New Challenges for Automata Learning

Property-Directed Verification of Recurrent Neural Networks

Benedikt Bollig

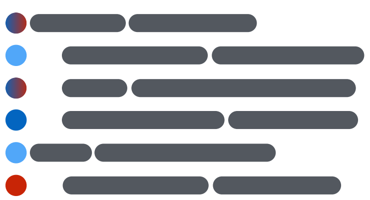
CNRS, LSV, ENS Paris-Saclay, Université Paris-Saclay

joint work with LeaRNNify team:

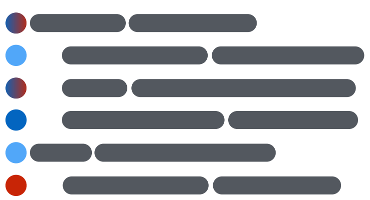
Benoît Barbot, Alain, Finkel, Serge Haddad, Igor Khmelnitsky, Daniel Neider,

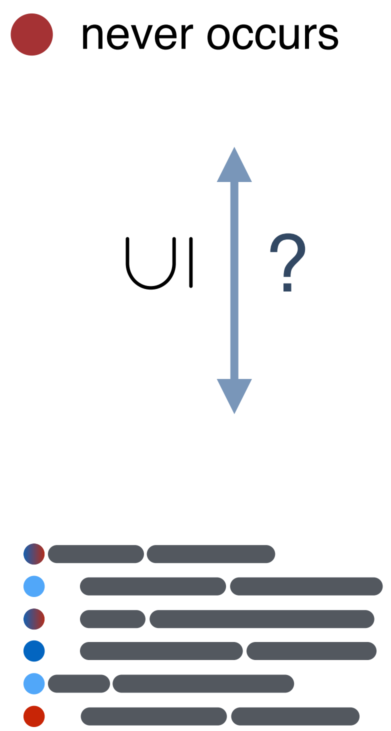
Martin Leucker, Rajarshi Roy, Lina Ye

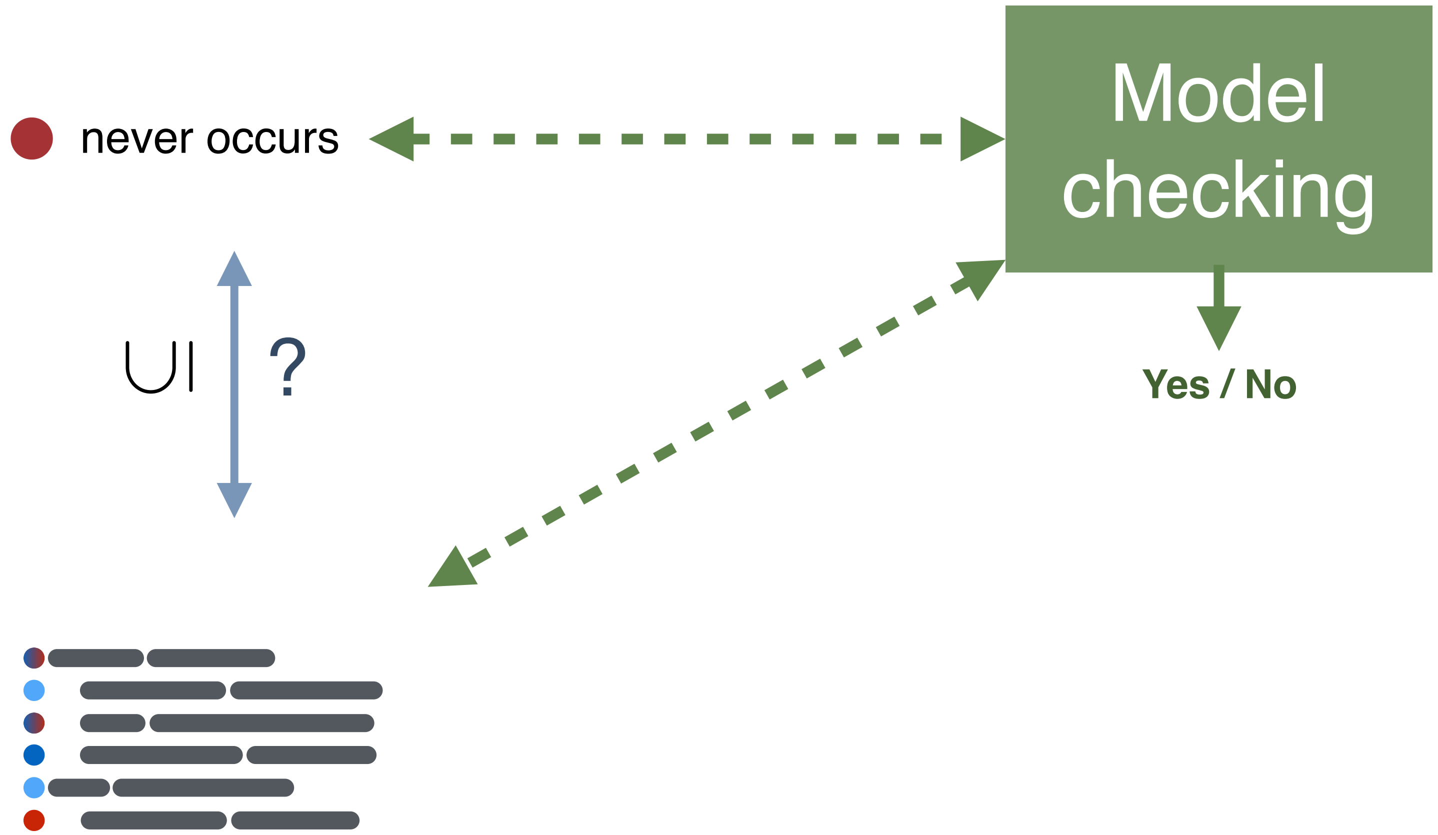
DATAIA Workshop "Safety & AI"
September 23, 2020
CentraleSupélec



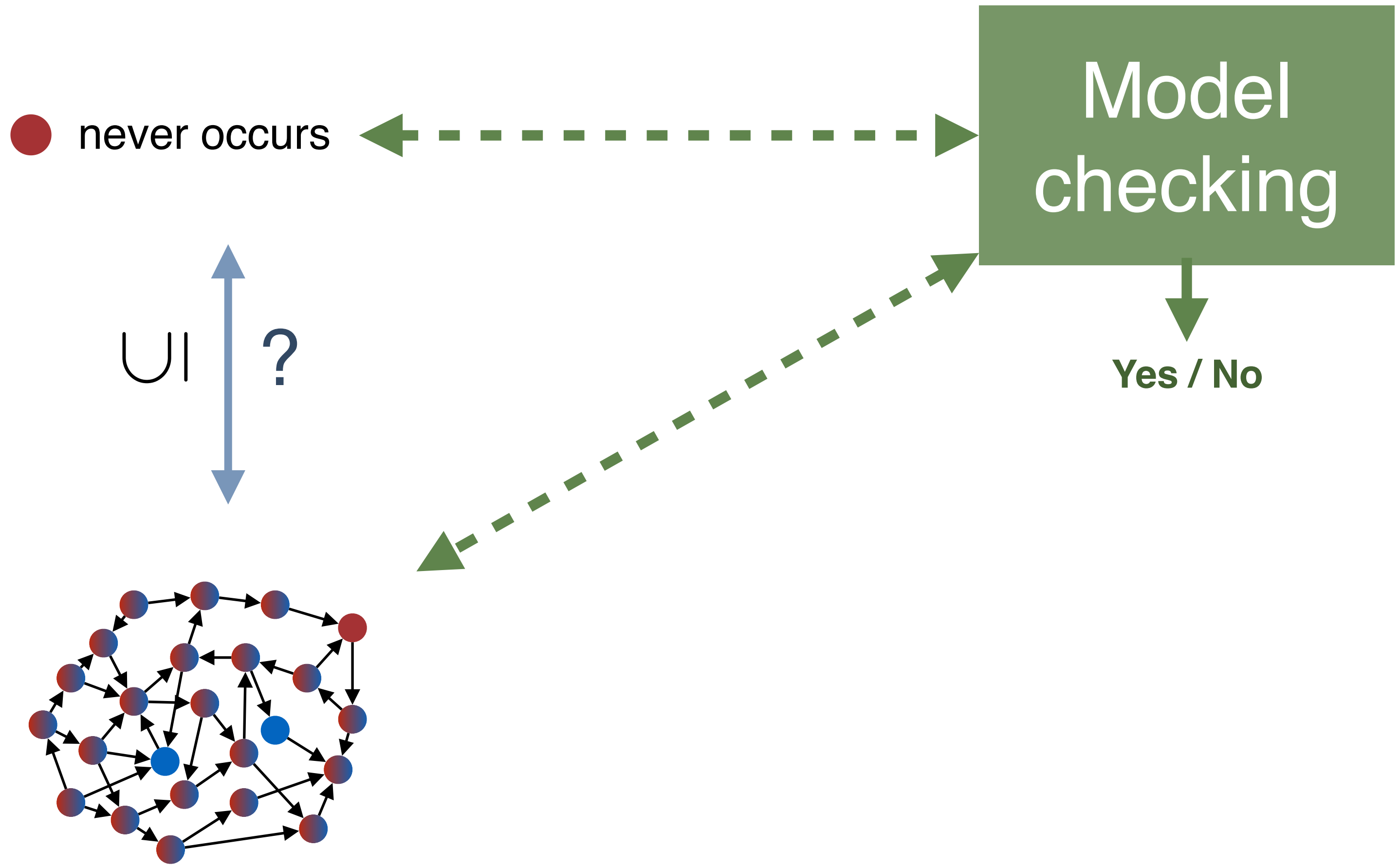
never occurs

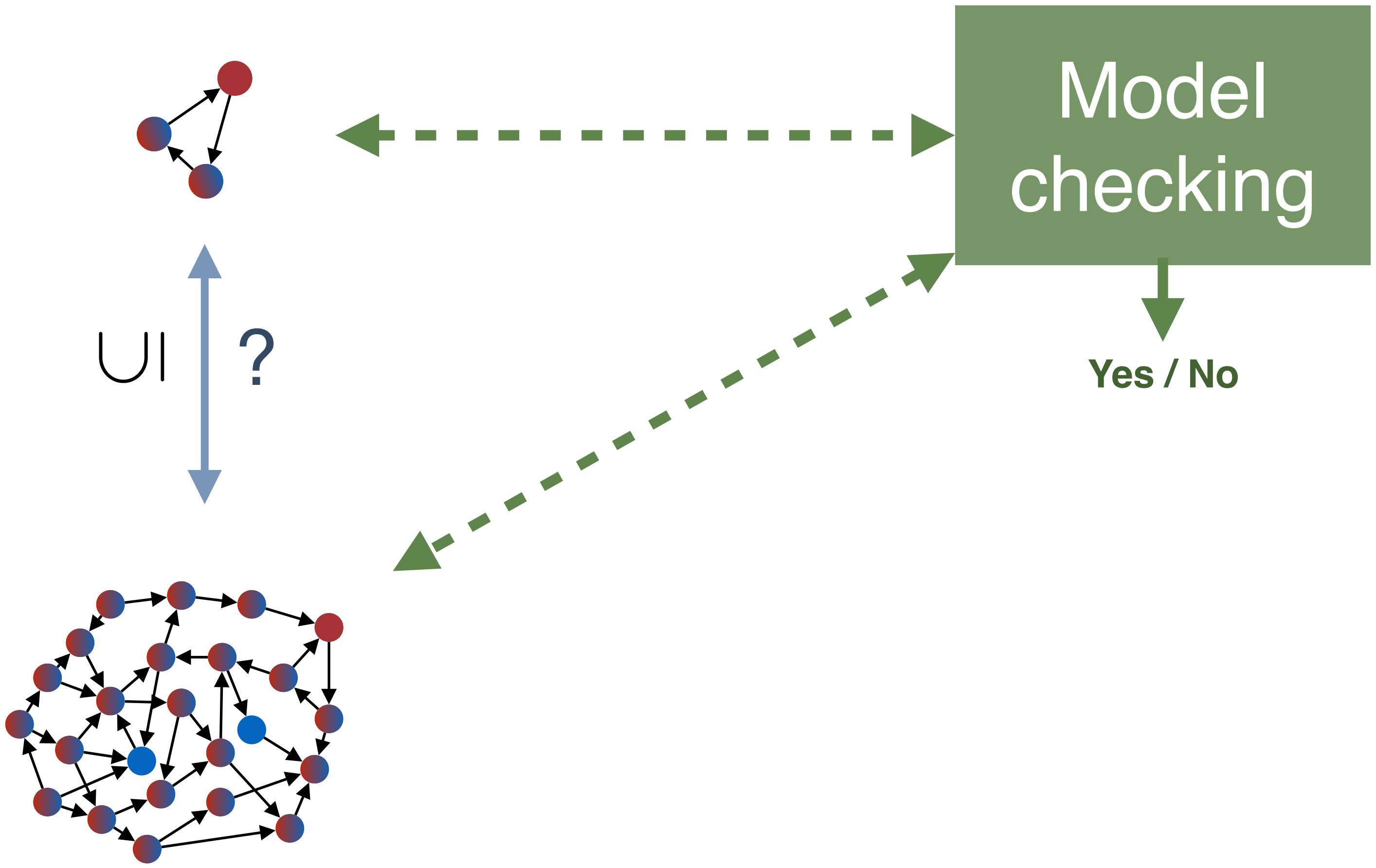



never occurs
?

