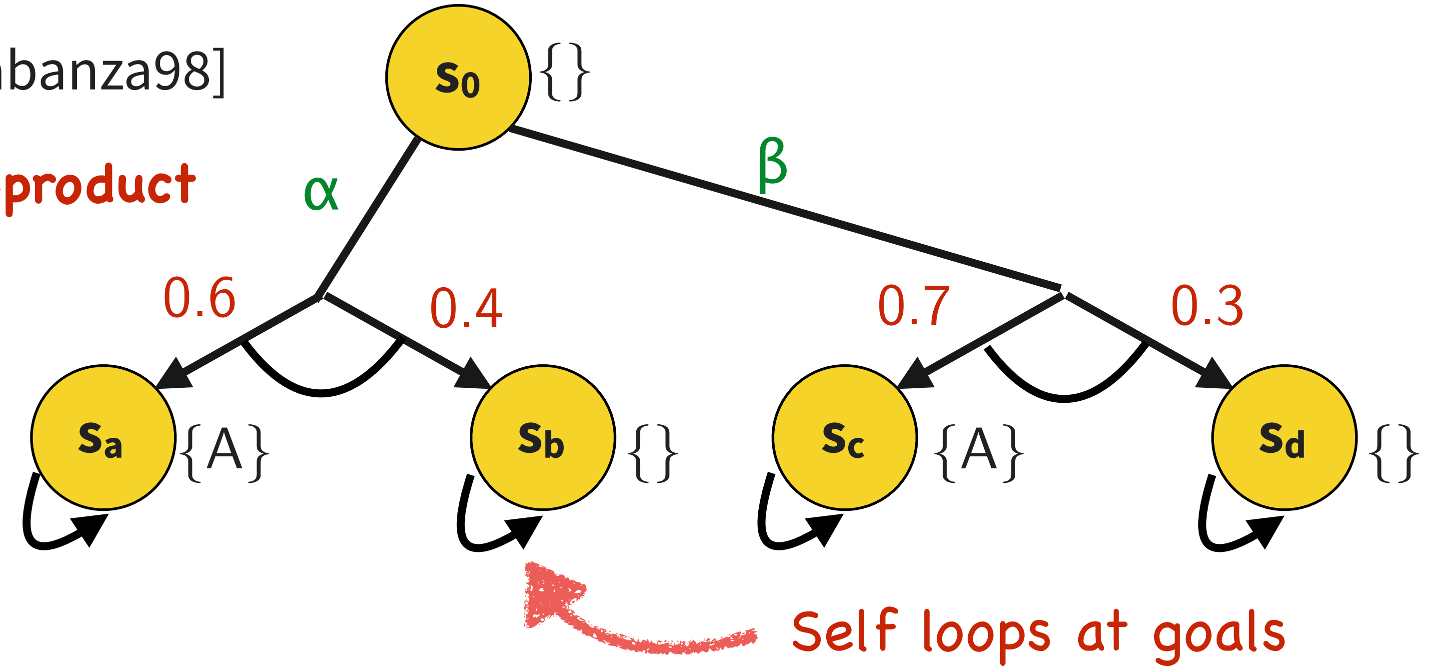
LTL is defined on *infinite* runs



Formula Progression [Bachus&Kabanza98]

Why? On-the-fly instead of upfront cross-product

LTL is defined on *infinite* runs



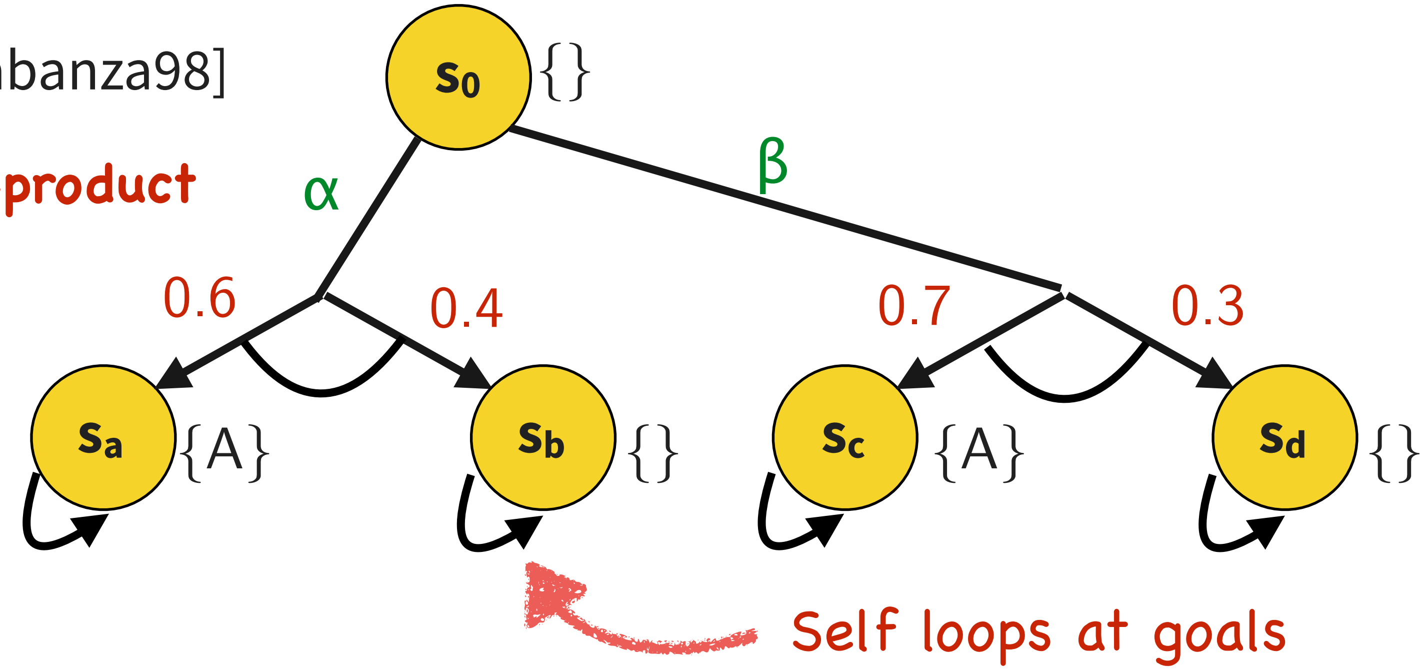
Formula Progression [Bachus&Kabanza98]

Why? On-the-fly instead of upfront cross-product

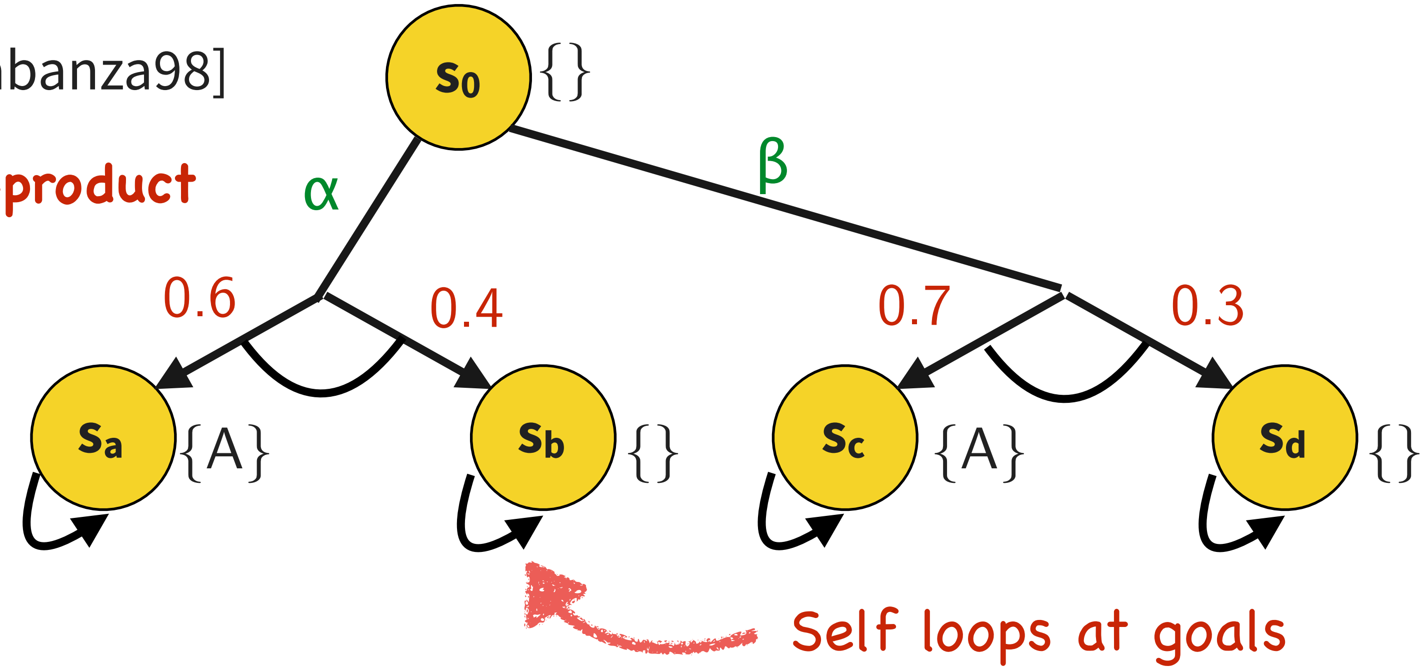
LTL is defined on *infinite* runs

LTL: $s_0\, s_a\, s_a \cdots \models \mathbf{G}\,\mathbf{X}\,A$

LTL_f: $s_0\, s_a \not\models \mathbf{G}\,\mathbf{X}\,A$



Formula Progression [Bachus&Kabanza98]



Why? On-the-fly instead of upfront cross-product

LTL is defined on *infinite* runs

LTL: $s_0\, s_a\, s_a \cdots \models \mathbf{G}\,\mathbf{X}\,A$

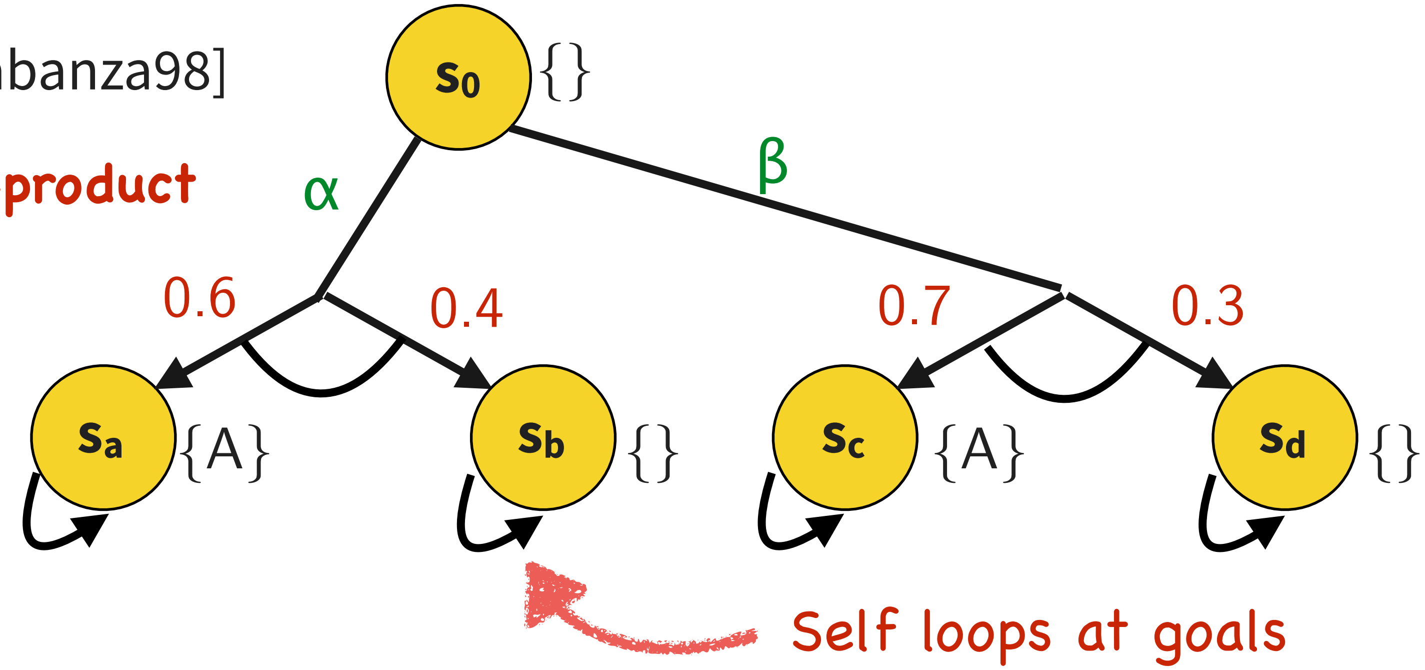
LTL$_f$: $s_0\, s_a \not\models \mathbf{G}\,\mathbf{X}\,A$

Progression: expand and simplify a given LTL formula along a path

Formula Progression [Bachus&Kabanza98]

Why? On-the-fly instead of upfront cross-product

LTL is defined on *infinite* runs



LTL: $\mathbf{s_0}\ \mathbf{s_a}\ \mathbf{s_a} \cdots \models \mathbf{G}\ \mathbf{X}\ A$

LTL$_f$: $\mathbf{s_0}\ \mathbf{s_a} \not\models \mathbf{G}\ \mathbf{X}\ A$

Progression: expand and simplify a given LTL formula along a path

$\mathbf{s_0}\ \mathbf{s_a}\ \mathbf{s_a} \cdots \models \mathbf{F}\ A$

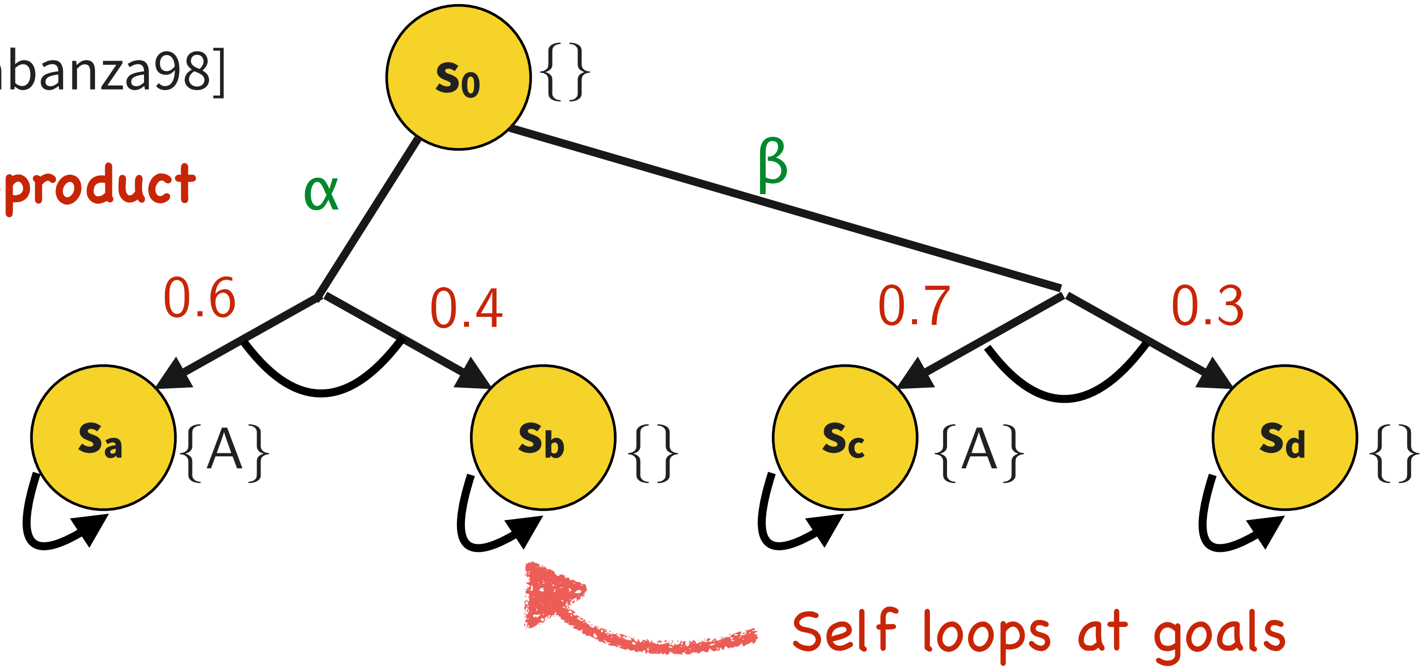
Formula Progression [Bachus&Kabanza98]