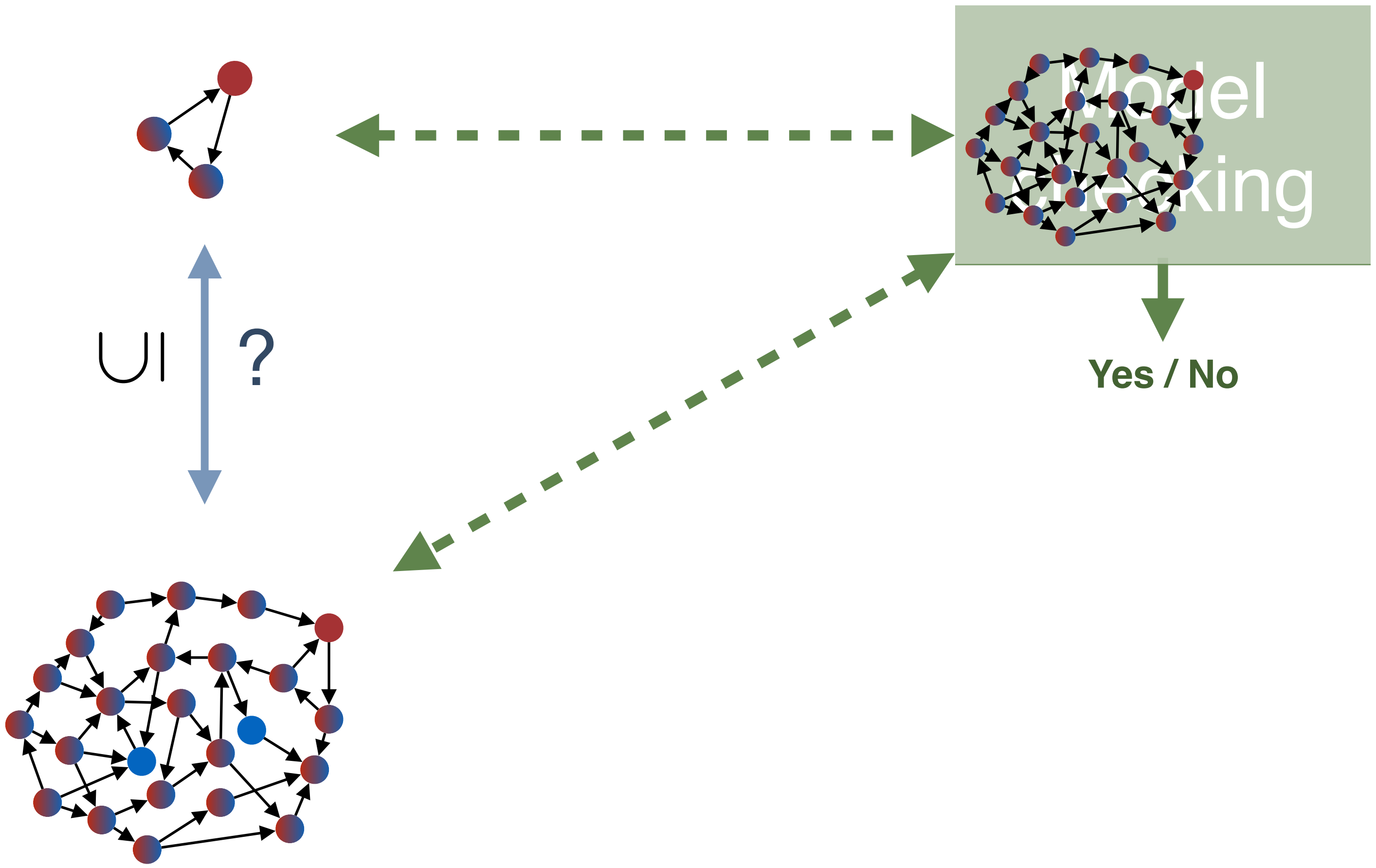


never occurs
Model checking
Yes / No
?

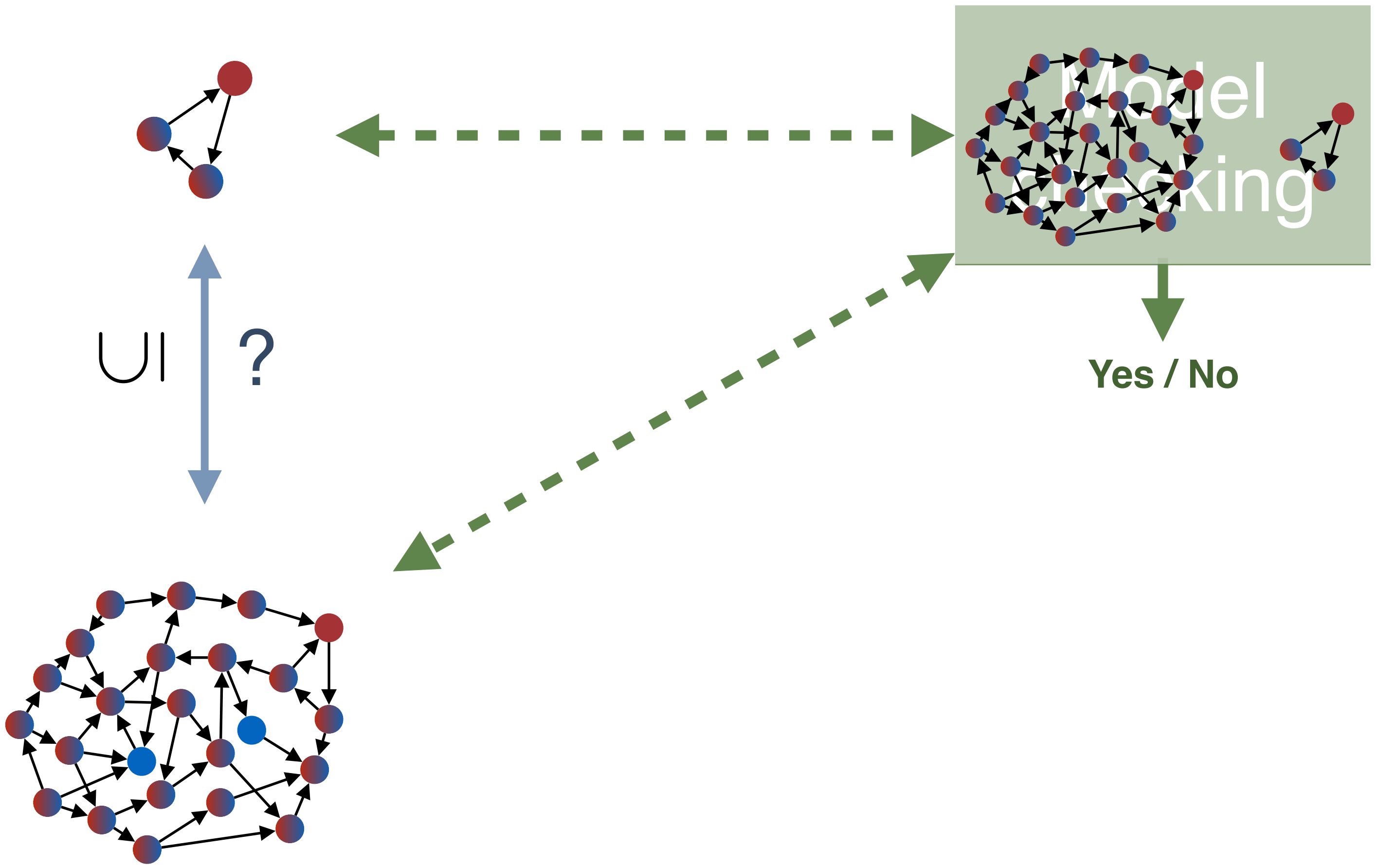

never occurs
Model checking
Yes / No
?

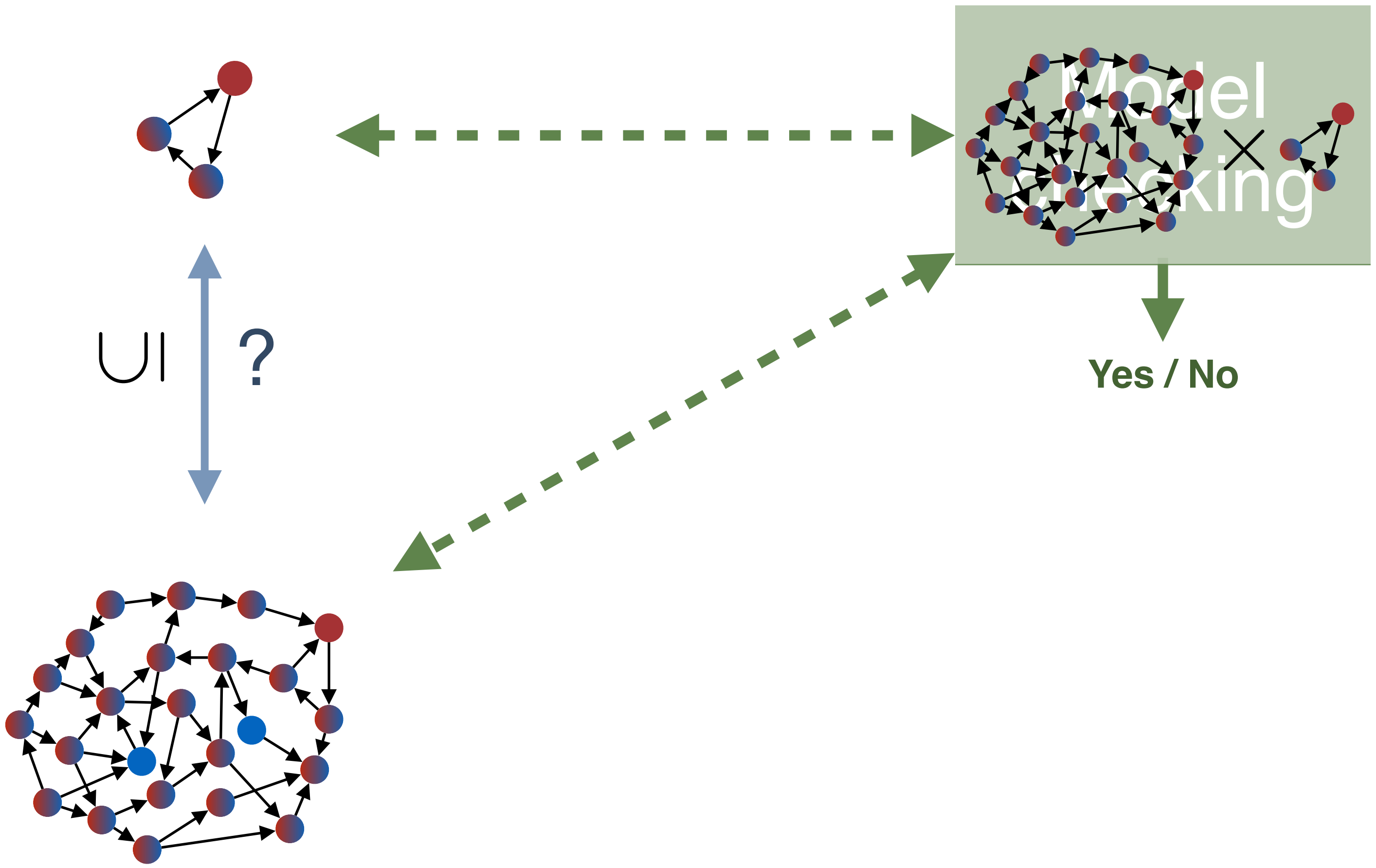


Model checking
Yes / No
?