Why? On-the-fly instead of upfront cross-product

LTL is defined on *infinite* runs

LTL: $s_0\ s_a\ s_a\ \cdots \models \mathbf{G}\,\mathbf{X}\,A$

LTL$_f$: $s_0\ s_a \quad \cancel{\models}\ \mathbf{G}\,\mathbf{X}\,A$



Progression: expand and simplify a given LTL formula along a path

$s_0\ s_a\ s_a \cdots \models \mathbf{F}\,A$ ✓

$s_0\ s_a\ s_a \cdots \models A \vee \mathbf{X}\,\mathbf{F}\,A$ (by expand)

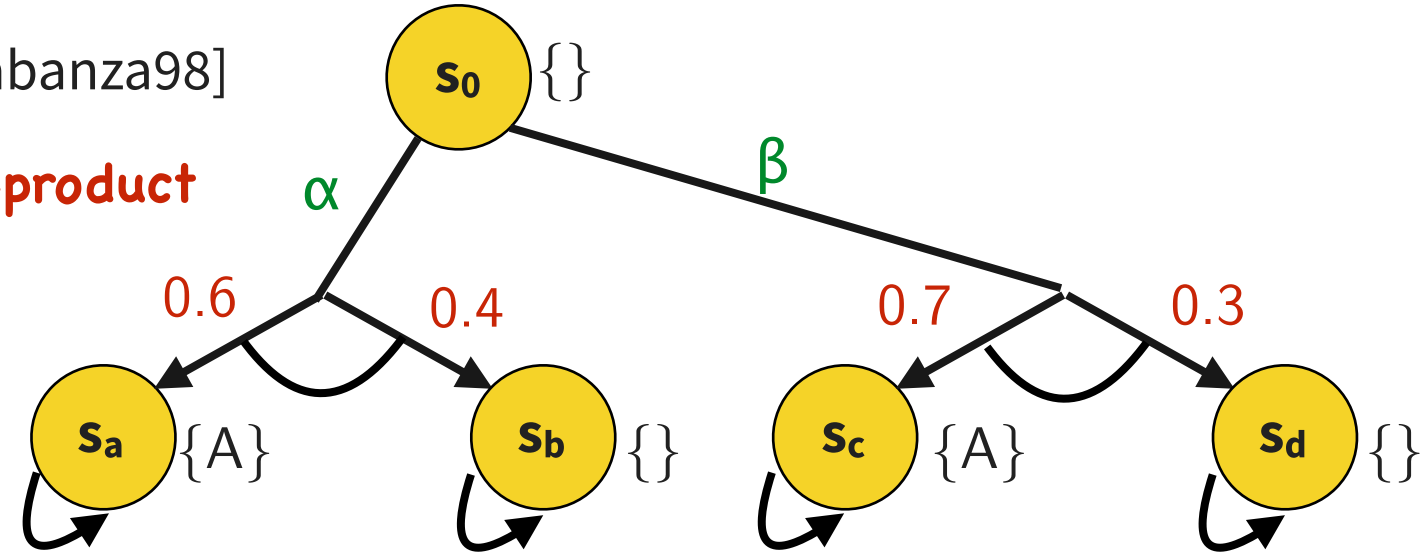
Formula Progression [Bachus&Kabanza98]

Why? On-the-fly instead of upfront cross-product

LTL is defined on *infinite* runs

LTL: $s_0\ s_a\ s_a\ \cdots \models \mathbf{G}\ \mathbf{X}\ A$

LTL_f: $s_0\ s_a \not\models \mathbf{G}\ \mathbf{X}\ A$



Self loops at goals

Progression: expand and simplify a given LTL formula along a path

$s_0\ s_a\ s_a\ \cdots \models \mathbf{F}\ A$

$s_0\ s_a\ s_a\ \cdots \models A \vee \mathbf{X}\ \mathbf{F}\ A$ (by expand)

$s_0\ s_a\ s_a\ \cdots \models \mathbf{X}\ \mathbf{F}\ A$ (by simplify)

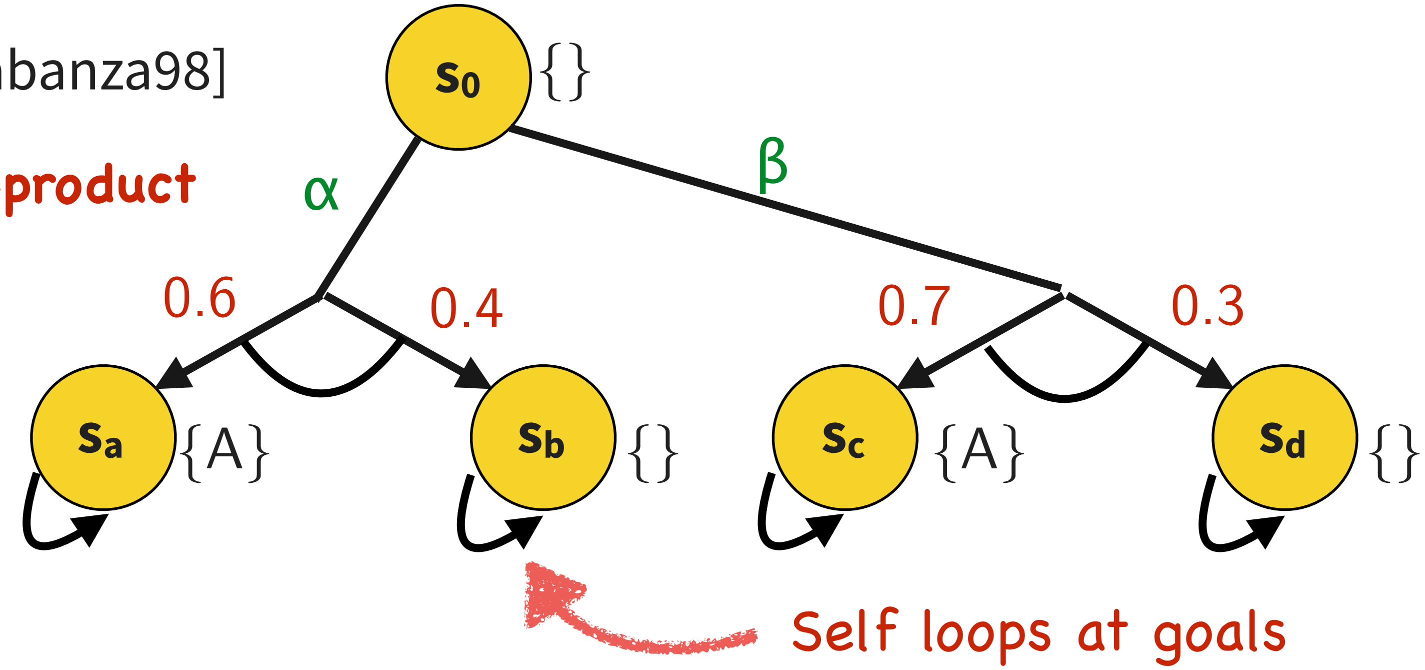
Formula Progression [Bachus&Kabanza98]

Why? On-the-fly instead of upfront cross-product

LTL is defined on *infinite* runs

LTL: $s_0\, s_a\, s_a \cdots \models \mathbf{G}\,\mathbf{X}\, A$

LTL$_f$: $s_0\, s_a \quad \not\models \mathbf{G}\,\mathbf{X}\, A$



Progression: expand and simplify a given LTL formula along a path

$s_0\, s_a\, s_a \cdots \models \mathbf{F}\, A$ ✓

$s_0\, s_a\, s_a \cdots \models A \vee \mathbf{X}\,\mathbf{F}\, A$ (by expand)

$s_0\, s_a\, s_a \cdots \models \mathbf{X}\,\mathbf{F}\, A$ (by simplify)

$s_a\, s_a \cdots \models \mathbf{F}\, A$ (by $\mathbf{X}$)

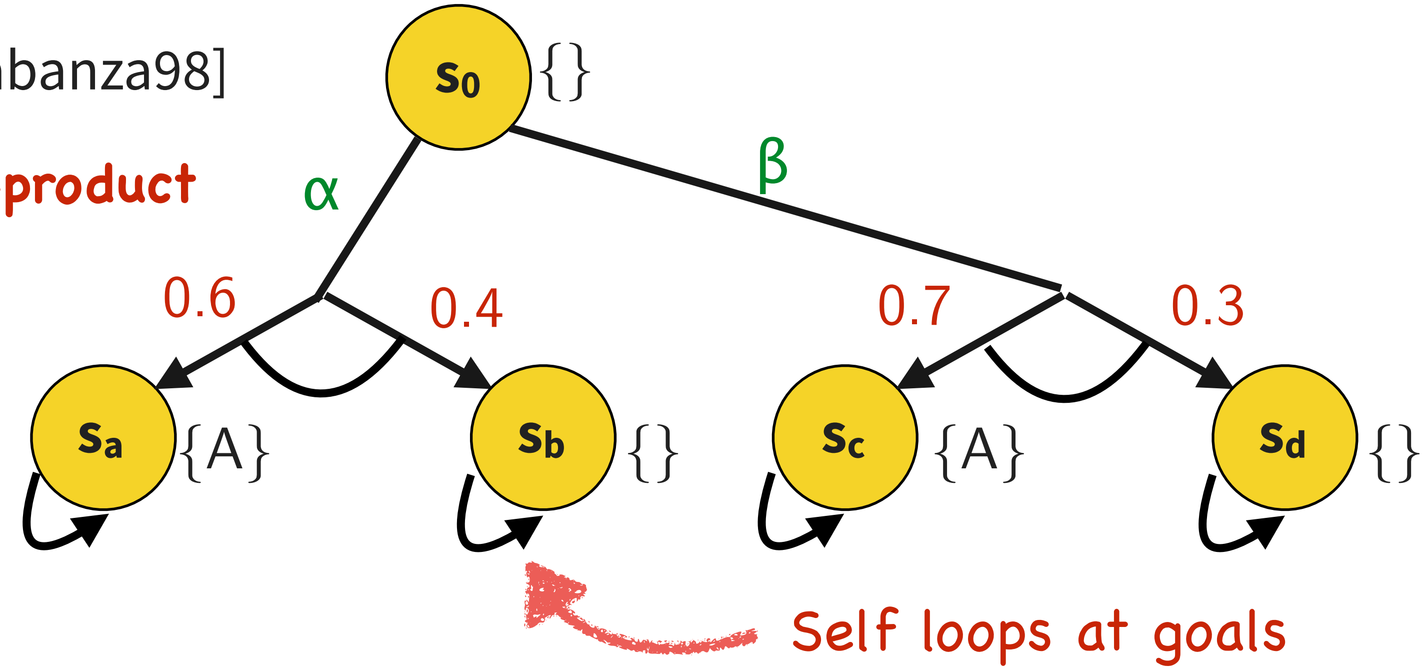
Formula Progression [Bachus&Kabanza98]

Why? On-the-fly instead of upfront cross-product

LTL is defined on *infinite* runs

LTL: $s_0\ s_a\ s_a\ \cdots \models \mathbf{G}\,\mathbf{X}\,A$

LTL$_f$: $s_0\ s_a \not\models \mathbf{G}\,\mathbf{X}\,A$



Progression: expand and simplify a given LTL formula along a path

$s_0\ s_a\ s_a\ \cdots \models \mathbf{F}\,A$ ✓

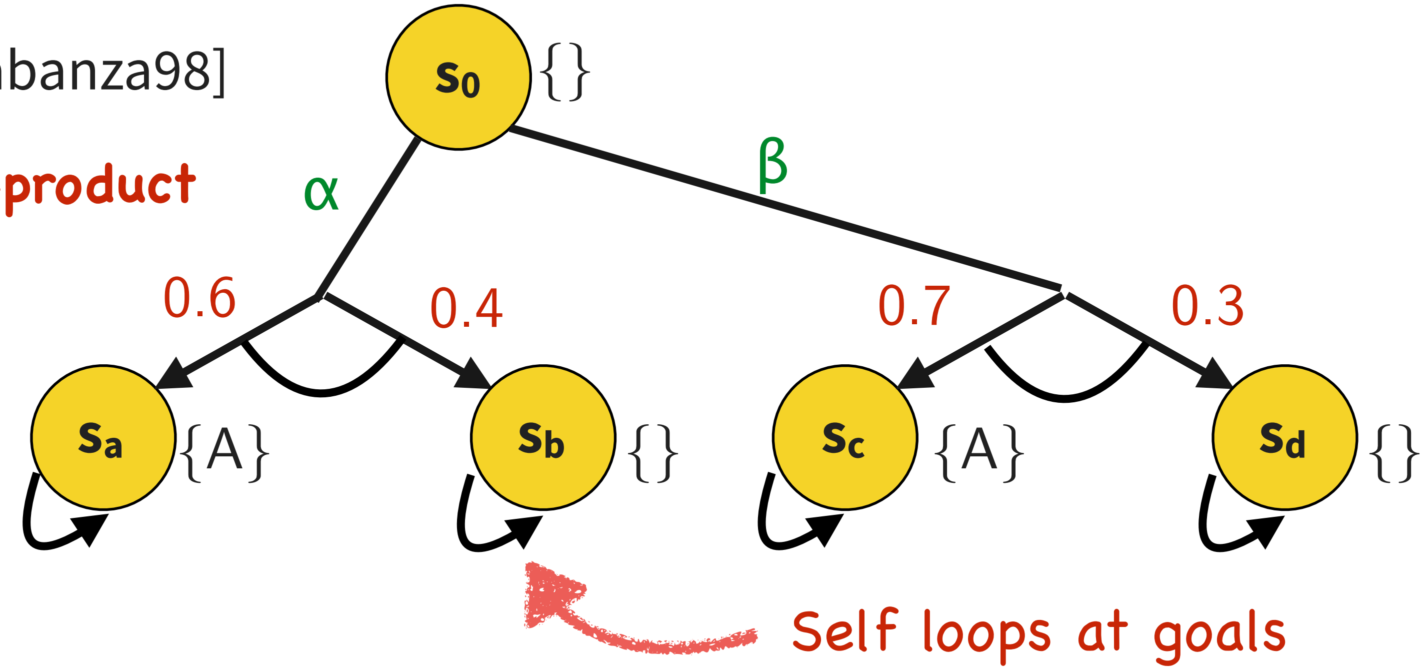
$s_0\ s_a\ s_a\ \cdots \models A \vee \mathbf{X}\,\mathbf{F}\,A$ (by expand)

$s_0\ s_a\ s_a\ \cdots \models \mathbf{X}\,\mathbf{F}\,A$ (by simplify)

$s_a\ s_a\ \cdots \models \mathbf{F}\,A$ (by **X**)

$s_a\ s_a\ \cdots \models A$ (by self-loop)

Formula Progression [Bachus&Kabanza98]



Why? On-the-fly instead of upfront cross-product

LTL is defined on *infinite* runs

LTL: $\mathbf{s_0\, s_a\, s_a} \cdots \models \mathbf{G}\,\mathbf{X}\,A$

LTL$_f$: $\mathbf{s_0\, s_a} \quad \not\models \mathbf{G}\,\mathbf{X}\,A$

Progression: expand and simplify a given LTL formula along a path

$\mathbf{s_0\, s_a\, s_a} \cdots \models \mathbf{F}\,A$ ✓

$\mathbf{s_0\, s_a\, s_a} \cdots \models A \vee \mathbf{X}\,\mathbf{F}\,A$ (by expand)

$\mathbf{s_0\, s_a\, s_a} \cdots \models \mathbf{X}\,\mathbf{F}\,A$ (by simplify)

$\mathbf{s_a\, s_a} \cdots \models \mathbf{F}\,A$ (by **X**)

$\mathbf{s_a\, s_a} \cdots \models A$ (by self-loop)

$\mathbf{s_a\, s_a} \cdots \models \top$ (by self-loop)