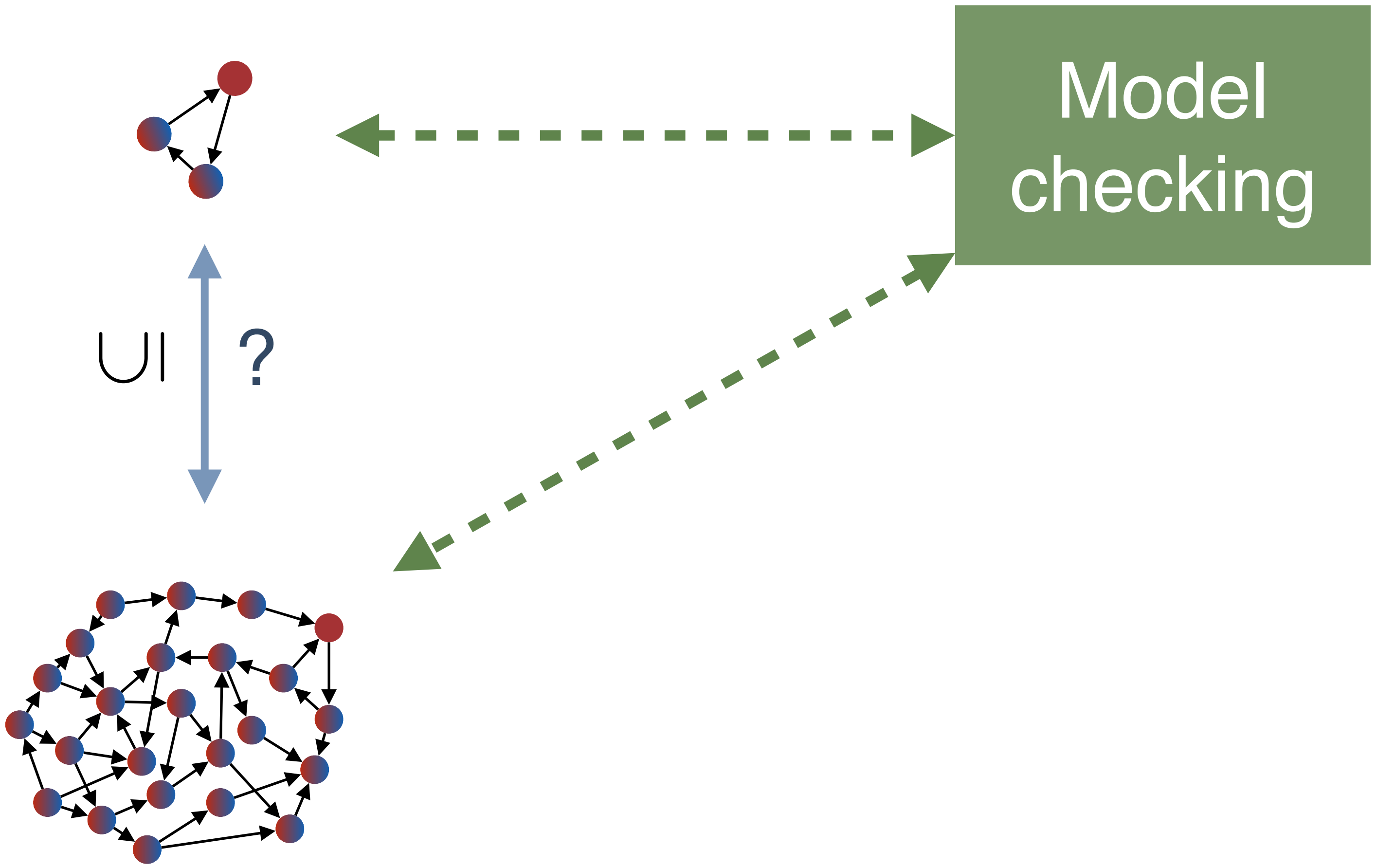


Model Checking
Yes / No
∪I ?


Yes / No

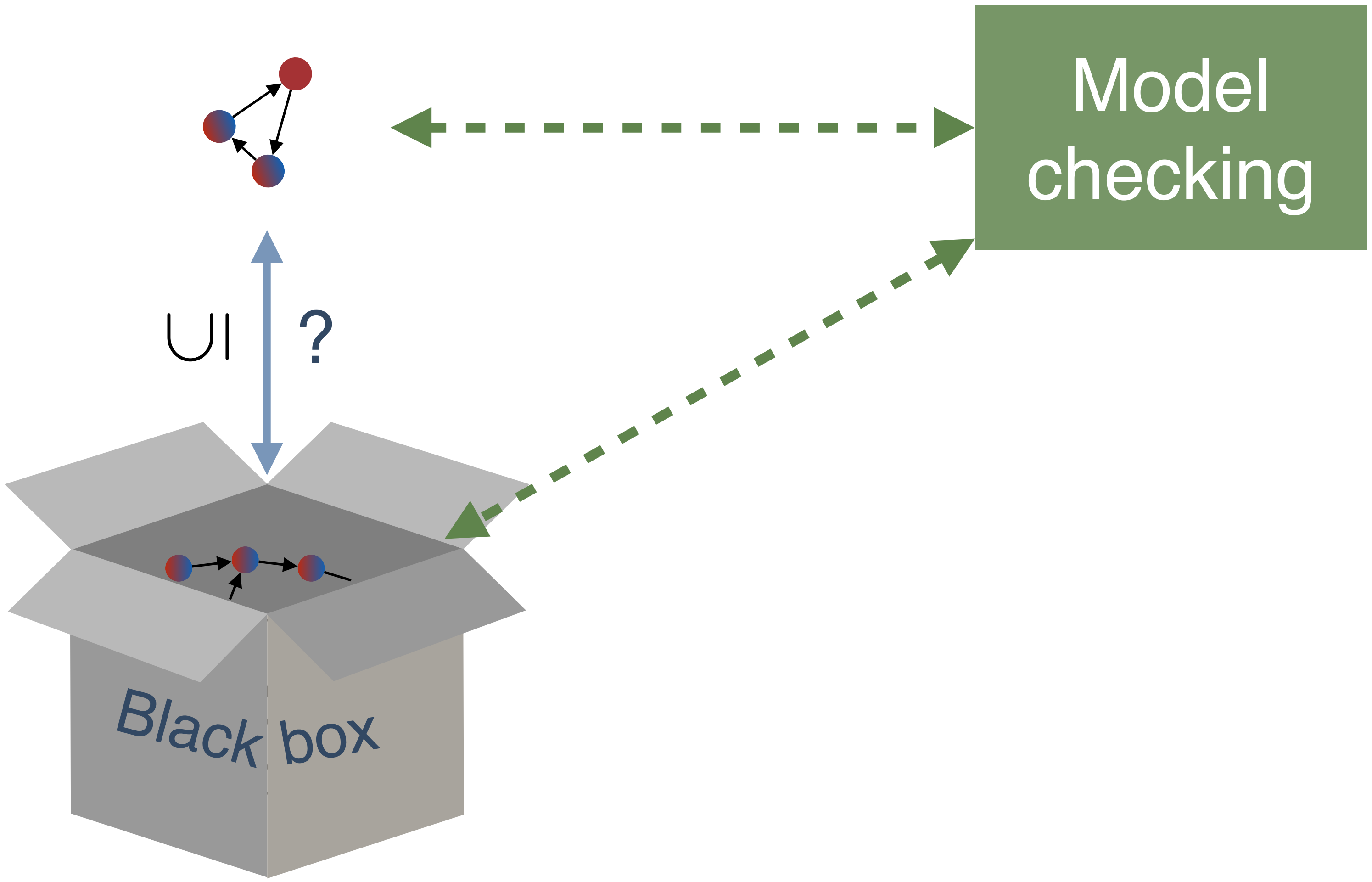


Model
×
?
Yes / No



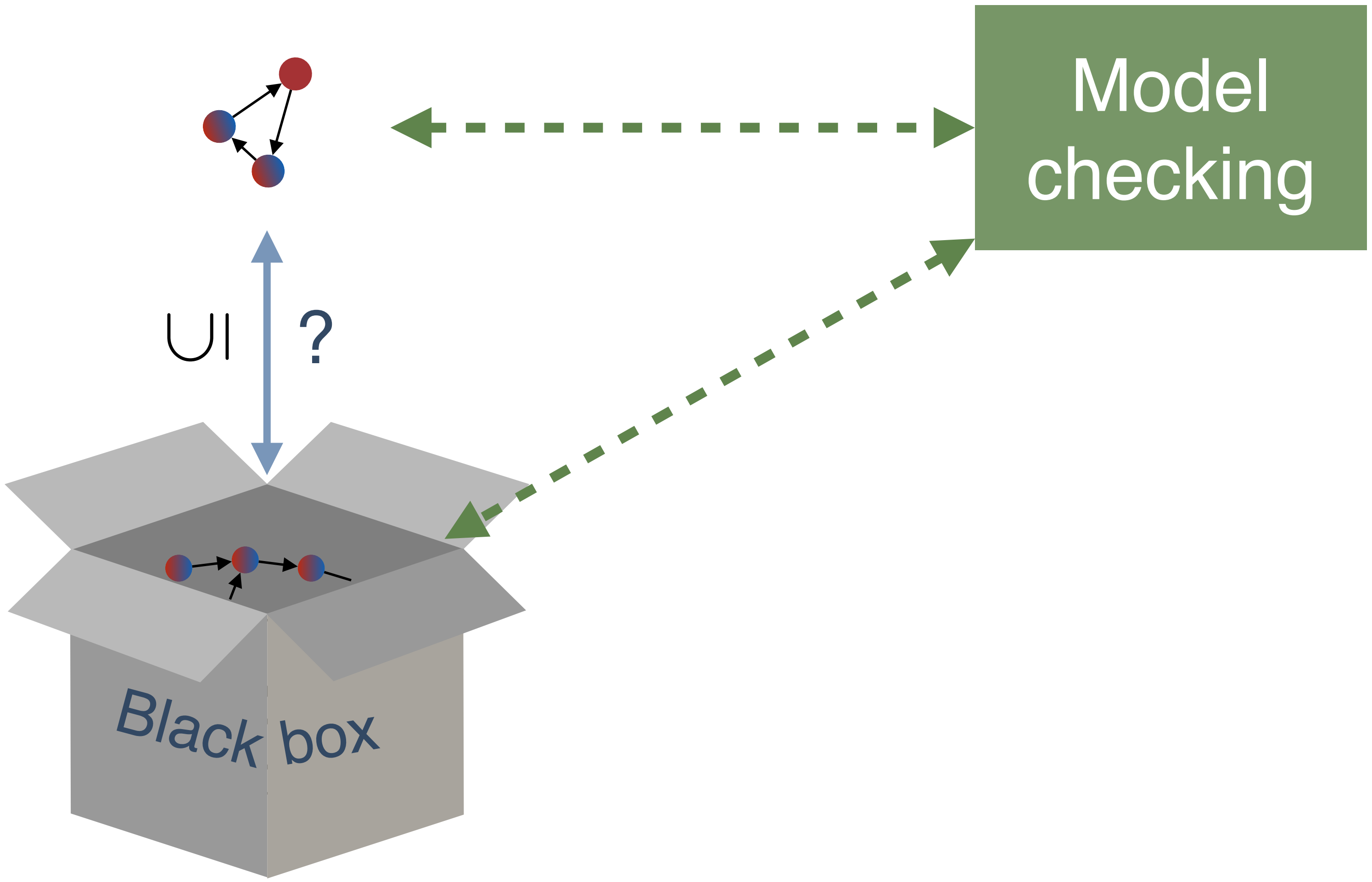
Model Checking

[Clarke-Emerson, Queille-Sifakis '80s]



Model Checking

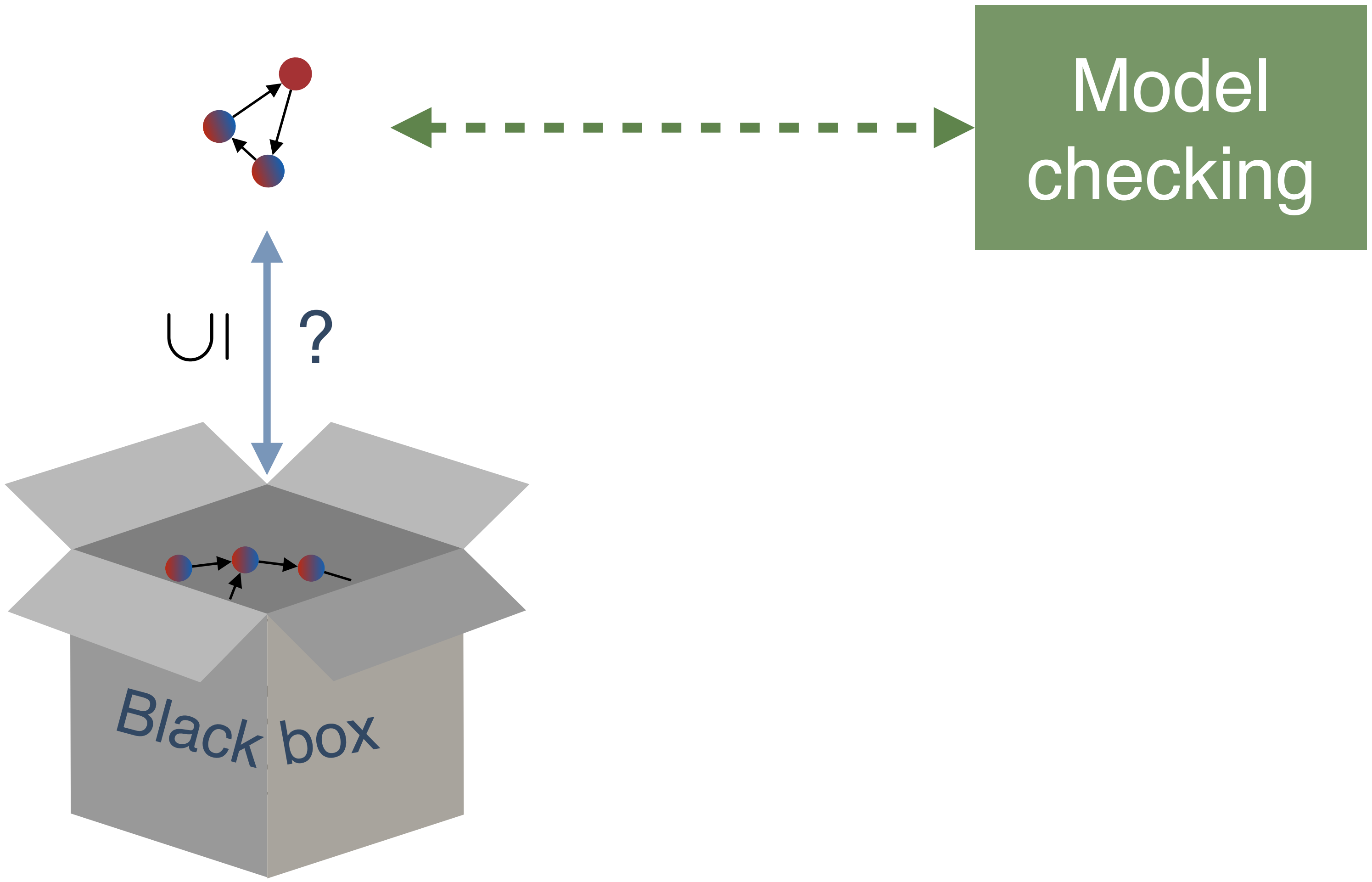
[Clarke-Emerson, Queille-Sifakis '80s]



Model Checking

[Clarke-Emerson, Queille-Sifakis '80s]

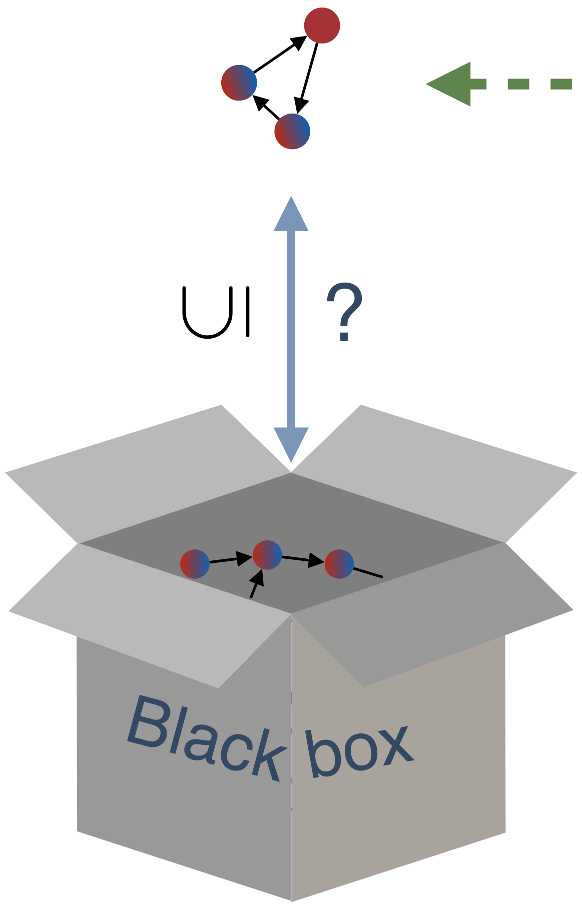
- legacy software
- code not open source
- third-party software
- embedded systems



- legacy software
- code not open source
- third-party software
- embedded systems

Model Checking

[Clarke-Emerson, Queille-Sifakis '80s]



Model checking

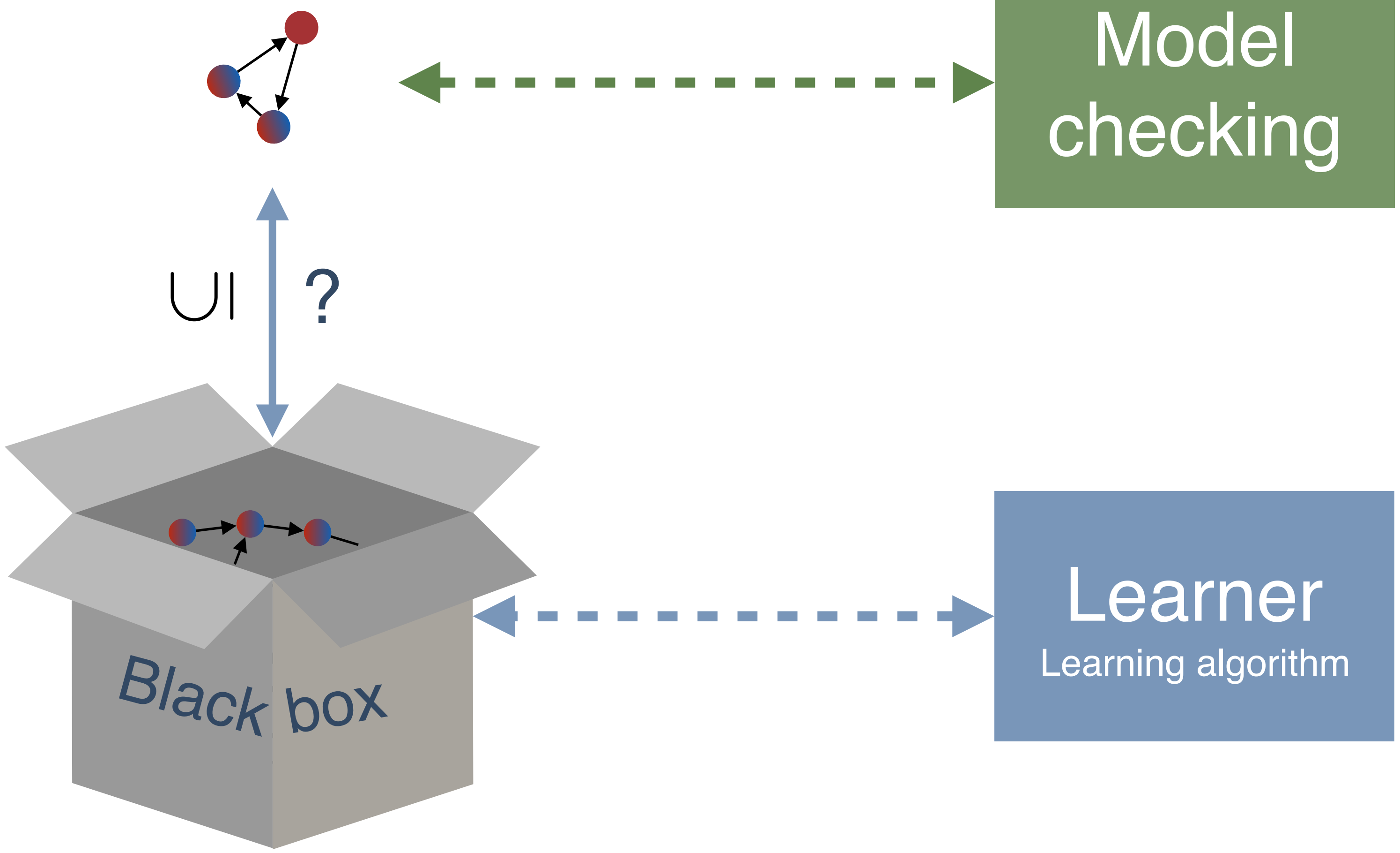
Model Checking

[Clarke-Emerson, Queille-Sifakis '80s]

Learner

Learning algorithm

- legacy software
- code not open source
- third-party software
- embedded systems

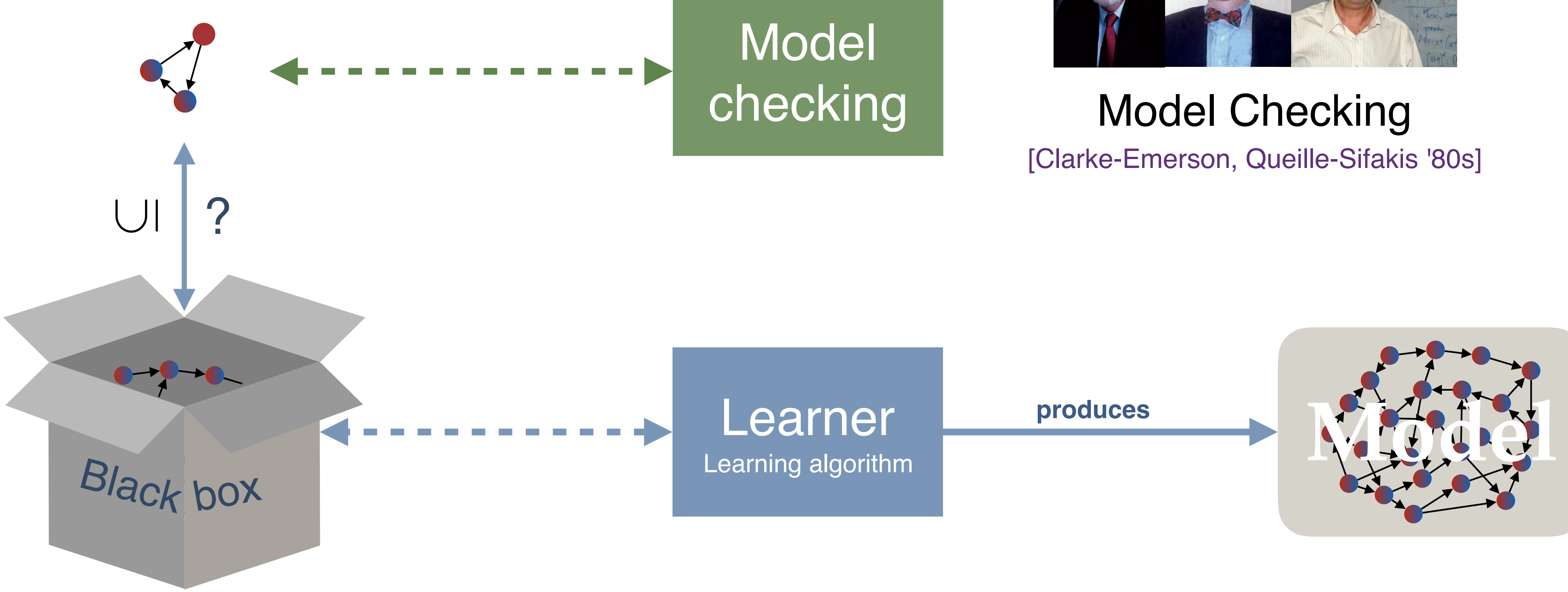


Model Checking
[Clarke-Emerson, Queille-Sifakis '80s]