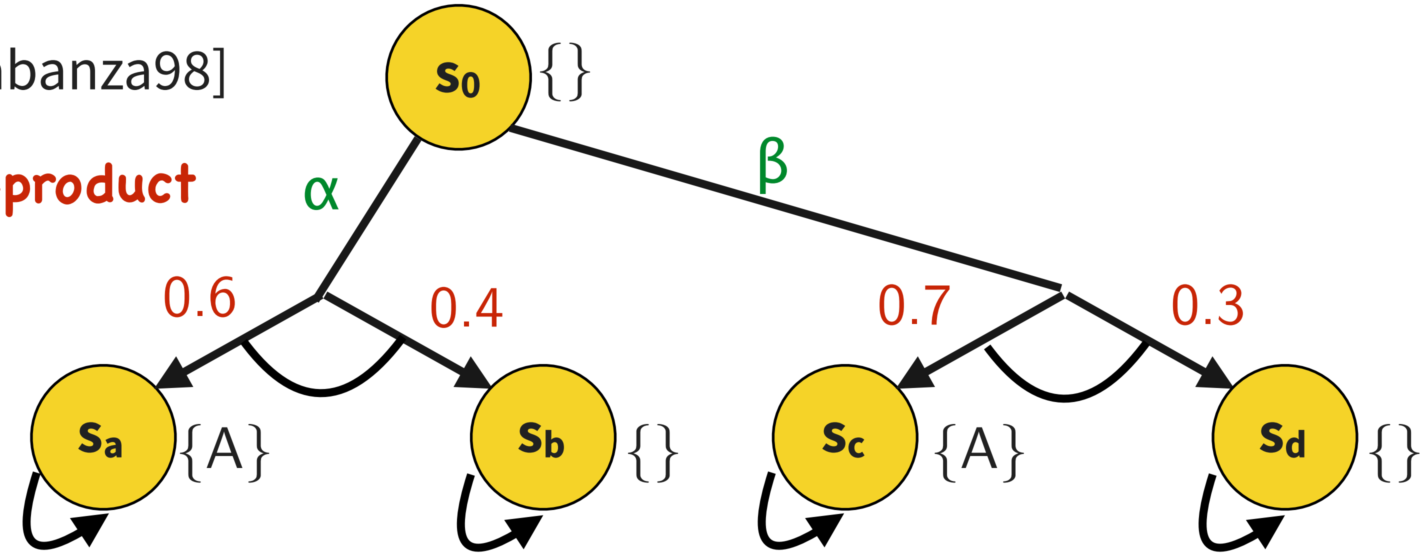
Formula Progression [Bachus&Kabanza98]

Why? On-the-fly instead of upfront cross-product

LTL is defined on *infinite* runs

LTL: $s_0\, s_a\, s_a \cdots \models \mathbf{G}\,\mathbf{X}\,A$

LTL$_f$: $s_0\, s_a \not\models \mathbf{G}\,\mathbf{X}\,A$

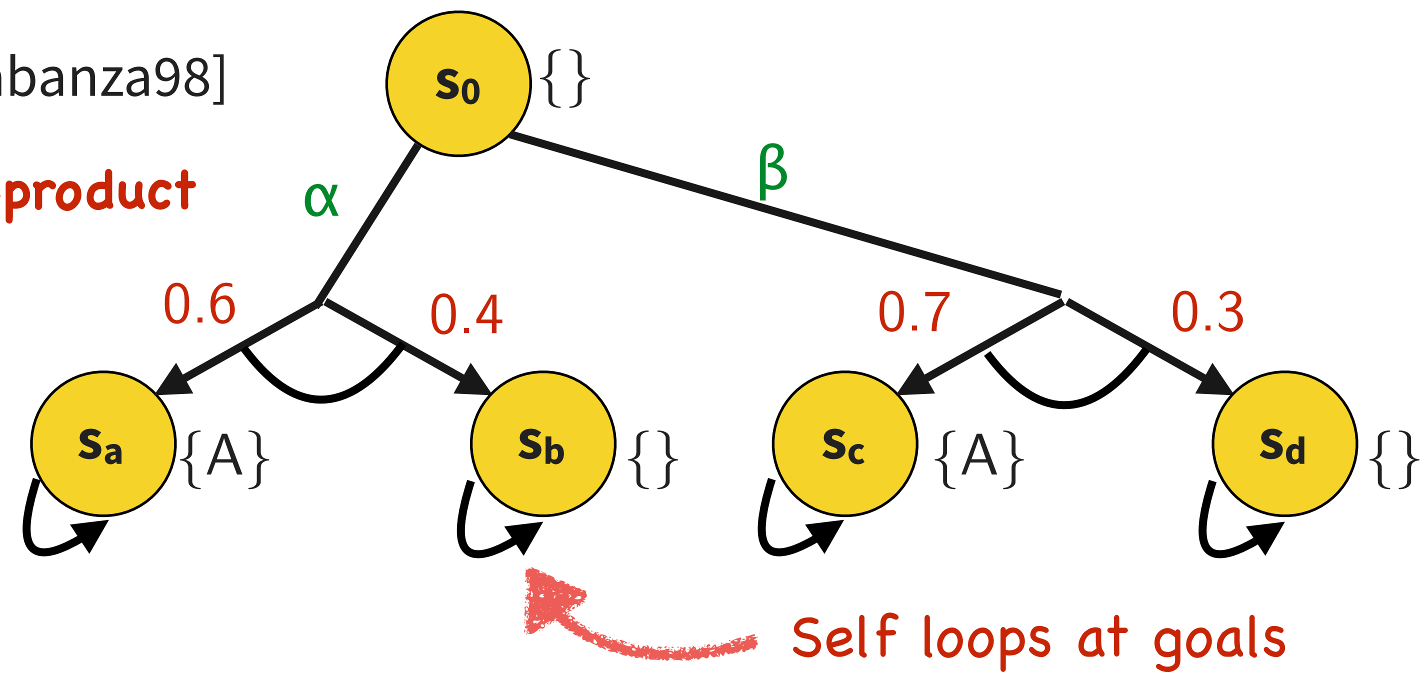


Self loops at goals

Progression: expand and simplify a given LTL formula along a path

$s_0\, s_a\, s_a \cdots \models \mathbf{F}\,A$ ✔		$s_0\, s_b\, s_b \cdots \models \mathbf{F}\,A$ ✘
$s_0\, s_a\, s_a \cdots \models A \vee \mathbf{X}\,\mathbf{F}\,A$	(by expand)	
$s_0\, s_a\, s_a \cdots \models \mathbf{X}\,\mathbf{F}\,A$	(by simplify)	
$s_a\, s_a \cdots \models \mathbf{F}\,A$	(by **X**)	
$s_a\, s_a \cdots \models A$	(by self-loop)	
$s_a\, s_a \cdots \models \top$	(by self-loop)	

Formula Progression [Bachus&Kabanza98]



Why? On-the-fly instead of upfront cross-product

LTL is defined on *infinite* runs

LTL: $s_0\ s_a\ s_a\ \cdots \models \mathbf{G}\,\mathbf{X}\,A$

LTL$_f$: $s_0\ s_a \quad \not\models \mathbf{G}\,\mathbf{X}\,A$

Progression: expand and simplify a given LTL formula along a path

$s_0\ s_a\ s_a \cdots \models \mathbf{F}\,A$	
$s_0\ s_a\ s_a \cdots \models A \vee \mathbf{X}\,\mathbf{F}\,A$	(by expand)
$s_0\ s_a\ s_a \cdots \models \mathbf{X}\,\mathbf{F}\,A$	(by simplify)
$s_a\ s_a \cdots \models \mathbf{F}\,A$	(by $\mathbf{X}$)
$s_a\ s_a \cdots \models A$	(by self-loop)
$s_a\ s_a \cdots \models \top$	(by self-loop)

$s_0\ s_b\ s_b \cdots \models \mathbf{F}\,A$	
$s_0\ s_b\ s_b \cdots \models A \vee \mathbf{X}\,\mathbf{F}\,A$	(by expand)

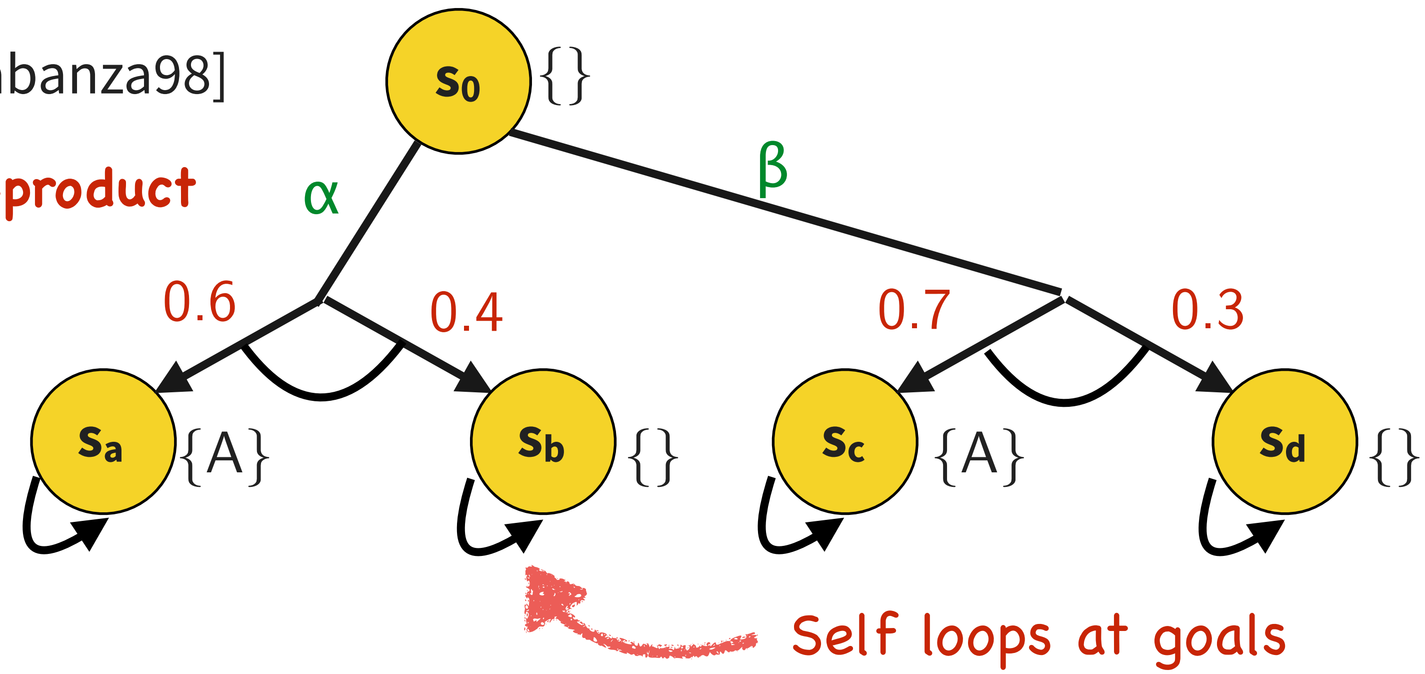
Formula Progression [Bachus&Kabanza98]

Why? On-the-fly instead of upfront cross-product

LTL is defined on *infinite* runs

LTL: $s_0\, s_a\, s_a \cdots \models \mathbf{G}\,\mathbf{X}\,A$

LTL$_f$: $s_0\, s_a \not\models \mathbf{G}\,\mathbf{X}\,A$



Progression: expand and simplify a given LTL formula along a path

$s_0\, s_a\, s_a \cdots \models \mathbf{F}\,A$

$s_0\, s_a\, s_a \cdots \models A \vee \mathbf{X}\,\mathbf{F}\,A$ (by expand)

$s_0\, s_a\, s_a \cdots \models \mathbf{X}\,\mathbf{F}\,A$ (by simplify)

$s_a\, s_a \cdots \models \mathbf{F}\,A$ (by **X**)

$s_a\, s_a \cdots \models A$ (by self-loop)

$s_a\, s_a \cdots \models \top$ (by self-loop)

$s_0\, s_b\, s_b \cdots \models \mathbf{F}\,A$

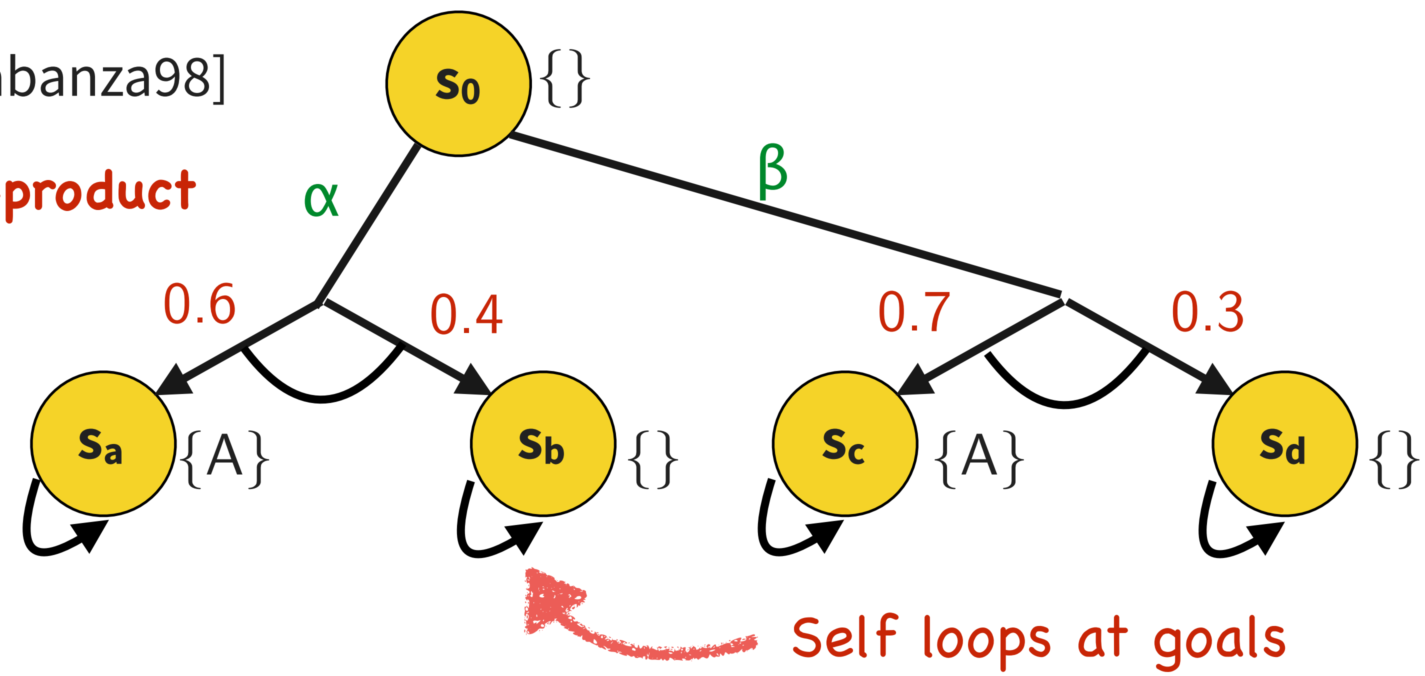
$s_0\, s_b\, s_b \cdots \models A \vee \mathbf{X}\,\mathbf{F}\,A$ (by expand)

$s_0\, s_b\, s_b \cdots \models \mathbf{X}\,\mathbf{F}\,A$ (by simplify)

Formula Progression [Bachus&Kabanza98]

Why? On-the-fly instead of upfront cross-product

LTL is defined on *infinite* runs



LTL: $s_0\, s_a\, s_a \cdots \models \mathbf{G}\,\mathbf{X}\,A$

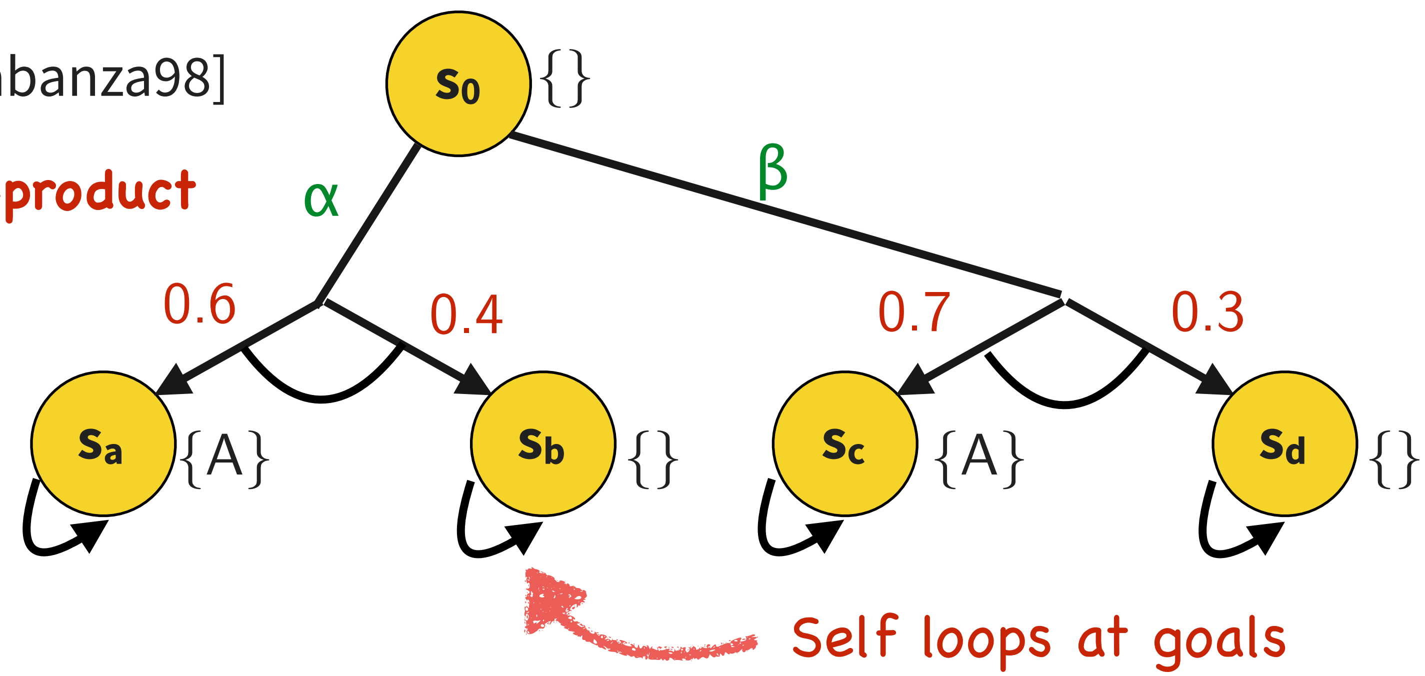
LTL$_f$: $s_0\, s_a \quad \not\models \mathbf{G}\,\mathbf{X}\,A$