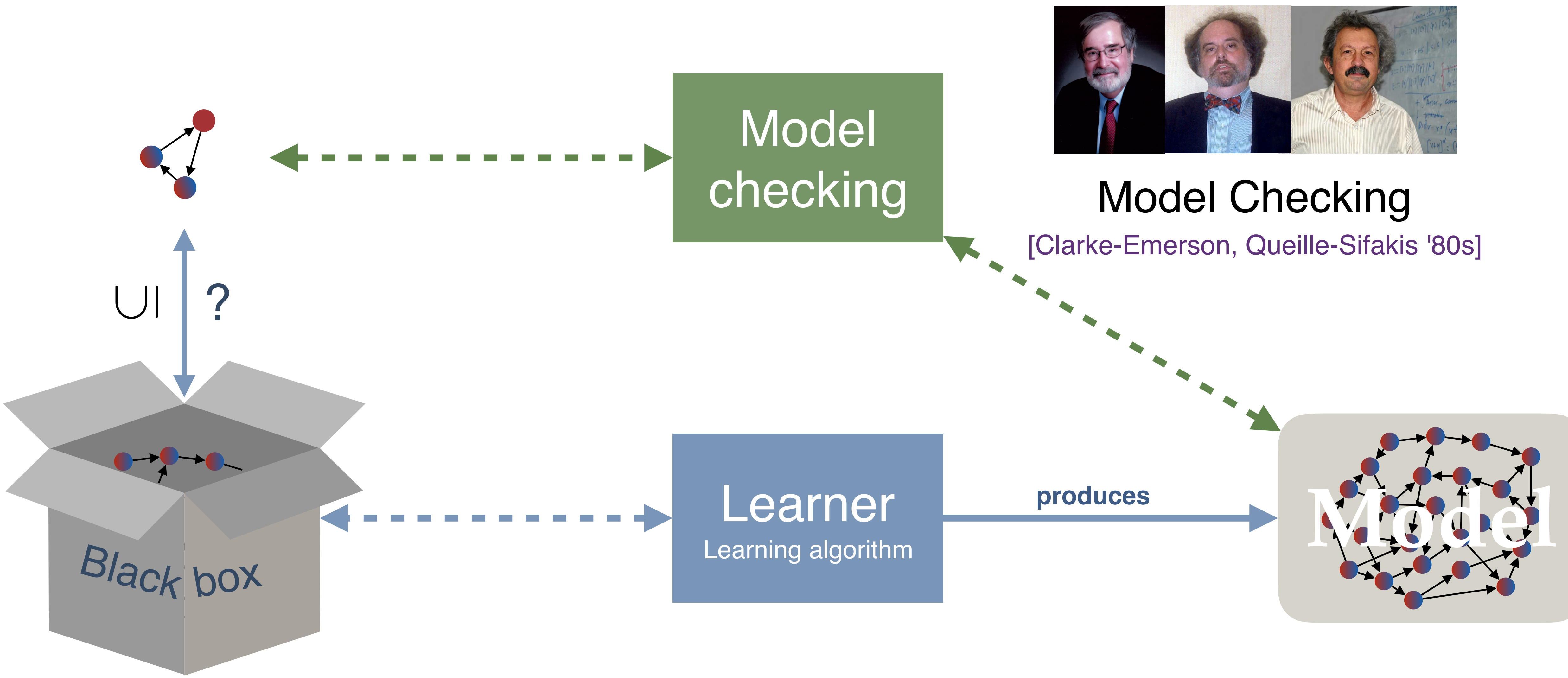
- legacy software
- code not open source
- third-party software
- embedded systems

Model Checking

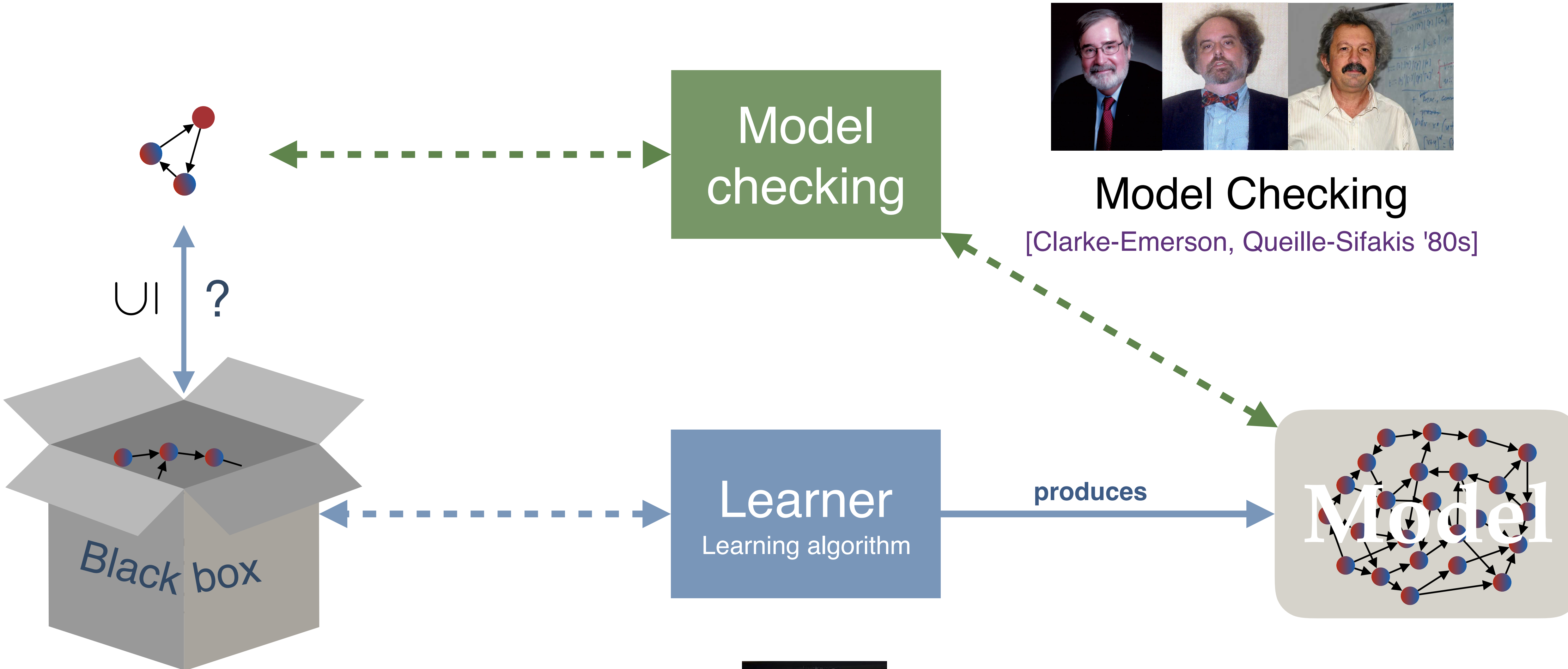
[Clarke-Emerson, Queille-Sifakis '80s]



- legacy software
- code not open source
- third-party software
- embedded systems

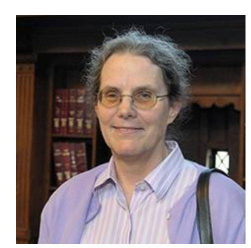


- legacy software
- code not open source
- third-party software
- embedded systems



Model Checking

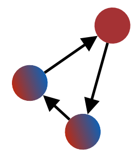
[Clarke-Emerson, Queille-Sifakis '80s]



Inference/Model Learning

[Moore '50s, Angluin '80s]

- legacy software
- code not open source
- third-party software
- embedded systems

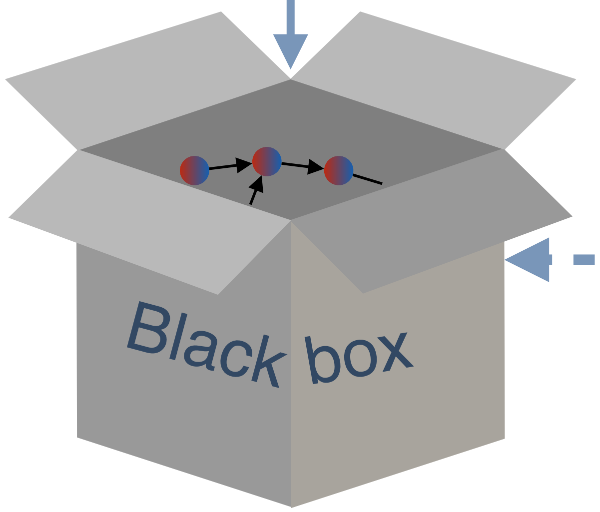


Model
checking

Model Checking

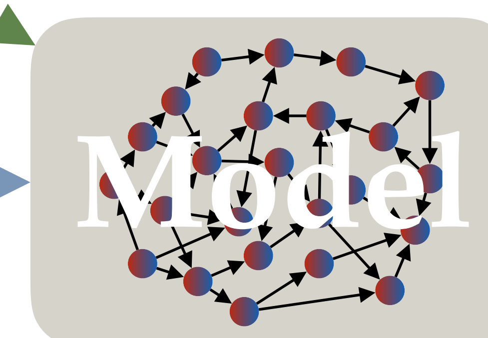
[Clarke-Emerson, Queille-Sifakis '80s]

⊔| ?

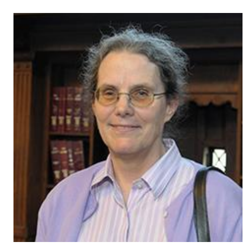


Learner
Learning algorithm

produces

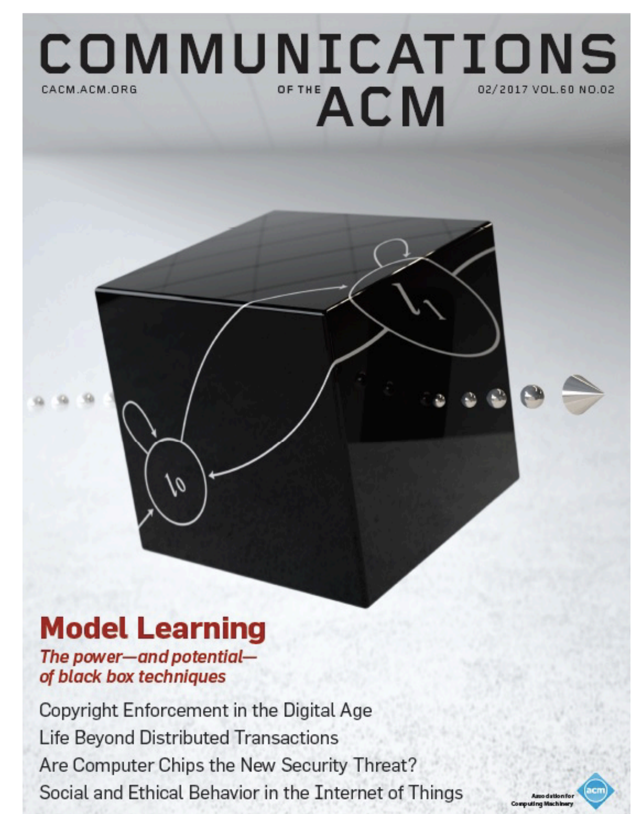


- legacy software
- code not open source
- third-party software
- embedded systems



Inference/Model Learning

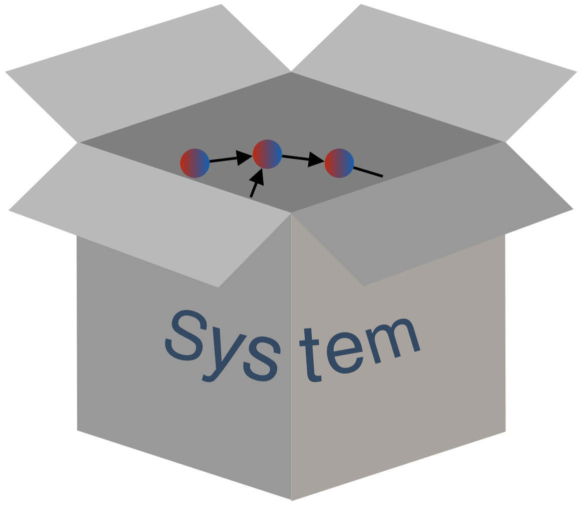
[Moore '50s, Angluin '80s]



[F. Vaandrager 2017]

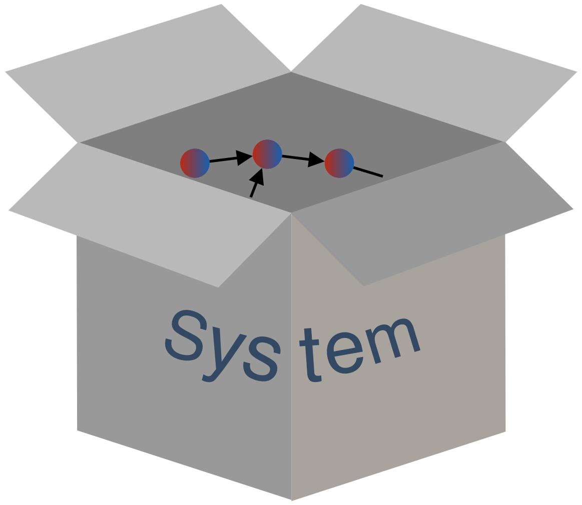
Angluin's L* Algorithm

[Angluin 1987]



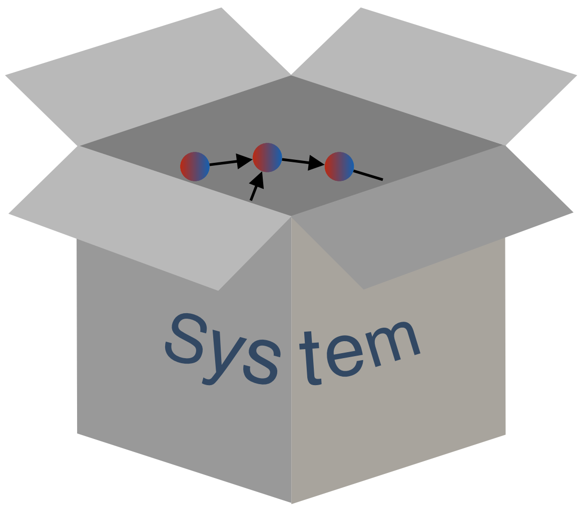
Angluin's L* Algorithm

[Angluin 1987]



$$L = (a+b)^* b (a+b)$$

[Angluin 1987]



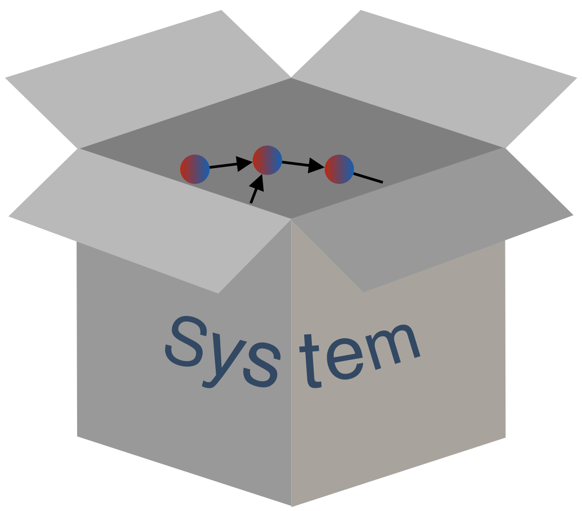
$$L = (a+b)^* b (a+b)$$

$$f_L : \Sigma^* \rightarrow \{0,1\}$$

$$f_L(w) = 1 \text{ iff } w \in L$$

Angluin's L* Algorithm

[Angluin 1987]



$$L = (a+b)^* b (a+b)$$

$$f_L : \Sigma^* \rightarrow \{0,1\}$$

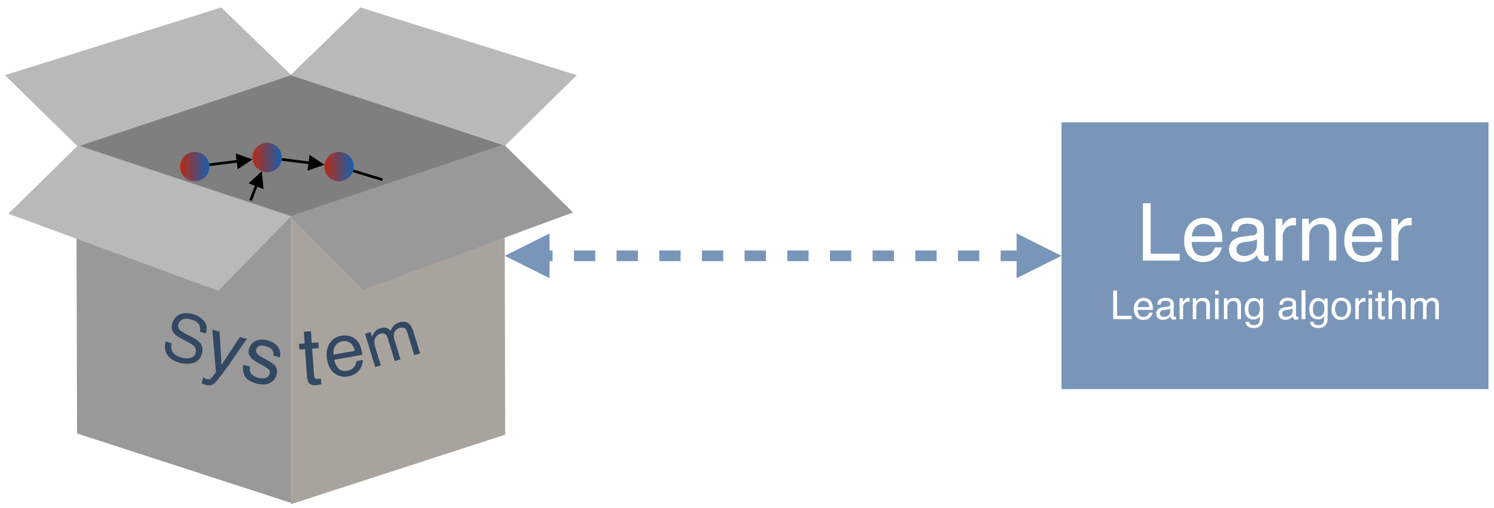
$$f_L(w) = 1 \text{ iff } w \in L$$

Learner

Learning algorithm

Angluin's L* Algorithm

[Angluin 1987]



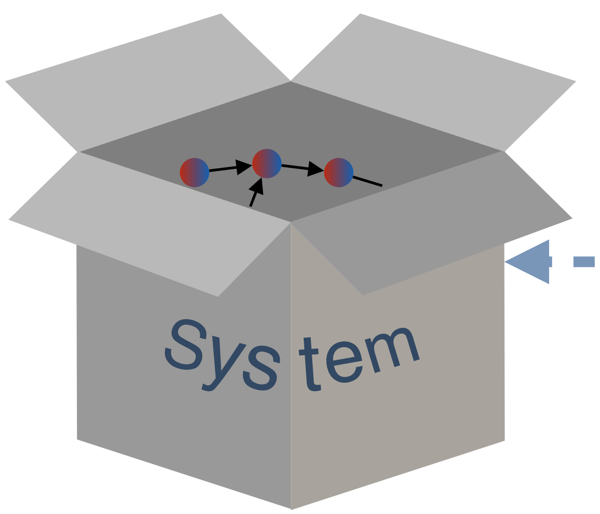
$$L = (a+b)^* b (a+b)$$

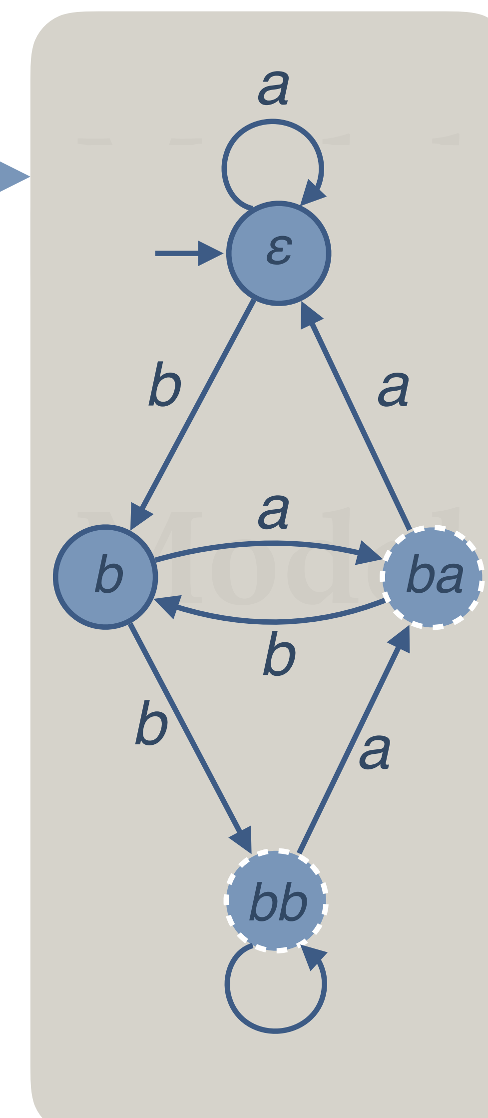
$$f_L : \Sigma^* \to \{0,1\}$$

$$f_L(w) = 1 \text{ iff } w \in L$$

Angluin's L* Algorithm

[Angluin 1987]