Progression: expand and simplify a given LTL formula along a path

$s_0\, s_a\, s_a \cdots \models \mathbf{F}\,A$	
$s_0\, s_a\, s_a \cdots \models A \vee \mathbf{X}\,\mathbf{F}\,A$	(by expand)
$s_0\, s_a\, s_a \cdots \models \mathbf{X}\,\mathbf{F}\,A$	(by simplify)
$s_a\, s_a \cdots \models \mathbf{F}\,A$	(by **X**)
$s_a\, s_a \cdots \models A$	(by self-loop)
$s_a\, s_a \cdots \models \top$	(by self-loop)

$s_0\, s_b\, s_b \cdots \models \mathbf{F}\,A$	
$s_0\, s_b\, s_b \cdots \models A \vee \mathbf{X}\,\mathbf{F}\,A$	(by expand)
$s_0\, s_b\, s_b \cdots \models \mathbf{X}\,\mathbf{F}\,A$	(by simplify)
$s_b\, s_b \cdots \models \mathbf{F}\,A$	(by **X**)

Formula Progression [Bachus&Kabanza98]



Why? On-the-fly instead of upfront cross-product

LTL is defined on *infinite* runs

LTL: $s_0\, s_a\, s_a \cdots \models \mathbf{G}\,\mathbf{X}\,A$

LTL$_f$: $s_0\, s_a \not\models \mathbf{G}\,\mathbf{X}\,A$

Progression: expand and simplify a given LTL formula along a path

$s_0\, s_a\, s_a \cdots \models \mathbf{F}\,A$		$s_0\, s_b\, s_b \cdots \models \mathbf{F}\,A$	
$s_0\, s_a\, s_a \cdots \models A \vee \mathbf{X}\,\mathbf{F}\,A$	(by expand)	$s_0\, s_b\, s_b \cdots \models A \vee \mathbf{X}\,\mathbf{F}\,A$	(by expand)
$s_0\, s_a\, s_a \cdots \models \mathbf{X}\,\mathbf{F}\,A$	(by simplify)	$s_0\, s_b\, s_b \cdots \models \mathbf{X}\,\mathbf{F}\,A$	(by simplify)
$s_a\, s_a \cdots \models \mathbf{F}\,A$	(by **X**)	$s_b\, s_b \cdots \models \mathbf{F}\,A$	(by **X**)
$s_a\, s_a \cdots \models A$	(by self-loop)	$s_b\, s_b \cdots \models A$	(by self-loop)
$s_a\, s_a \cdots \models \top$	(by self-loop)		

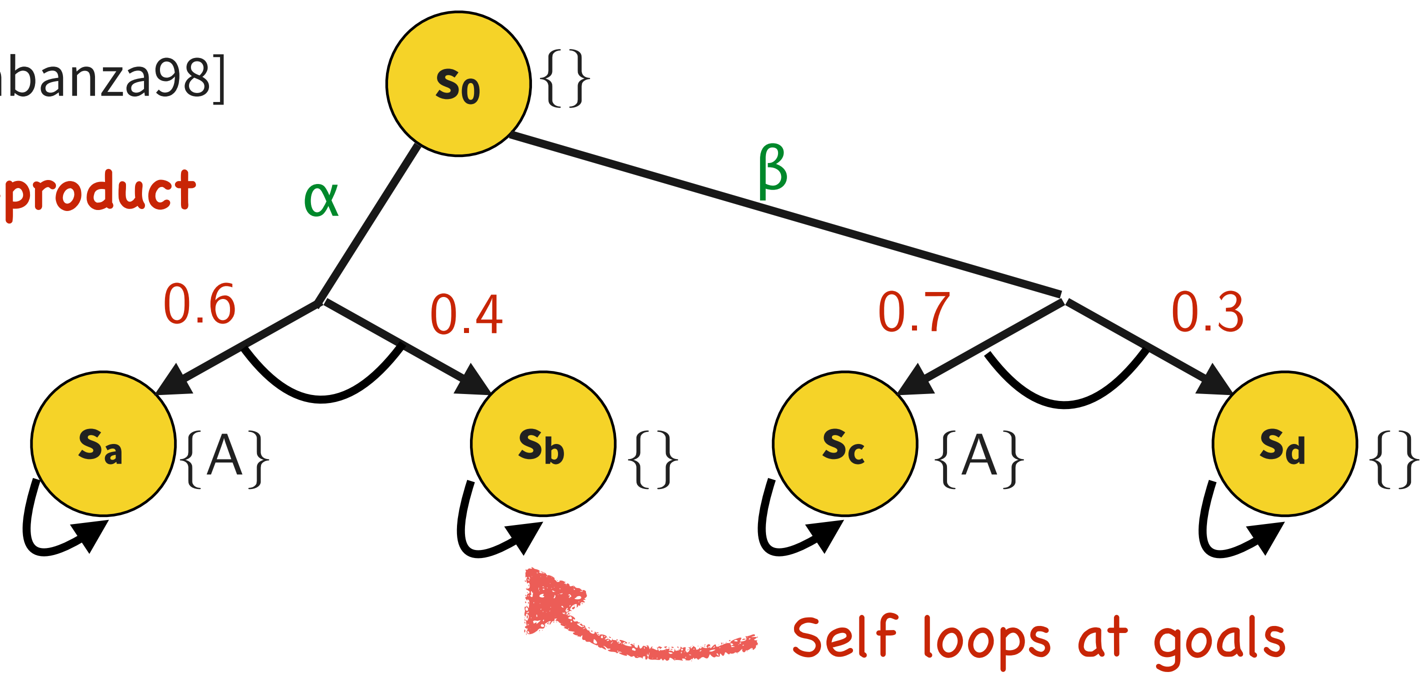
Formula Progression [Bachus&Kabanza98]

Why? On-the-fly instead of upfront cross-product

LTL is defined on *infinite* runs

LTL: $\mathbf{s_0\ s_a\ s_a} \cdots \models \mathbf{G\ X}\ A$

LTL_f: $\mathbf{s_0\ s_a}$ $\not\models \mathbf{G\ X}\ A$



Progression: expand and simplify a given LTL formula along a path

$\mathbf{s_0\ s_a\ s_a} \cdots \models \mathbf{F}\ A$ ✓		$\mathbf{s_0\ s_b\ s_b} \cdots \models \mathbf{F}\ A$ ✗	
$\mathbf{s_0\ s_a\ s_a} \cdots \models A \vee \mathbf{X\ F}\ A$	(by expand)	$\mathbf{s_0\ s_b\ s_b} \cdots \models A \vee \mathbf{X\ F}\ A$	(by expand)
$\mathbf{s_0\ s_a\ s_a} \cdots \models \mathbf{X\ F}\ A$	(by simplify)	$\mathbf{s_0\ s_b\ s_b} \cdots \models \mathbf{X\ F}\ A$	(by simplify)
$\mathbf{s_a\ s_a} \cdots \models \mathbf{F}\ A$	(by **X**)	$\mathbf{s_b\ s_b} \cdots \models \mathbf{F}\ A$	(by **X**)
$\mathbf{s_a\ s_a} \cdots \models A$	(by self-loop)	$\mathbf{s_b\ s_b} \cdots \models A$	(by self-loop)
$\mathbf{s_a\ s_a} \cdots \models \top$	(by self-loop)	$\mathbf{s_b\ s_b} \cdots \models \bot$	(by self-loop)

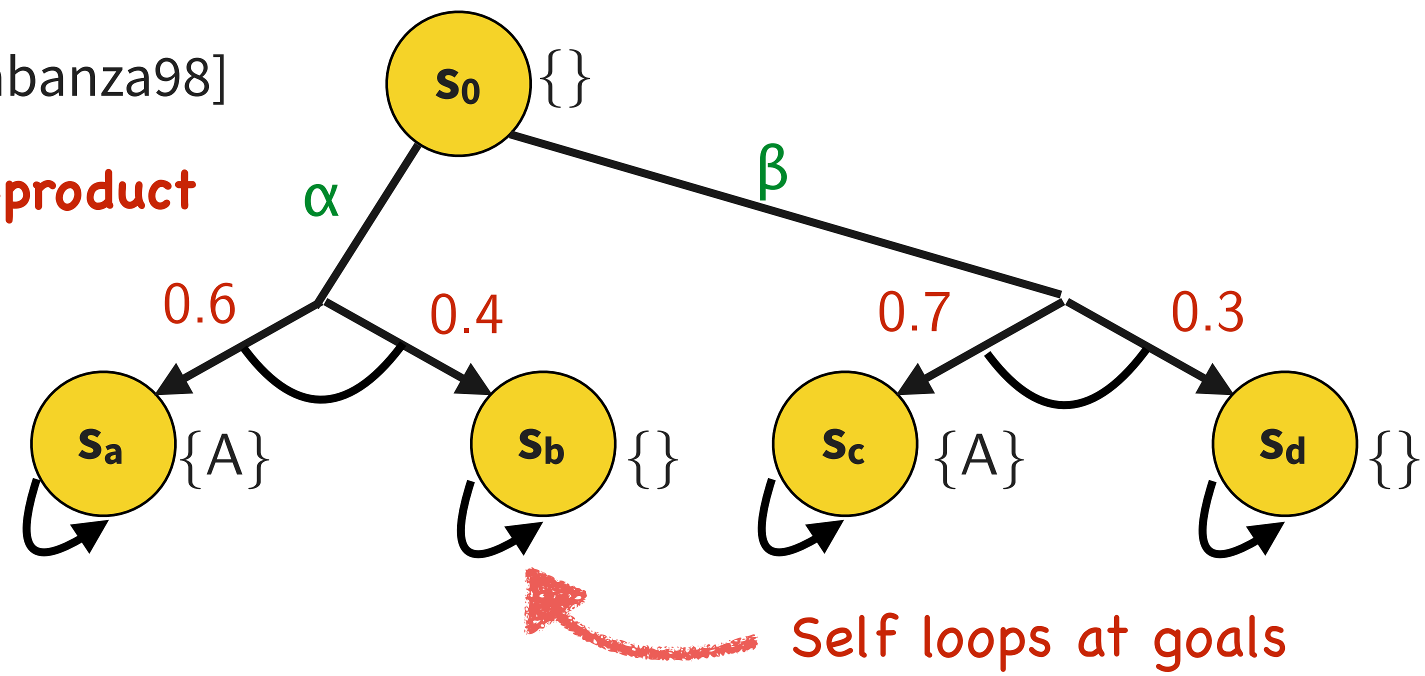
Formula Progression [Bachus&Kabanza98]

Why? On-the-fly instead of upfront cross-product

LTL is defined on *infinite* runs

LTL: $s_0\, s_a\, s_a \cdots \models \mathbf{G}\,\mathbf{X}\,A$

LTL$_f$: $s_0\, s_a \not\models \mathbf{G}\,\mathbf{X}\,A$



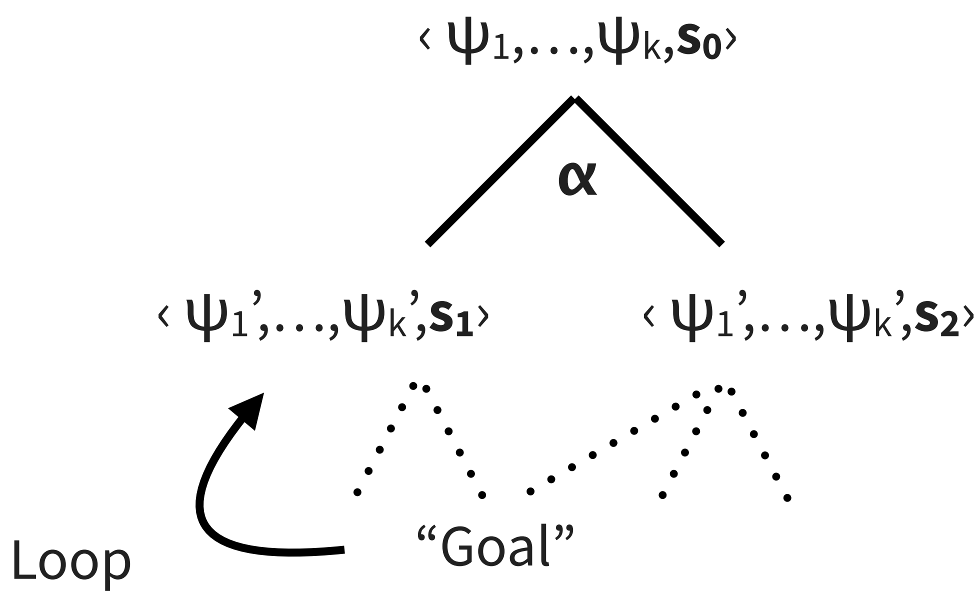
Progression: expand and simplify a given LTL formula along a path

$s_0\, s_a\, s_a \cdots \models \mathbf{F}\,A$ ✓		$s_0\, s_b\, s_b \cdots \models \mathbf{F}\,A$ ✗	
$s_0\, s_a\, s_a \cdots \models A \lor \mathbf{X}\,\mathbf{F}\,A$	(by expand)	$s_0\, s_b\, s_b \cdots \models A \lor \mathbf{X}\,\mathbf{F}\,A$	(by expand)
$s_0\, s_a\, s_a \cdots \models \mathbf{X}\,\mathbf{F}\,A$	(by simplify)	$s_0\, s_b\, s_b \cdots \models \mathbf{X}\,\mathbf{F}\,A$	(by simplify)
$s_a\, s_a \cdots \models \mathbf{F}\,A$	(by **X**)	$s_b\, s_b \cdots \models \mathbf{F}\,A$	(by **X**)
$s_a\, s_a \cdots \models A$	(by self-loop)	$s_b\, s_b \cdots \models A$	(by self-loop)
$s_a\, s_a \cdots \models \top$	(by self-loop)	$s_b\, s_b \cdots \models \bot$	(by self-loop)

All transitions "if and only if"

Multi-Objective Progression in the State Space

A = DRA(ψ_1) × ⋯ × DRA(ψ_k) × DRA(**F** Goal) × **S**



‘ is the progression operator

Questions/Issues

- **Q**: Does *repeated* progression terminate?

  A: It better does, but some rules even increases formula size: **F** A → A ∨ **X** **F** A

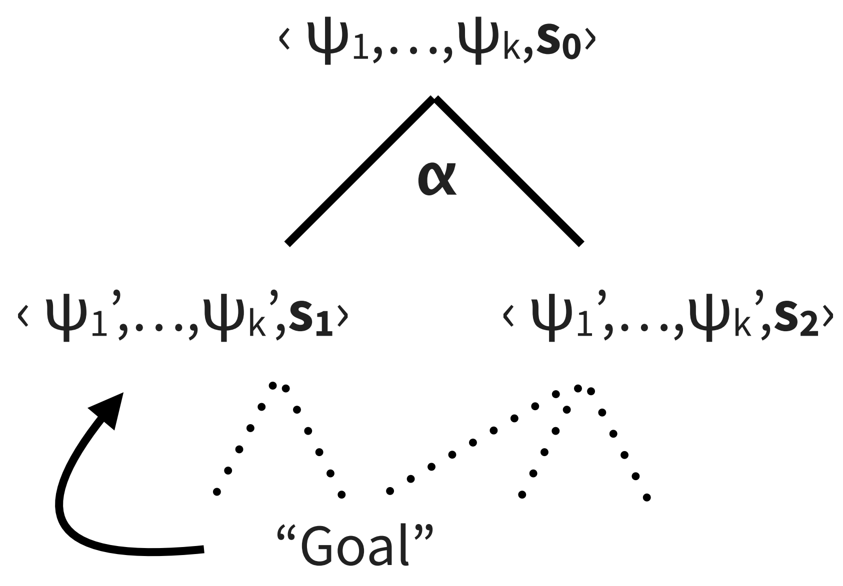
- **Q**: How to detect a loop ‹**ψ**,**s**› ≡ ‹**ψ’’…’**,**s**› ?