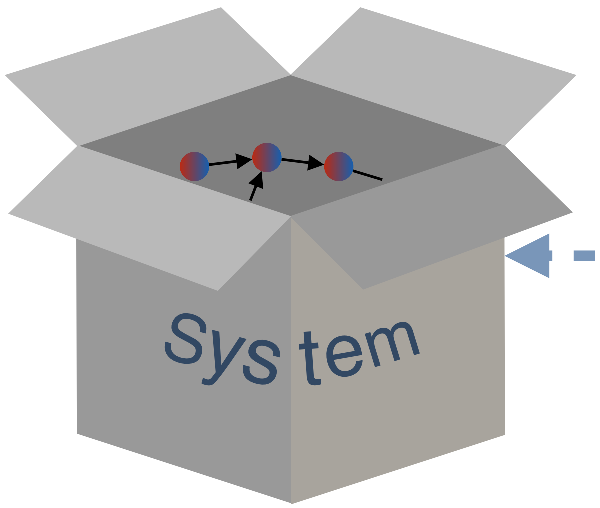
$$L = (a+b)^* b (a+b)$$

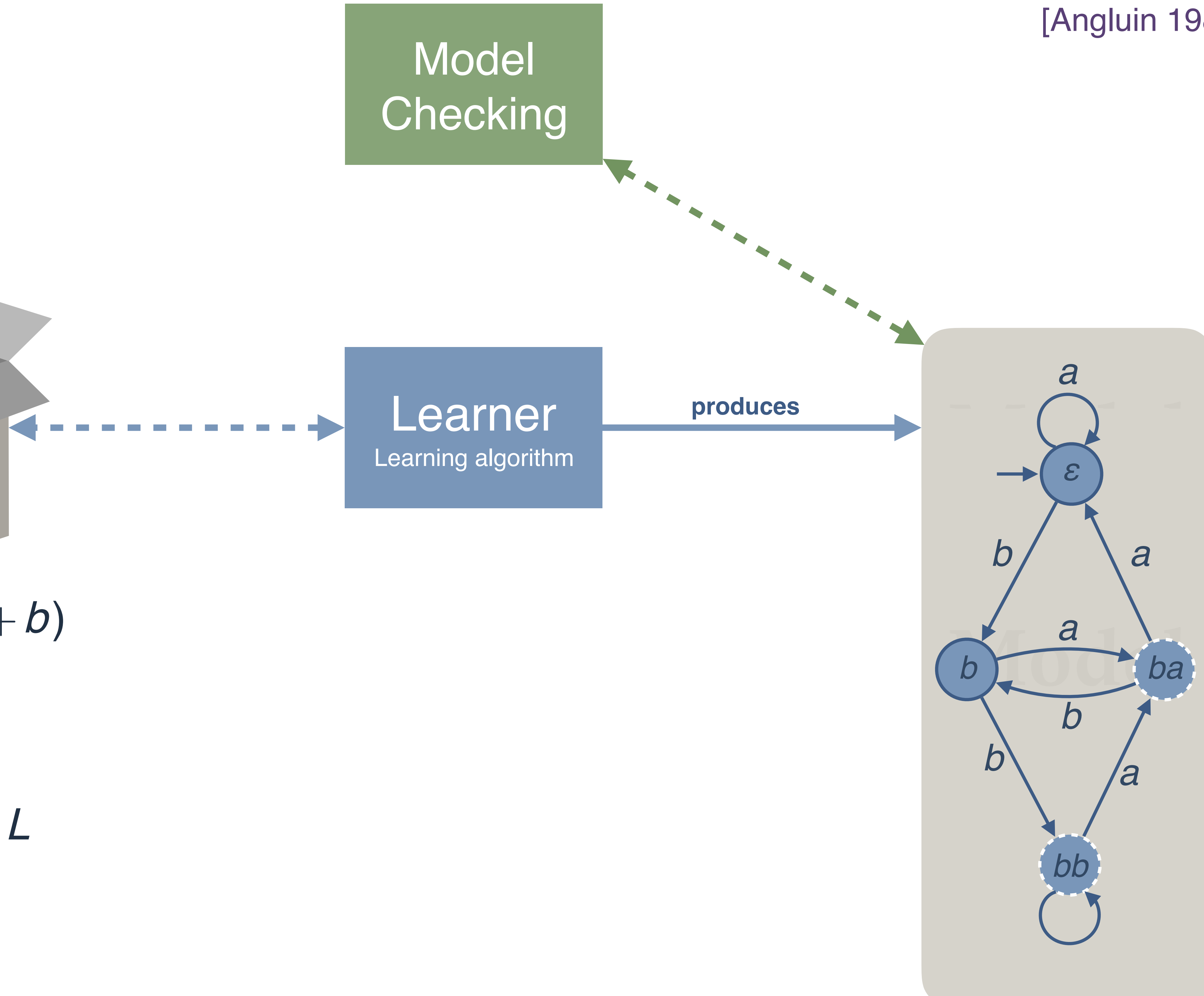
$$f_L : \Sigma^* \rightarrow \{0,1\}$$

$$f_L(w) = 1 \text{ iff } w \in L$$

Angluin's L* Algorithm

[Angluin 1987]





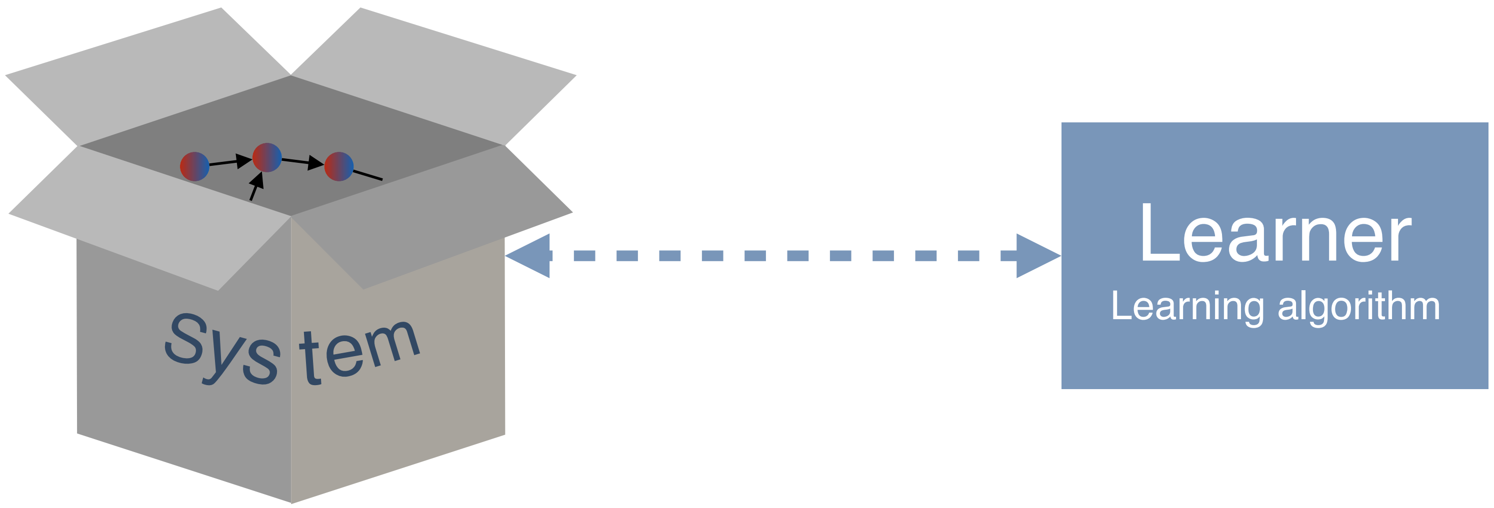
$$L = (a+b)^* b (a+b)$$

$$f_L : \Sigma^* \to \{0,1\}$$

$$f_L(w) = 1 \text{ iff } w \in L$$

Angluin's L* Algorithm

[Angluin 1987]



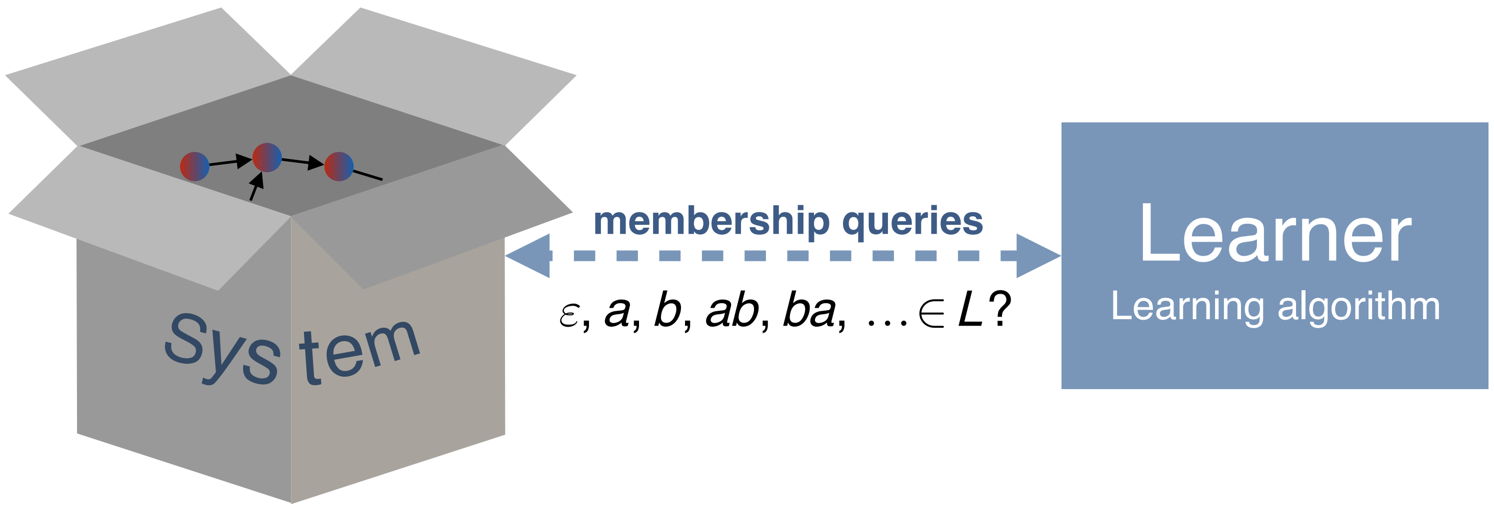
$$L = (a+b)^* b (a+b)$$

$$f_L : \Sigma^* \to \{0,1\}$$

$$f_L(w) = 1 \text{ iff } w \in L$$

Angluin's L* Algorithm

[Angluin 1987]