  A: Check equivalence of LTL formulas. Exponential!

  A: Check equality of *canonical representation* of LTL formulas. Polynomial!

Multi-Objective Progression in the State Space

$\mathbf{A} = \mathrm{DRA}(\psi_1) \times \cdots \times \mathrm{DRA}(\psi_k) \times \mathrm{DRA}(\mathbf{F}\ \mathrm{Goal}) \times \mathbf{S}$



' is the progression operator

Questions/Issues

- **Q**: Does *repeated* progression terminate?

  A: It better does, but some rules even increases formula size: $\mathbf{F}\,A \rightarrow A \vee \mathbf{X}\,\mathbf{F}\,A$

- **Q**: How to detect a loop $\langle \boldsymbol{\psi}, \mathbf{s} \rangle \equiv \langle \boldsymbol{\psi}''^{\cdots}{}', \mathbf{s} \rangle$?

  A: Check equivalence of LTL formulas. Exponential!

  A: Check equality of *canonical representation* of LTL formulas. Polynomial!

Tseitin-style progression

Tseitin Transformation for Classical Logic

- Earliest *polynomial* conjunctive normal form (CNF) transformation [Tseitin 1966]
- Improved versions popular with first-order theorem proving [Azmy&Weidenbach 2013]

How it works

- Introduce *names* for complex subformulas before multiplying-out

$(A \land B) \lor \psi \quad \rightarrow \quad (A \lor \psi) \land (B \lor \psi)$ Duplicates ψ

$(A \land B) \lor \psi \quad \rightarrow \quad \psi_{(A \land B)} \lor \psi$ $\psi_{(A \land B)}$ is a name for $(A \land B)$

$\neg\psi_{(A \land B)} \lor A$

$\neg\psi_{(A \land B)} \lor B$

Definition of $\psi_{(A \land B)}$

- Requires polynomially many names, one for each subformula
- Apply once-and-forall to given formula and obtain equi-satisfiable CNF
- That CNF is a conjunction of disjunction of 3-literal clauses

Tseitin Transformation for Classical Logic

- Earliest *polynomial* conjunctive normal form (CNF) transformation [Tseitin 1966]
- Improved versions popular with first-order theorem proving [Azmy&Weidenbach 2013]

How it works

- Introduce *names* for complex subformulas before multiplying-out

✗ $(A \wedge B) \vee \psi \quad \rightarrow \quad (A \vee \psi) \wedge (B \vee \psi)$ — Duplicates ψ

✓ $(A \wedge B) \vee \psi \quad \rightarrow \quad \psi_{(A \wedge B)} \vee \psi$ — $\psi_{(A \wedge B)}$ is a name for $(A \wedge B)$

$\neg\psi_{(A \wedge B)} \vee A$

$\neg\psi_{(A \wedge B)} \vee B$ — Definition of $\psi_{(A \wedge B)}$

- Requires polynomially many names, one for each subformula
- Apply once-and-forall to given formula and obtain equi-satisfiable CNF
- That CNF is a conjunction of disjunction of 3-literal clauses

→ **We need to apply Tseitin CNF to every derived formula: Tseitin-style *progression***

Tseitin-Style Progression

First (?) application to LTL progression

All LTL formulas are now in 3-CNF

$$\{ \ \ldots \ \ \{ L^i_1, L^i_2, L^i_3 \} \ \ \ldots \ \}$$

Tseitin-Style Progression

First (?) application to LTL progression

All LTL formulas are now in 3-CNF

$$\{ \;\ldots\; \{ L^i_1, L^i_2, L^i_3 \} \;\ldots\; \}$$

3-CNF:
$\wedge$-connected set of 3-literal clauses

Tseitin-Style Progression

First (?) application to LTL progression

All LTL formulas are now in 3-CNF

$$\{ \ \dots \ \{ \mathbf{L}^i_1, \mathbf{L}^i_2, \mathbf{L}^i_3 \} \ \dots \ \}$$

3-CNF:
$\wedge$-connected set of 3-literal clauses

$\mathbf{L}^i_k \in \mathrm{sub}(\psi) \cup \{ \neg\phi, \mathbf{X}\,\phi, \mathbf{X}\,\neg\phi \mid \phi \in \mathrm{sub}(\psi) \} \cup$ “Names” $\cup \dots$

where ψ = initially given formula

Tseitin-Style Progression

First (?) application to LTL progression

All LTL formulas are now in 3-CNF

$\{ \;\dots\; \{ L^i_1, L^i_2, L^i_3 \} \;\dots\; \}$

3-CNF:
$\wedge$-connected set of 3-literal clauses

$L^i_k \in \mathrm{sub}(\psi) \cup \{ \neg\phi, \mathbf{X}\,\phi, \mathbf{X}\,\neg\phi \mid \phi \in \mathrm{sub}(\psi) \} \cup$ "Names" $\cup \dots$

Progression

- Sequence $\mathbf{s_0} \vDash \{\{ \psi \}\} \;\rightarrow\; \mathbf{s_1} \vDash \mathbf{\Gamma_1} \;\rightarrow\; \mathbf{s_2} \vDash \mathbf{\Gamma_2} \;\rightarrow\; \dots \rightarrow\; \mathbf{s_i} \vDash \mathbf{\Gamma_i}$ where ψ = initially given formula
- Initially $\mathbf{s_0} \vDash \mathbf{\Gamma_0}$ where $\mathbf{\Gamma_0}$ = simplified 3-CNF of $\{\{ \psi \}\}$
- Step $\mathbf{s_i} \vDash \mathbf{\Gamma_i} \;\rightarrow\; \mathbf{s_{i+1}} \vDash \mathbf{\Gamma_{i+1}}$:

  (1) Eliminate names from $\mathbf{\Gamma_i}$ and strip **X**-operators

  (2) $\mathbf{\Gamma_{i+1}}$ = simplified 3-CNF of (1)
- Stop if $\mathbf{s_k} \vDash \mathbf{\Gamma_k} \;=\; \mathbf{s_i} \vDash \mathbf{\Gamma_i}$ for some $\mathbf{k} < \mathbf{i}$

  Replaces $\equiv$-test for LTL-formulas by *polynomial* set equality test!

Complexity

Literal signature $|\Sigma| \in O(|\psi|^2)$

$O(|\Sigma|^3) \;=\; O(|\psi|^6)$ different clauses

$2^{O(|\psi|^6)}$ different clause sets

Theorem

Space and time complexity polynomial in $|\mathbf{S}|$ and *single* exponential $|\psi|$

Tseitin-Style Progre

All LTL formulas are now in

{ … { L^i_1, L^i_2, L^i_3 }

Progression

- Sequence $\mathbf{s_0} \vDash \{\{\, \psi \,\}\}$ →
- Initially $\mathbf{s_0} \vDash \boldsymbol{\Gamma_0}$ where $\boldsymbol{\Gamma_0}$ =
- Step $\mathbf{s_i} \vDash \boldsymbol{\Gamma_i}$ → $\mathbf{s_{i+1}} \vDash \boldsymbol{\Gamma_{i+1}}$

  (1) Eliminate names from

  (2) $\boldsymbol{\Gamma_{i+1}}$ = simplified 3-CNF
- Stop if $\mathbf{s_k} \vDash \boldsymbol{\Gamma_k}$ = $\mathbf{s_i} \vDash \boldsymbol{\Gamma_i}$ for

  Replaces ≡-test for LTL-for

Complexity

Literal signature $|\Sigma| \in O(|$

$O(|\Sigma|^3) = O(|\psi|^6)$ differe

$2^{O(|\psi|^6)}$ different claus

Theorem

Space and time complexity

$$\{\{\}\} \uplus \Gamma \Rightarrow_s \{\{\}\} \quad \text{if } \Gamma \neq \emptyset \qquad \text{(Triv)}$$

$$\{\{\top\} \uplus \Psi\} \uplus \Gamma \Rightarrow_s \Gamma \qquad (\top)$$

$$\{\{\neg\top\} \uplus \Psi\} \uplus \Gamma \Rightarrow_s \{\Psi\} \cup \Gamma \qquad (\neg\top)$$

$$\{\{(v,d)\} \uplus \Psi\} \uplus \Gamma \Rightarrow_s \Gamma \quad \text{if } (v,d) \in AP \text{ and } s[v] = d \qquad \text{(Eval1)}$$

$$\{\{(v,d)\} \uplus \Psi\} \uplus \Gamma \Rightarrow_s \{\Psi\} \cup \Gamma \quad \text{if } (v,d) \in AP \text{ and } s[v] \neq d \qquad \text{(Eval2)}$$

$$\{\{\neg(v,d)\} \uplus \Psi\} \uplus \Gamma \Rightarrow_s \{\Psi\} \cup \Gamma \quad \text{if } (v,d) \in AP \text{ and } s[v] = d \qquad \text{(Eval3)}$$

$$\{\{\neg(v,d)\} \uplus \Psi\} \uplus \Gamma \Rightarrow_s \Gamma \quad \text{if } (v,d) \in AP \text{ and } s[v] \neq d \qquad \text{(Eval4)}$$

$$\{\{\neg\neg\psi\} \uplus \Psi\} \uplus \Gamma \Rightarrow_s \{\{\psi\} \cup \Psi\} \cup \Gamma \qquad (\neg\neg)$$

$$\begin{aligned}\{\{\psi_1 \vee \psi_2\} \uplus \Psi\} \uplus \Gamma \Rightarrow_s {} & \{\{A_{\psi_1 \vee \psi_2}\} \cup \Psi, \\ & \{\neg A_{\psi_1 \vee \psi_2}, \psi_1, \psi_2\}\} \cup \Gamma\end{aligned} \qquad (\vee)$$

$$\begin{aligned}\{\{\neg(\psi_1 \vee \psi_2)\} \uplus \Psi\} \uplus \Gamma \Rightarrow_s {} & \{\{\neg A_{\psi_1 \vee \psi_2}\} \cup \Psi, \\ & \{A_{\psi_1 \vee \psi_2}, \overline{\psi_1}\}, \\ & \{A_{\psi_1 \vee \psi_2}, \overline{\psi_2}\}\} \cup \Gamma\end{aligned} \qquad (\neg\vee)$$

$$\begin{aligned}\{\{\psi_1 \wedge \psi_2\} \uplus \Psi\} \uplus \Gamma \Rightarrow_s {} & \{\{A_{\psi_1 \wedge \psi_2}\} \cup \Psi, \\ & \{\neg A_{\psi_1 \wedge \psi_2}, \psi_1\}, \\ & \{\neg A_{\psi_1 \wedge \psi_2}, \psi_2\}\} \cup \Gamma\end{aligned} \qquad (\wedge)$$

$$\begin{aligned}\{\{\neg(\psi_1 \wedge \psi_2)\} \uplus \Psi\} \uplus \Gamma \Rightarrow_s {} & \{\{\neg A_{\psi_1 \wedge \psi_2}\} \cup \Psi, \\ & \{A_{\psi_1 \wedge \psi_2}, \overline{\psi_1}, \overline{\psi_2}\}\} \cup \Gamma\end{aligned} \qquad (\neg\wedge)$$

$$\begin{aligned}\{\{\psi_1 \,\mathbf{U}\, \psi_2\} \uplus \Psi\} \uplus \Gamma \Rightarrow_s {} & \{\{A_{\psi_1 \mathbf{U} \psi_2}\} \cup \Psi, \\ & \{\neg A_{\psi_1 \mathbf{U} \psi_2}, \psi_2, A_{\psi_1 \wedge \mathbf{X}(\psi_1 \mathbf{U} \psi_2)}\}, \\ & \{\neg A_{\psi_1 \wedge \mathbf{X}(\psi_1 \mathbf{U} \psi_2)}, \psi_1\}, \\ & \{\neg A_{\psi_1 \wedge \mathbf{X}(\psi_1 \mathbf{U} \psi_2)}, \mathbf{X}\,(\psi_1 \,\mathbf{U}\, \psi_2)\}\} \cup \Gamma\end{aligned} \qquad (\mathbf{U})$$

$$\begin{aligned}\{\{\neg(\psi_1 \,\mathbf{U}\, \psi_2)\} \uplus \Psi\} \uplus \Gamma \Rightarrow_s {} & \{\{\neg A_{\psi_1 \mathbf{U} \psi_2}\} \cup \Psi, \\ & \{A_{\psi_1 \mathbf{U} \psi_2}, \overline{\psi_2}\}, \\ & \{A_{\psi_1 \mathbf{U} \psi_2}, \neg A_{\psi_1 \wedge \mathbf{X}(\psi_1 \mathbf{U} \psi_2)}\}, \\ & \{A_{\psi_1 \wedge \mathbf{X}(\psi_1 \mathbf{U} \psi_2)}, \overline{\Psi_1}, \mathbf{X}\,\neg(\psi_1 \,\mathbf{U}\, \psi_2)\}\} \cup \Gamma\end{aligned} \qquad (\neg\mathbf{U})$$

$$\{\{\neg \mathbf{X}\,\psi\} \uplus \Psi\} \uplus \Gamma \Rightarrow_s \{\{\mathbf{X}\,\overline{\psi}\} \cup \Psi\} \cup \Gamma \qquad (\neg\mathbf{X})$$

Tseitin-Style Progression

First (?) application to LTL progression

All LTL formulas are now in 3-CNF

$$\{\ \ldots\ \boxed{\{ L^i_1, L^i_2, L^i_3 \}}\ \ldots\ \}$$

3-CNF:
$\wedge$-connected set of 3-literal clauses

$L^i_k \in \mathrm{sub}(\psi) \cup \{ \neg\phi, \mathbf{X}\,\phi, \mathbf{X}\,\neg\phi \mid \phi \in \mathrm{sub}(\psi) \} \cup$ "Names" $\cup \ldots$

Progression

- Sequence $\mathbf{s_0} \models \{\{ \psi \}\} \rightarrow \mathbf{s_1} \models \mathbf{\Gamma_1} \rightarrow \mathbf{s_2} \models \mathbf{\Gamma_2} \rightarrow \ldots \rightarrow \mathbf{s_i} \models \mathbf{\Gamma_i}$ where ψ = initially given formula
- Initially $\mathbf{s_0} \models \mathbf{\Gamma_0}$ where $\mathbf{\Gamma_0}$ = simplified 3-CNF of $\{\{ \psi \}\}$
- Step $\mathbf{s_i} \models \mathbf{\Gamma_i} \rightarrow \mathbf{s_{i+1}} \models \mathbf{\Gamma_{i+1}}$:

  (1) Eliminate names from $\mathbf{\Gamma_i}$ and strip **X**-operators

  (2) $\mathbf{\Gamma_{i+1}}$ = simplified 3-CNF of (1)
- Stop if $\mathbf{s_k} \models \mathbf{\Gamma_k} = \mathbf{s_i} \models \mathbf{\Gamma_i}$ for some $\mathbf{k} < \mathbf{i}$

  Replaces $\equiv$-test for LTL-formulas by *polynomial* set equality test!

Complexity

Literal signature $|\Sigma| \in O(|\psi|^2)$

$O(|\Sigma|^3) = O(|\psi|^6)$ different clauses

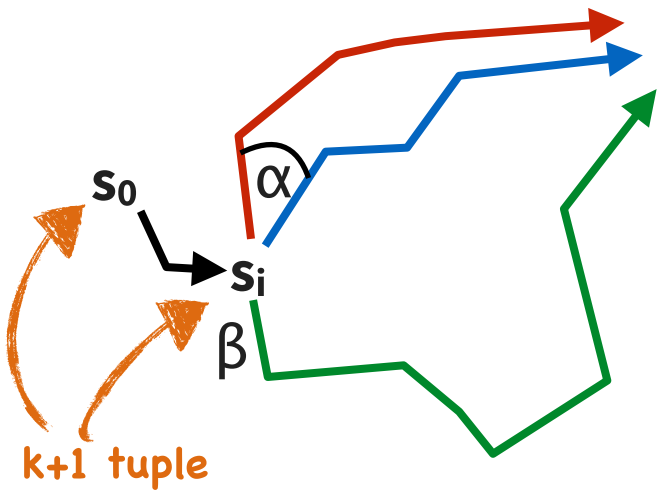
$2^{O(|\psi|^6)}$ different clause sets

Theorem

Space and time complexity polynomial in $|\mathbf{S}|$ and *single* exponential $|\psi|$

Policy Synthesis by Translation to Linear Program

Search Space

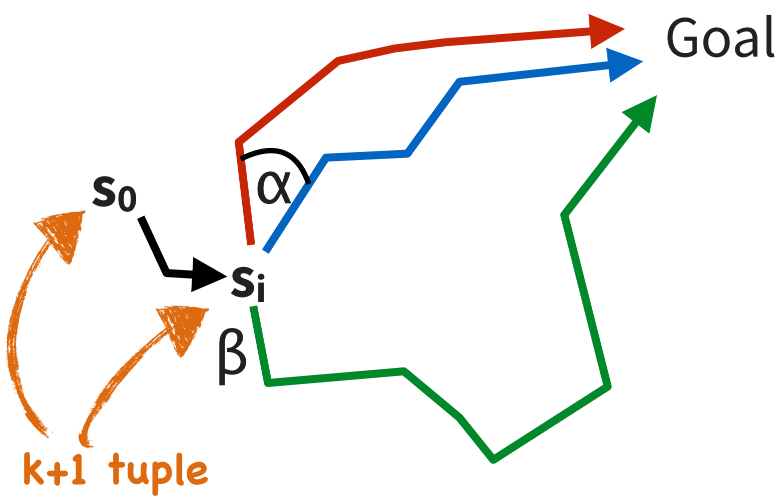


Policy π

$\pi(\alpha \mid s_i) = ?$ $\pi(\beta \mid s_i) = ?$ ….

Policy Synthesis by Translation to Linear Program

Search Space



Policy π

$\pi(\alpha \mid s_i) = ?$ $\pi(\beta \mid s_i) = ?$ ....

Linear program computes expected values

Expected number of times α is executed in s_i

$$x(s_i, \alpha) = \sum_{s_i} \pi(\alpha \mid s_i) \times \Pr(\;s_i\;)$$

Expected policy costs

$$\mathrm{Cost}(\;) \times \Pr(\;) +$$
$$\mathrm{Cost}(\;) \times \Pr(\;) +$$
$$\mathrm{Cost}(\;) \times \Pr(\;)$$

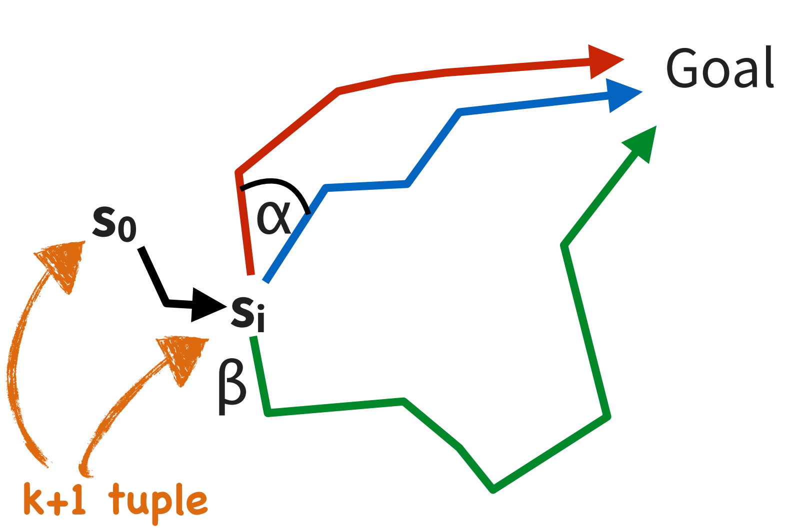
Primary: e.g. time

Secondary: e.g. fuel < 50

$$= \;\ldots + x(s_i, \alpha) \times C(\alpha)$$
$$+\; x(s_i, \beta) \times C(\beta) + \ldots$$

Policy Synthesis by Translation to Linear Program

Search Space



Policy π

$\pi(\alpha \mid s_i) = ?$ $\pi(\beta \mid s_i) = ?$ ….

Linear program computes expected values

Expected number of times α is executed in s_i

$$\mathbf{x(s_i, \alpha)} = \underline{\sum}_{s_i} \pi(\alpha \mid s_i) \times \Pr(\text{—} \; s_i)$$

Expected policy costs

Cost(—— ——) × Pr(—— ——) +
Cost(—— ——) × Pr(—— ——) +
Cost(—— ——) × Pr(—— ——)

Primary: e.g. time
Secondary: e.g. fuel < 50

$$= \; \ldots + \mathbf{x(s_i, \alpha) \times C(\alpha)}$$
$$+ \; \mathbf{x(s_i, \beta) \times C(\beta)} + \ldots$$

Linear Program Solver

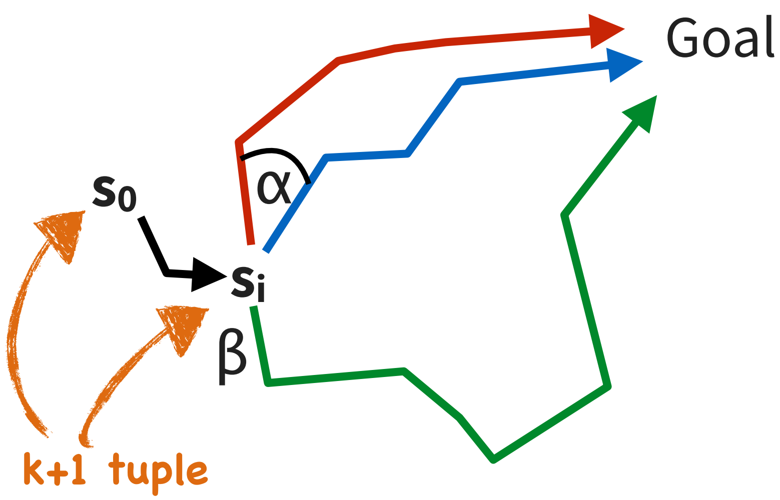
Optimal solution of linear program, i.e., values for **x(si, α)** s.th.

- **primary** cost is *minimized*, and
- **secondary** cost constraints are *satisfied*

in expectation

Policy Synthesis by Translation to Linear Program

Search Space



Linear program computes expected values

Expected number of times α is executed in s_i

$$\mathbf{x(s_i, \alpha)} = \sum_{s_i} \boldsymbol{\pi(\alpha \mid s_i)} \times \Pr(\text{—} \, \mathbf{s_i})$$

Expected policy costs

Cost(— —) × Pr(— —) +
Cost(— —) × Pr(— —) +
Cost(— —) × Pr(— —)

Primary: e.g. time
Secondary: e.g. fuel < 50

$$= \ \dots + \mathbf{x(s_i, \alpha) \times C(\alpha)}$$
$$+ \ \mathbf{x(s_i, \beta) \times C(\beta)} + \dots$$

Policy π

$\boldsymbol{\pi(\alpha \mid s_i)} = \mathbf{?} \quad \boldsymbol{\pi(\beta \mid s_i)} = \mathbf{?} \quad \dots.$

$\boldsymbol{\pi(\alpha \mid s_i)} = \mathbf{x(s_i, \alpha)} \, / \, (\mathbf{x(s_i, \alpha)} + \mathbf{x(s_i, \beta)})$

Linear Program Solver

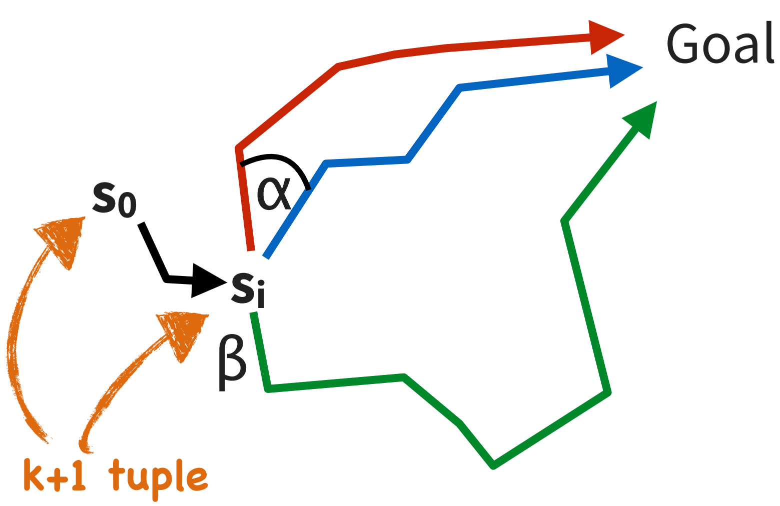
Optimal solution of linear program, i.e., values for **x(si, α)** s.th.

- **primary** cost is *minimized*, and
- **secondary** cost constraints are *satisfied*

in expectation

Policy Synthesis by Translation to Linear Program

Search Space



Linear program computes expected values

Expected number of times α is executed in s_i

$$x(s_i, \alpha) = \sum_{s_i} \pi(\alpha \mid s_i) \times \Pr(\text{—} s_i)$$

Expected policy costs

Cost(— —) × Pr(— —) +
Cost(— —) × Pr(— —) +
Cost(— —) × Pr(— —)

Primary: e.g. time
Secondary: e.g. fuel < 50

$$= \ldots + x(s_i, \alpha) \times C(\alpha)$$
$$+ x(s_i, \beta) \times C(\beta) + \ldots$$

Amenable to heuristics

Policy π

$\pi(\alpha \mid s_i) = ?$ $\pi(\beta \mid s_i) = ?$ ....

$\pi(\alpha \mid s_i) = x(s_i, \alpha) / (x(s_i, \alpha) + x(s_i, \beta))$

Linear Program Solver

Optimal solution of linear program, i.e., values for **x(si, α)** s.th.

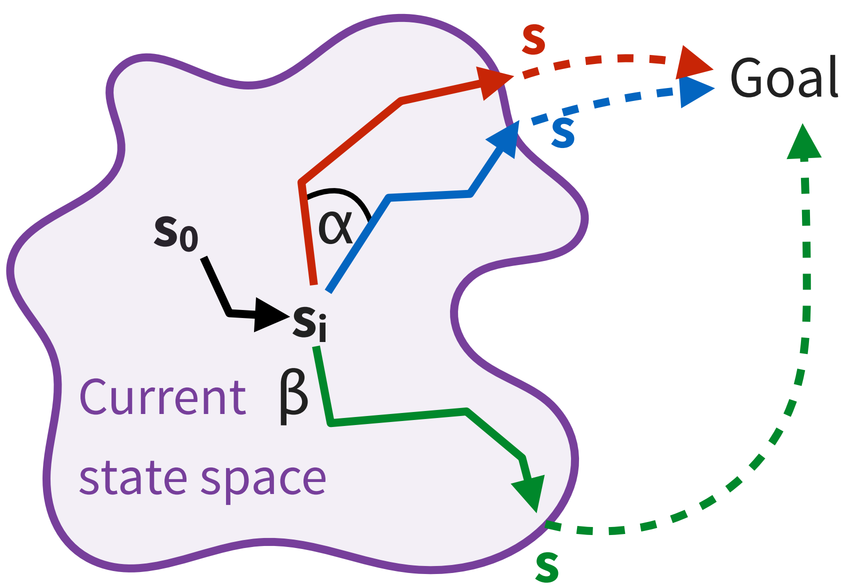
- **primary** cost is *minimized*, and
- **secondary** cost constraints are *satisfied*

in expectation

Heuristics Search: i-dual and i²-dual

- First heuristic search algorithms for constrained SSPs [Trevizan, Thiebaux, Haslum, Williams, Santana] i.e. primary expected cost ("time") and secondary expected cost constraints ("fuel < 5")
- Sound, complete and optimal for admissible heuristics H (H must understimate expected costs)

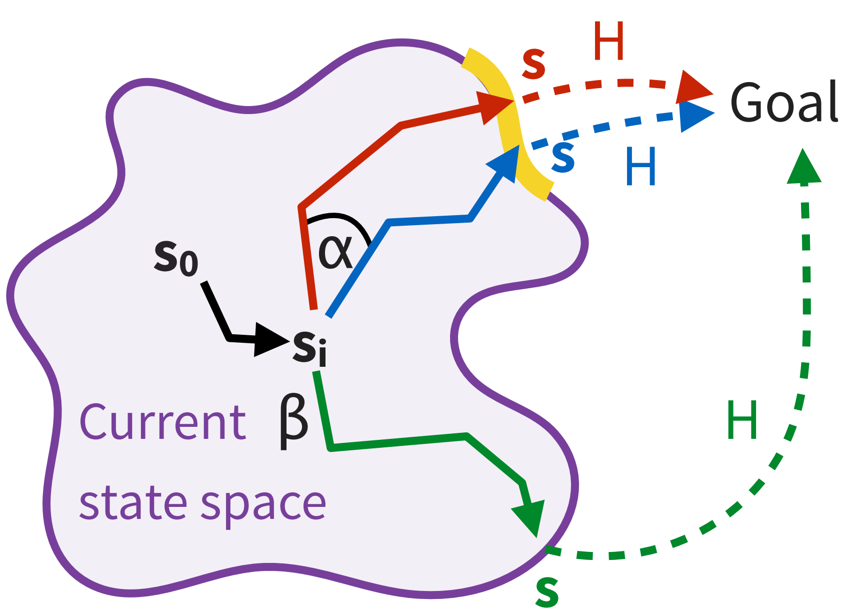
Exploring the state space …



Heuristics Search: i-dual and i²-dual

- First heuristic search algorithms for constrained SSPs [Trevizan, Thiebaux, Haslum, Williams, Santana] i.e. primary expected cost ("time") and secondary expected cost constraints ("fuel < 5")
- Sound, complete and optimal for admissible heuristics H (H must understimate expected costs)

Exploring the state space ...



... with A*-like heuristic estimation function H

(1) Compute best policy **π*** for current state space
by translation into LP with **fringe as artificial goals with costs H**
π* minimizes f = g + H

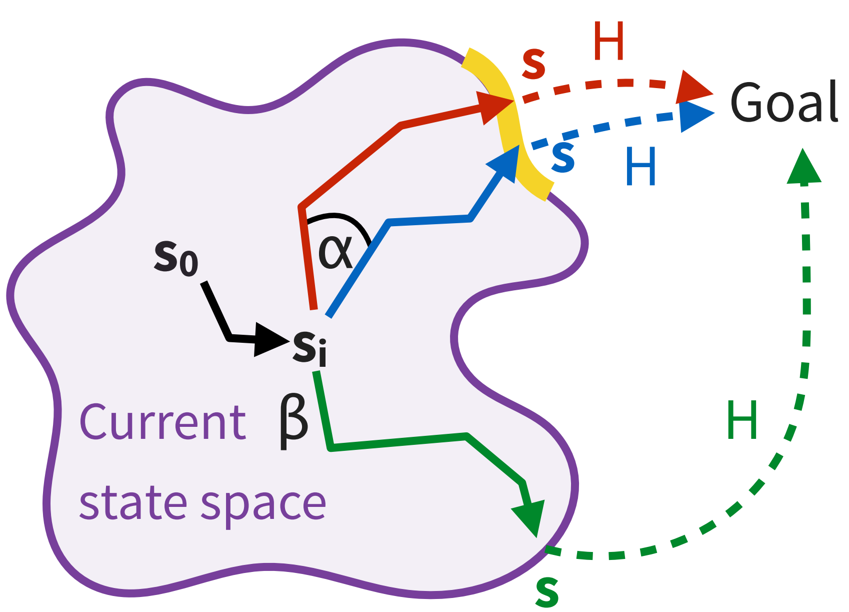
(2) Expand all **fringe states reachable under π***

(3) If all reachable fringe states are original goals then stop else repeat

Heuristics Search: i-dual and i²-dual

- First heuristic search algorithms for constrained SSPs [Trevizan, Thiebaux, Haslum, Williams, Santana] i.e. primary expected cost ("time") and secondary expected cost constraints ("fuel < 5")
- Sound, complete and optimal for admissible heuristics H (H must understimate expected costs)

Exploring the state space …



… with A*-like heuristic estimation function H

(1) Compute best policy **π*** for current state space

by translation into LP with **fringe as artificial goals with costs H**

π* minimizes f = g + H

(2) Expand all **fringe states reachable under π***

(3) If all reachable fringe states are original goals

then stop else repeat

Search space

- Over policies, not paths; g(s) may change in each step
- Policies may become constrained

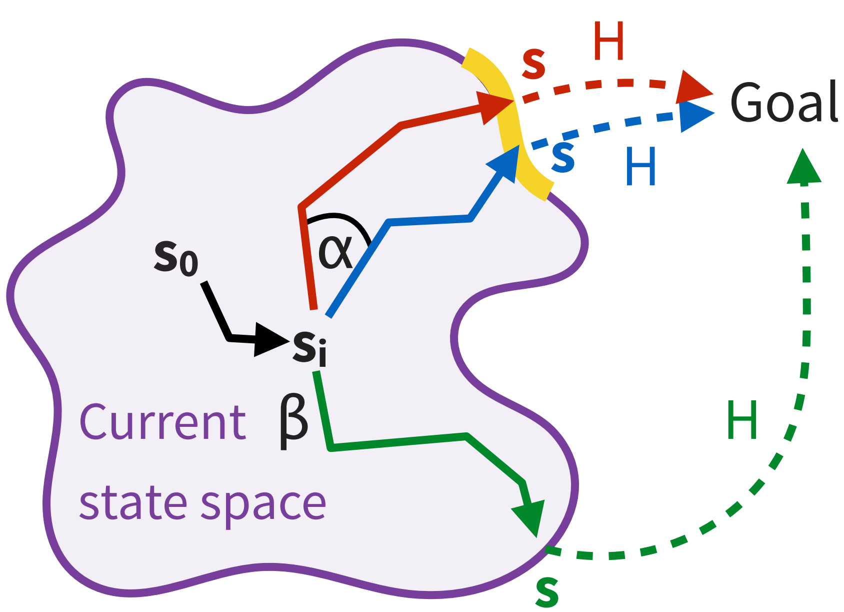
E.g. Pr(— — - -) < 0.1 if H_{fuel}(**s**) = 50

as otherwise fuel < 5 not achievable

Heuristics Search: i-dual and i²-dual

- First heuristic search algorithms for constrained SSPs [Trevizan, Thiebaux, Haslum, Williams, Santana] i.e. primary expected cost (“time”) and secondary expected cost constraints (“fuel < 5”)
- Sound, complete and optimal for admissible heuristics H (H must understimate expected costs)

Exploring the state space …



→ For PLTL constraints

… with A*-like heuristic estimation function H

(1) Compute best policy **π*** for current state space

by translation into LP with **fringe as artificial goals with costs H**

π* minimizes f = g + H

(2) Expand all **fringe states reachable under π***

(3) If all reachable fringe states are original goals

then stop else repeat

Search space

- Over policies, not paths; g(s) may change in each step
- Policies may become constrained

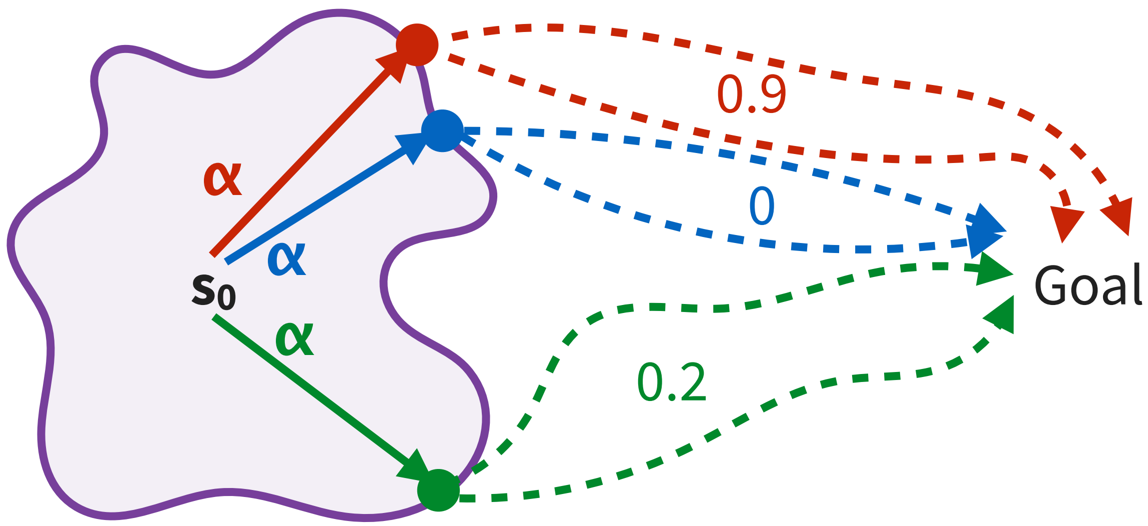
  E.g. Pr(— — - -) < 0.1 if H_{fuel}(**s**) = 50

  as otherwise fuel < 5 not achievable

Heuristic Search for PLTL - PLTL-dual

A universal heuristic for search space pruning

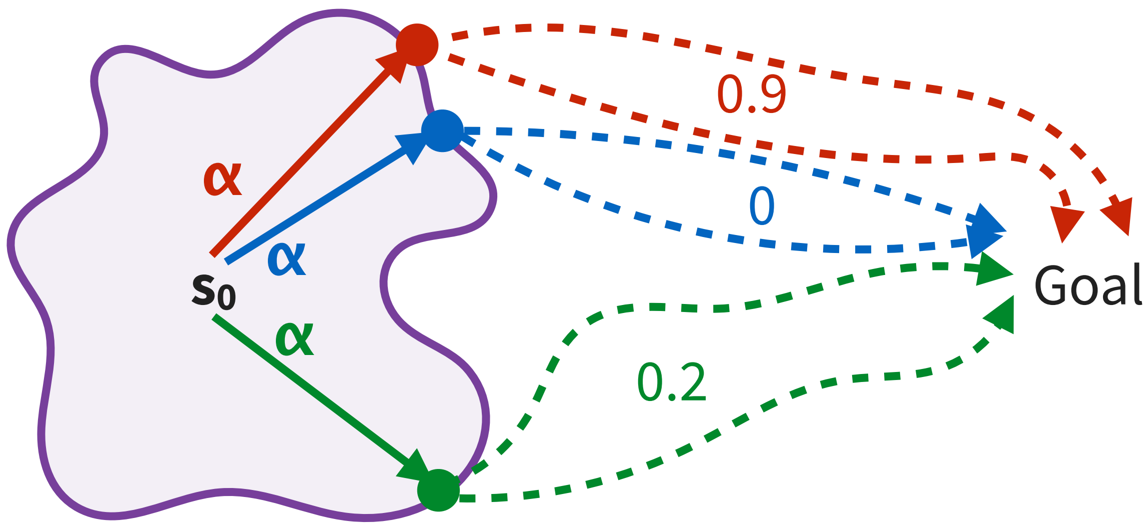
Find policy π <u>s.th</u> $\mathbf{s_0}, \pi \models \mathbf{P}_{\geq 0.9}\Psi$



Heuristic Search for PLTL - PLTL-dual

A universal heuristic for search space pruning

Find policy π s.th **s_0**, π ⊨ **P**$_{\geq 0.9}\Psi$



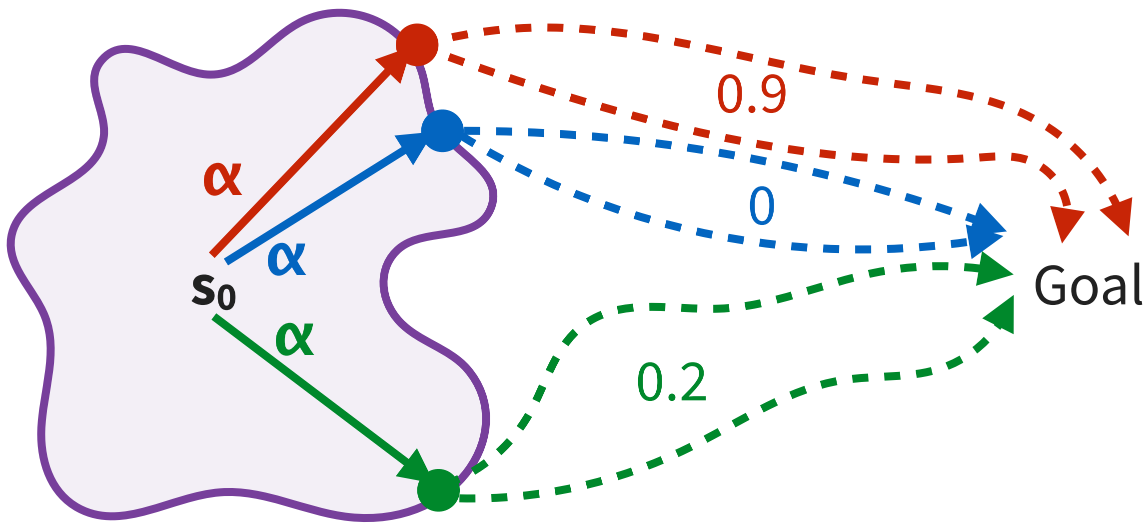
Optimal (final) policy π*

π*(α, s_0) = 1 **π*(α, s_0)** = 0 **π*(α, s_0)** = 0

Heuristic Search for PLTL - PLTL-dual

A universal heuristic for search space pruning

Find policy π s.th $\mathbf{s_0}, \pi \models \mathbf{P}_{\geq 0.9}\Psi$



Optimal (final) policy π*

$\pi^*(\alpha, s_0) = 1 \quad \pi^*(\alpha, s_0) = 0 \quad \pi^*(\alpha, s_0) = 0$

Max among all π* ≤ **Heuristic value**

$\Pr\{ \bullet\text{----} \mid \Psi \} = 0.9 \leq H(\bullet) = 1$

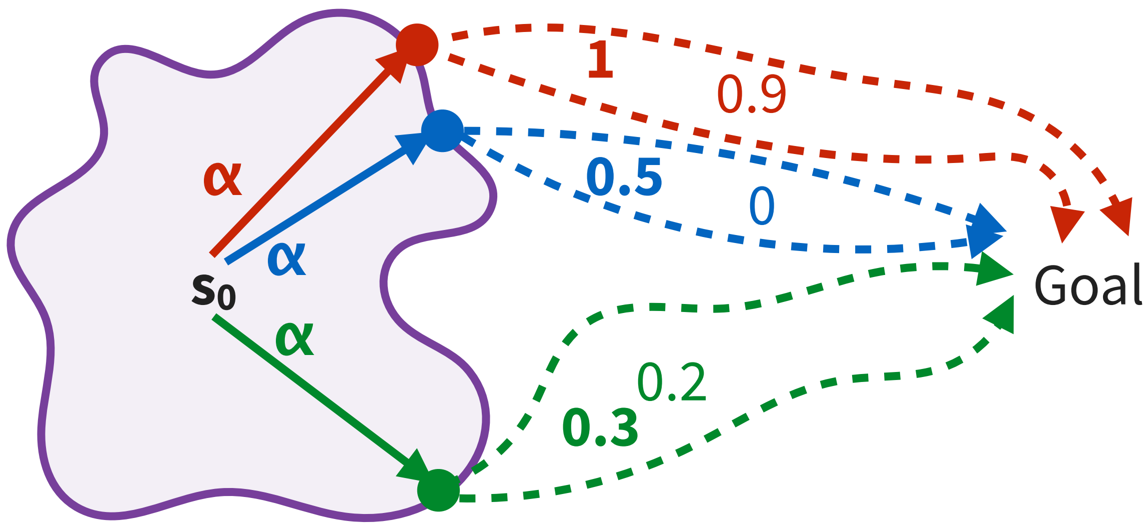
$\Pr\{ \bullet\text{----} \mid \Psi \} = 0 \leq H(\bullet) = 0.5$

$\Pr\{ \bullet\text{----} \mid \Psi \} = 0.2 \leq H(\bullet) = 0.3$

Heuristic Search for PLTL - PLTL-dual

A universal heuristic for search space pruning

Find policy π <u>s.th</u> $\mathbf{s_0}, \pi \models \mathbf{P}_{\geq 0.9}\Psi$



Optimal (final) policy π*

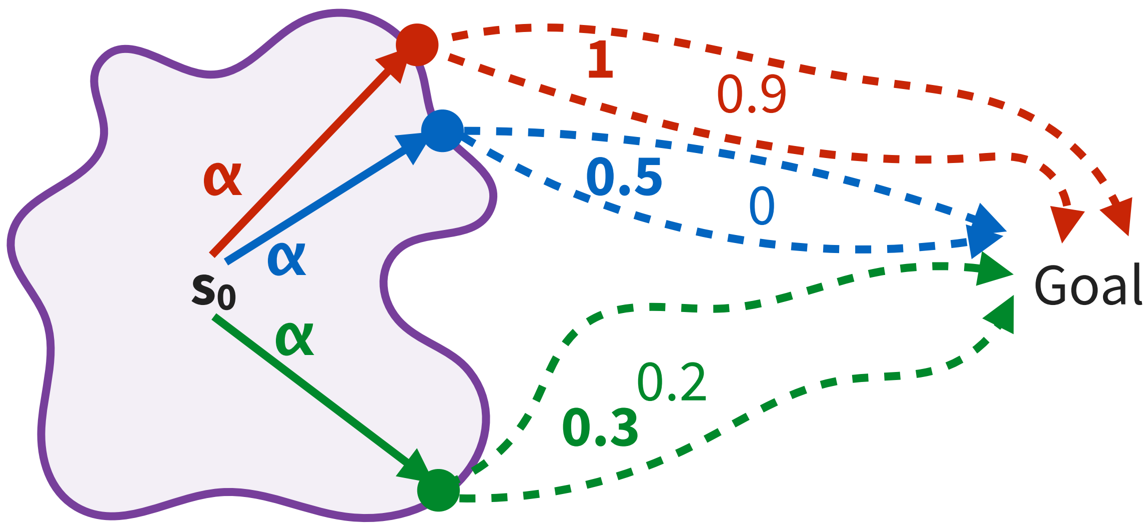
$\boldsymbol{\pi^*(\alpha, s_0)} = 1$ $\boldsymbol{\pi^*(\alpha, s_0)} = 0$ $\boldsymbol{\pi^*(\alpha, s_0)} = 0$

Max among all π*	**≤**	**Heuristic value**
Pr{ ● - - - | Ψ } = 0.9	≤	H(●) = 1
Pr{ ● - - - | Ψ } = 0	≤	H(●) = 0.5
Pr{ ● - - - | Ψ } = 0.2	≤	H(●) = 0.3

Heuristic Search for PLTL - PLTL-dual

A universal heuristic for search space pruning

Find policy π s.th $s_0, \pi \vDash P_{\geq 0.9}\Psi$



Optimal (final) policy π*

$\pi^*(\alpha, s_0) = 1$ $\pi^*(\alpha, s_0) = 0$ $\pi^*(\alpha, s_0) = 0$

Max among all π* ≤ **Heuristic value**

Pr{ ● - - - | Ψ } = 0.9 ≤ H(●) = 1

Pr{ ● - - - | Ψ } = 0 ≤ H(●) = 0.5

Pr{ ● - - - | Ψ } = 0.2 ≤ H(●) = 0.3

Entailed feasibilty policy constraint

$\pi(\alpha, s_0) \leq 0.2$

Otherwise, e.g. with $\pi(\alpha, s_0) = \mathbf{0.21}$

$\mathbf{0.21} \cdot 0.5 + \pi(\alpha, s_0) \cdot 1 \geq 0.9$

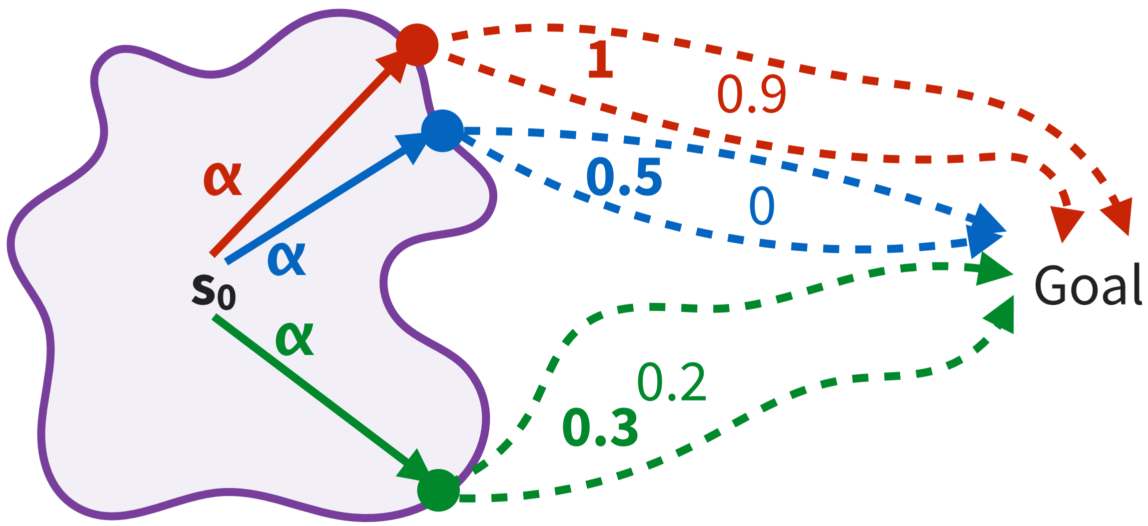
$\Rightarrow \pi(\alpha, s_0) \geq 0.795$

But $\mathbf{0.21} + \mathbf{0.795} = 1.005 > \mathbf{1}$

Heuristic Search for PLTL - PLTL-dual

A universal heuristic for search space pruning

Find policy π s.th $\mathbf{s_0}, \pi \vDash \mathbf{P}_{\geq 0.9}\Psi$



How to compute H(●) with NBAs

1. Ψ' := Ψ ∧ "finite extension semantics"
2. Compute NBA **B** for Ψ'
3. Trace **B** to find ● - states (overapproximation)
4. Trace **B** from ● - states as initial states to Goal
   - using relaxed actions from **S** consistent with trace
   - as a SSP **T**
5. Solve **T** putting 1 unit of flow into ● - states
6. Get H(●) from flow into Goal

Optimal (final) policy π*

$\pi^*(\alpha, s_0) = 1$ $\pi^*(\alpha, s_0) = 0$ $\pi^*(\alpha, s_0) = 0$

Max among all π* ≤ Heuristic value

Pr { ●- - - | Ψ } = 0.9 ≤ H(●) = 1

Pr { ●- - - | Ψ } = 0 ≤ H(●) = 0.5

Pr { ●- - - | Ψ } = 0.2 ≤ H(●) = 0.3

Entailed feasibilty policy constraint

$\pi(\alpha, s_0) \leq 0.2$

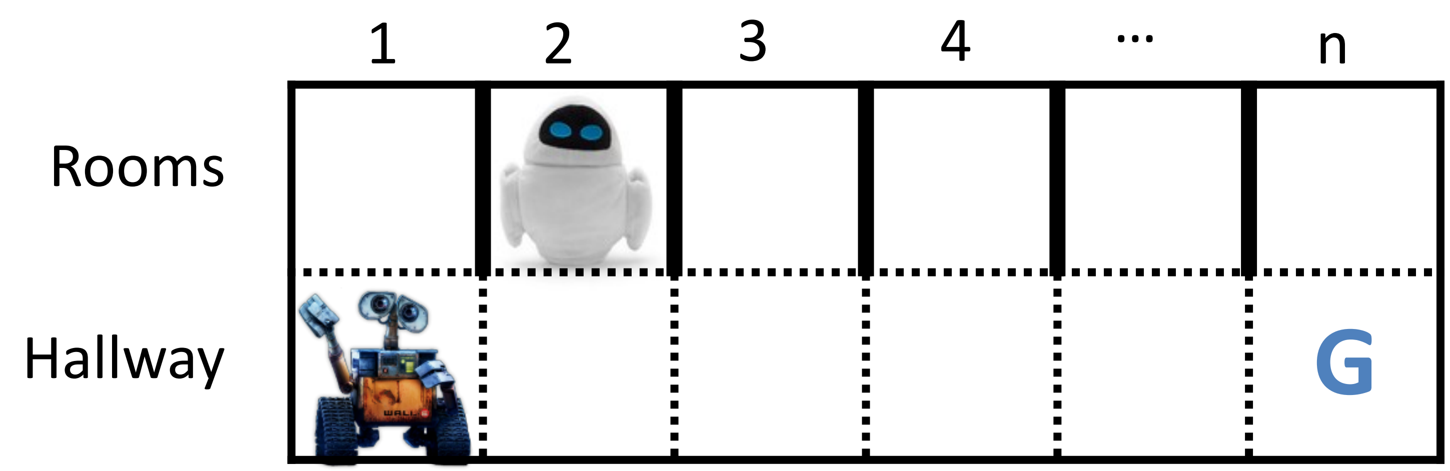
Otherwise, e.g. with $\pi(\alpha, s_0) = \mathbf{0.21}$

$\mathbf{0.21} \cdot 0.5 + \pi(\alpha, s_0) \cdot 1 \geq 0.9$

$\Rightarrow \pi(\alpha, s_0) \geq 0.795$

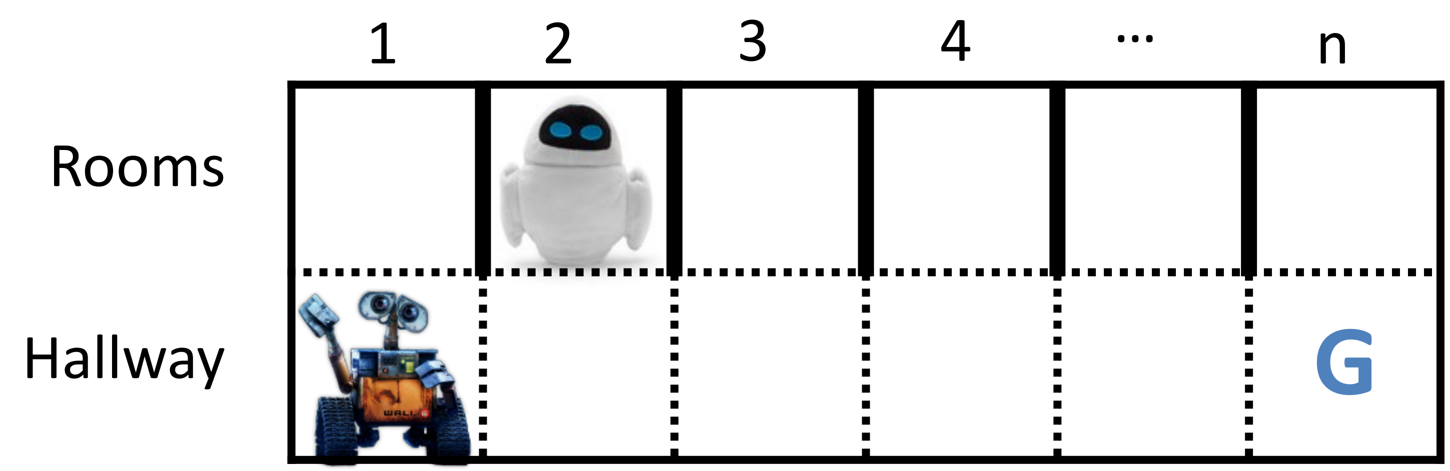
But $\mathbf{0.21} + \mathbf{0.795} = 1.005 > \mathbf{1}$

Experiment: Wall-e and Eve



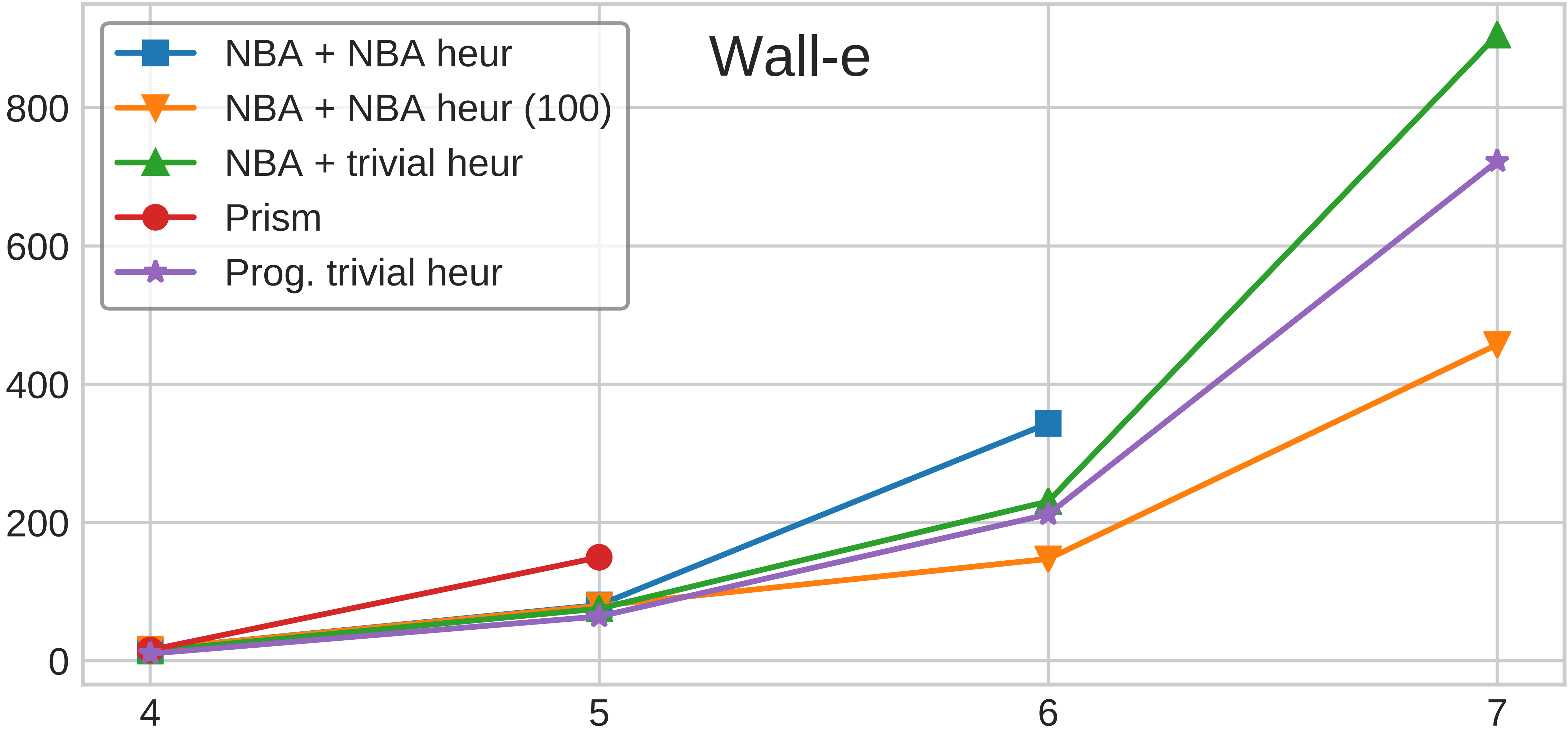
- **Goal**: Wall-e at G
- **Constraints**:
  1. Wall-e and Eve must eventually be together (P ≥ 0.5)
  2. Eve must be in a room until they are together (P ≥ 0.8)
  3. Once together, they eventually stay together (P = 1)
  4. Eve must visit the rooms 1, 2, and 3 (P = 1)
  5. Wall-e never visits a room twice (P ≥ 0.8)

Experiment: Wall-e and Eve



- **Goal**: Wall-e at G
- **Constraints**:
  1. Wall-e and Eve must eventually be together (P ≥ 0.5)
  2. Eve must be in a room until they are together (P ≥ 0.8)
  3. Once together, they eventually stay together (P = 1)
  4. Eve must visit the rooms 1, 2, and 3 (P = 1)
  5. Wall-e never visits a room twice (P ≥ 0.8)

Experiments - Wall-E



NBA heur: full heuristics, may yield “many” states

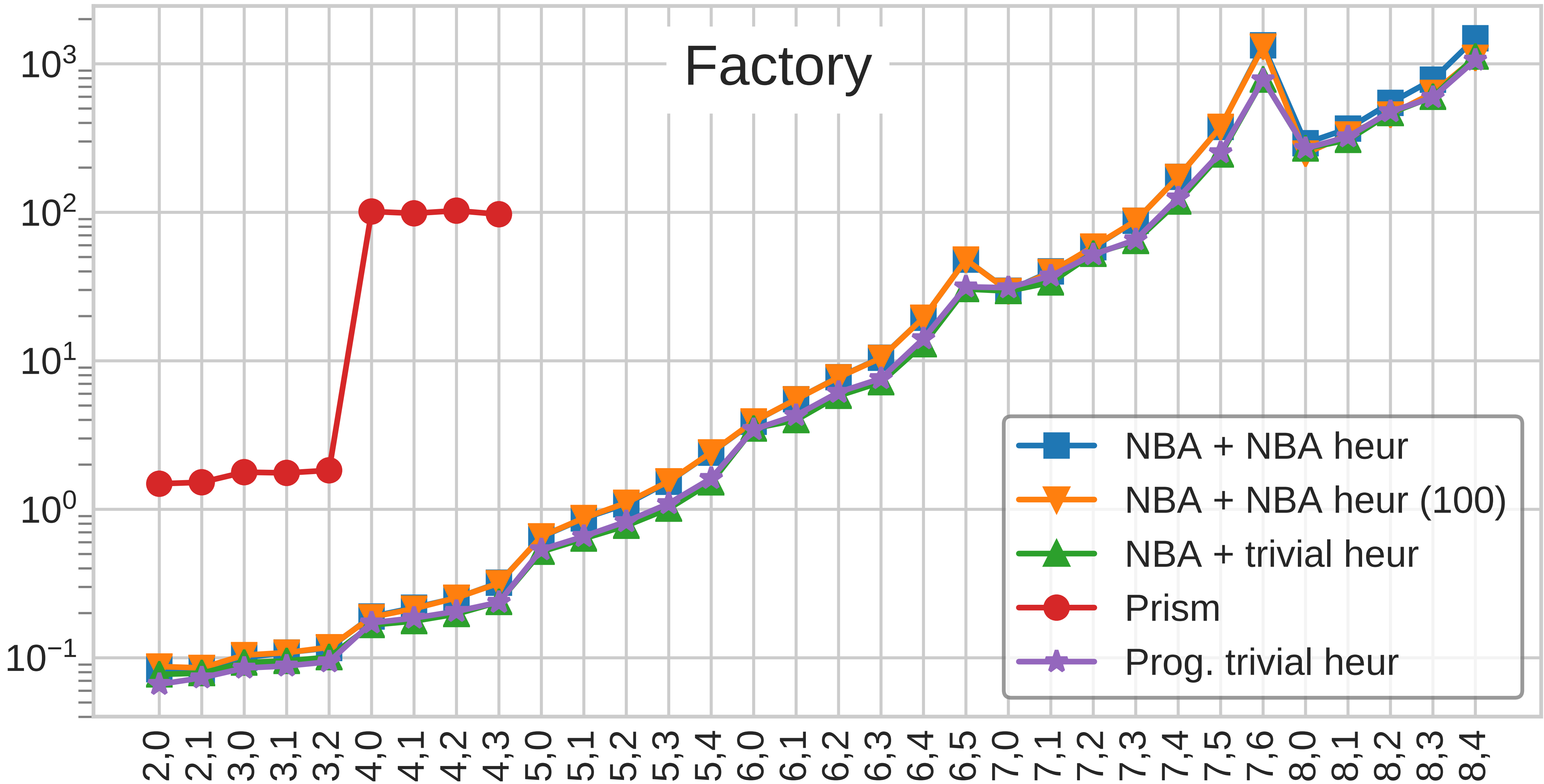
NBA heur (100): use trivial heuristics if > 100 states in NBA

Good also for progression: violated LTL constraints detected early by simplification

Wall-E never visits room1 twice

G (wall-E_room1 ⇒ (wall-E_room1 **U G** ¬wall-E_room1) (ψ_3)

Experiments - Factory



Conclusion

Summary

- Policy synthesis algorithm for multi-objective PLTL constraints $\Psi = \mathbf{P_1}\,\psi_1 \wedge \cdots \wedge \mathbf{P_k}\,\psi_k$

  Resulting history-independent (Markovian) policy over cross-product state space converts to finite-memory policy in the standard way
- Tseitin-style progression

  Better worst-case complexity: single-exponential (vs double-exponential) in $|\Psi|$
- NBA-based A*-like heuristics
- "Promising experiments"

Future Work

- Implement progression in full
- Heuristics based on progression (vs NBA)
- Multi-objective PLTL verification (on infinite runs) based on progression
- Quantification over finite domains. Non-prob: [Baier&McIlraith 2006]
- Beyond PLTL, e.g. $\mathbf{P}_{>0.8}\,\mathbf{G}\,(\mathbf{A} \rightarrow \mathbf{P}_{>0.4}\,\mathbf{F}\,\mathrm{B})$