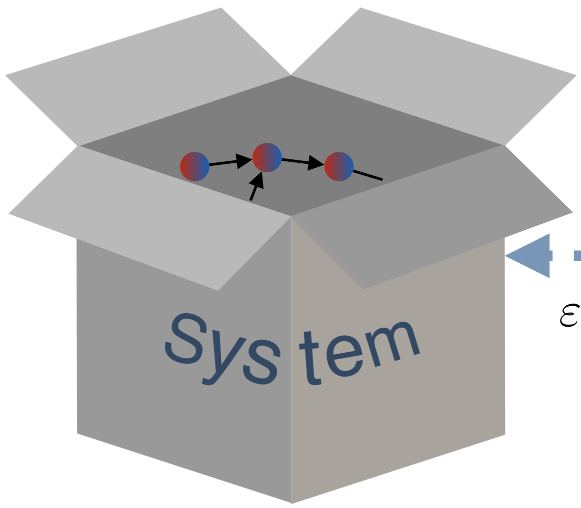
$$L = (a+b)^* b (a+b)$$

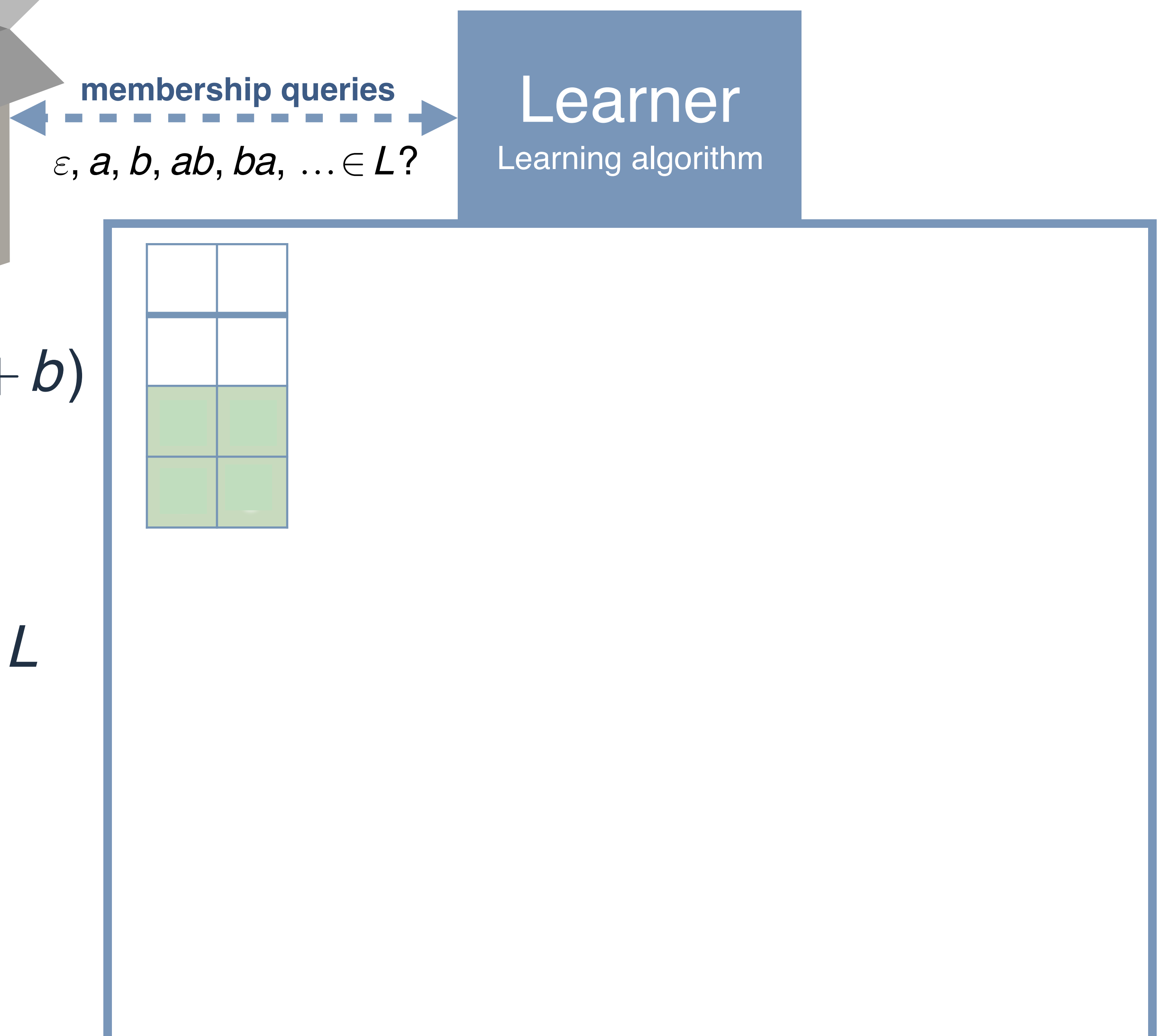
$$f_L : \Sigma^* \to \{0,1\}$$

$$f_L(w) = 1 \text{ iff } w \in L$$

Angluin's L* Algorithm

[Angluin 1987]





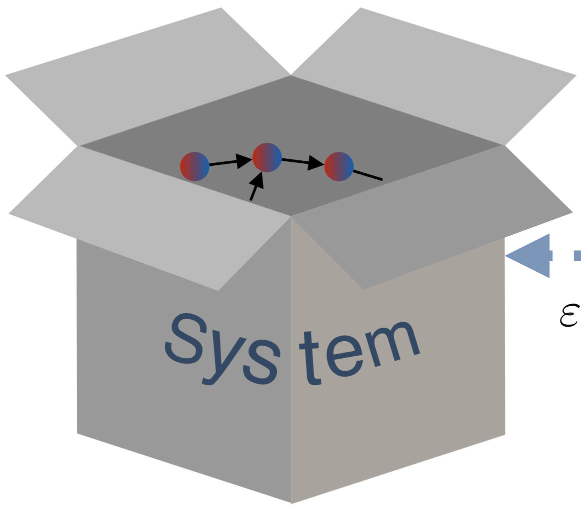
$$L = (a+b)^* b (a+b)$$

$$f_L : \Sigma^* \rightarrow \{0,1\}$$

$$f_L(w) = 1 \text{ iff } w \in L$$

Angluin's L* Algorithm

[Angluin 1987]



membership queries

ε, a, b, ab, ba, $\ldots \in L$?

Learner

Learning algorithm

	ε
ε	0

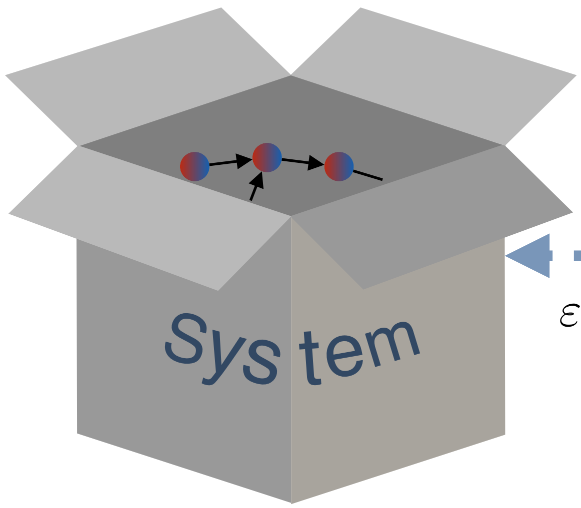
$$L = (a+b)^* b (a+b)$$

$$f_L : \Sigma^* \rightarrow \{0,1\}$$

$$f_L(w) = 1 \text{ iff } w \in L$$

Angluin's L* Algorithm

[Angluin 1987]



membership queries

ε, a, b, ab, ba, $\ldots \in L$?

Learner

Learning algorithm

$$L = (a+b)^* b (a+b)$$

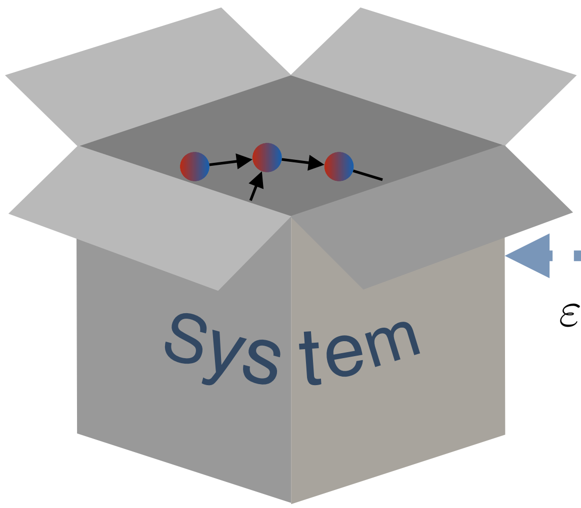
$$f_L : \Sigma^* \to \{0,1\}$$

$$f_L(w) = 1 \text{ iff } w \in L$$

	ε
ε	0
a	0
b	0

Angluin's L* Algorithm

[Angluin 1987]



membership queries

ε, a, b, ab, ba, $\ldots \in L$?

Learner

Learning algorithm

$$L = (a+b)^{*} b (a+b)$$

$$f_L : \Sigma^{*} \to \{0,1\}$$

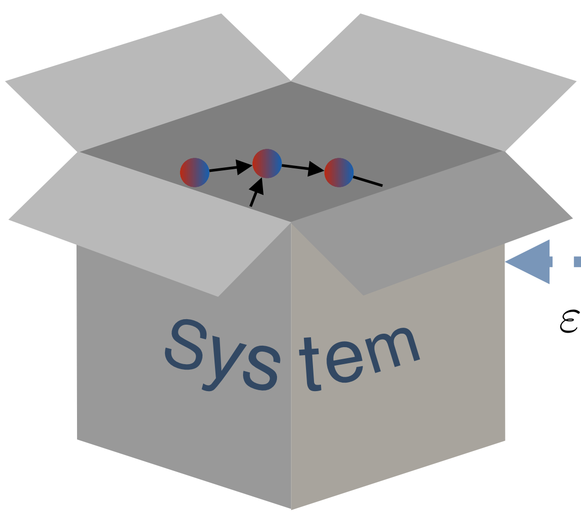
$$f_L(w) = 1 \text{ iff } w \in L$$

	ε
ε	0
a	0
b	0

upper part = states

Angluin's L* Algorithm

[Angluin 1987]



membership queries

$\varepsilon, a, b, ab, ba, \ldots \in L?$

Learner

Learning algorithm

$$L = (a+b)^* b (a+b)$$

$$f_L : \Sigma^* \to \{0,1\}$$

$$f_L(w) = 1 \text{ iff } w \in L$$

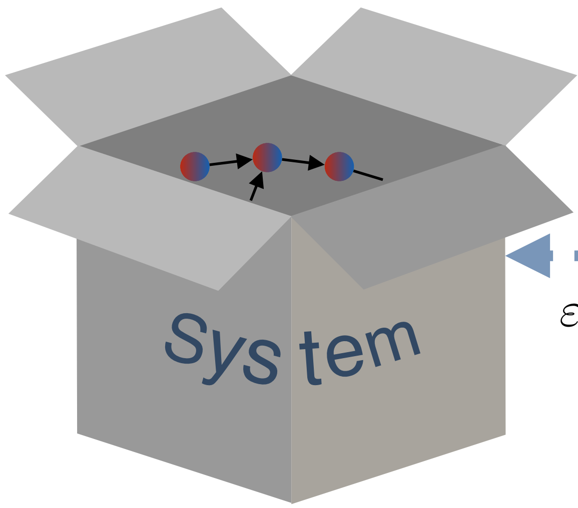
	ε
ε	0
a	0
b	0

upper part = states

lower part = transitions

Angluin's L* Algorithm

[Angluin 1987]



membership queries

ε, a, b, ab, ba, $\ldots \in L$?

Learner

Learning algorithm

	ε
ε	0
a	0
b	0

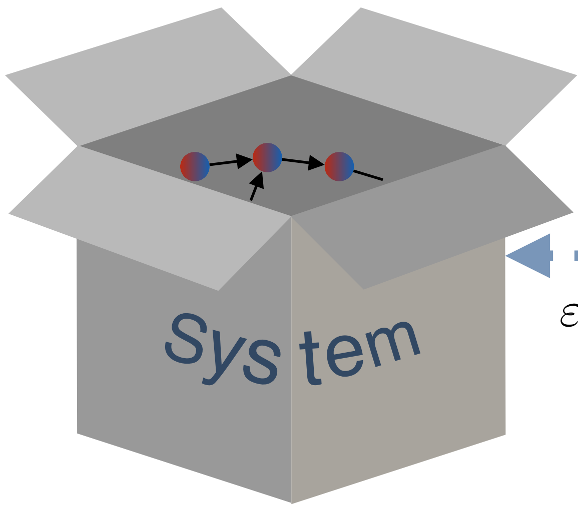
$$L = (a+b)^* b (a+b)$$

$$f_L : \Sigma^* \rightarrow \{0,1\}$$

$$f_L(w) = 1 \text{ iff } w \in L$$

Angluin's L* Algorithm

[Angluin 1987]



membership queries

$\varepsilon, a, b, ab, ba, \ldots \in L?$

Learner

Learning algorithm

	ε
ε	0
a	0
b	0

coherent ✔

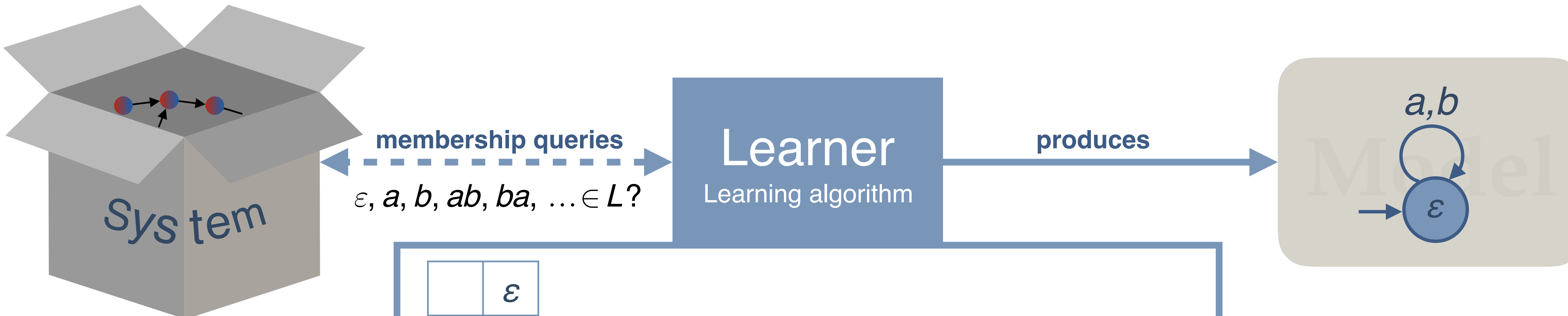
$$L = (a+b)^* b (a+b)$$

$$f_L : \Sigma^* \rightarrow \{0,1\}$$

$$f_L(w) = 1 \text{ iff } w \in L$$

Angluin's L* Algorithm

[Angluin 1987]



$$L = (a+b)^* b (a+b)$$

	ε
ε	0
a	0
b	0

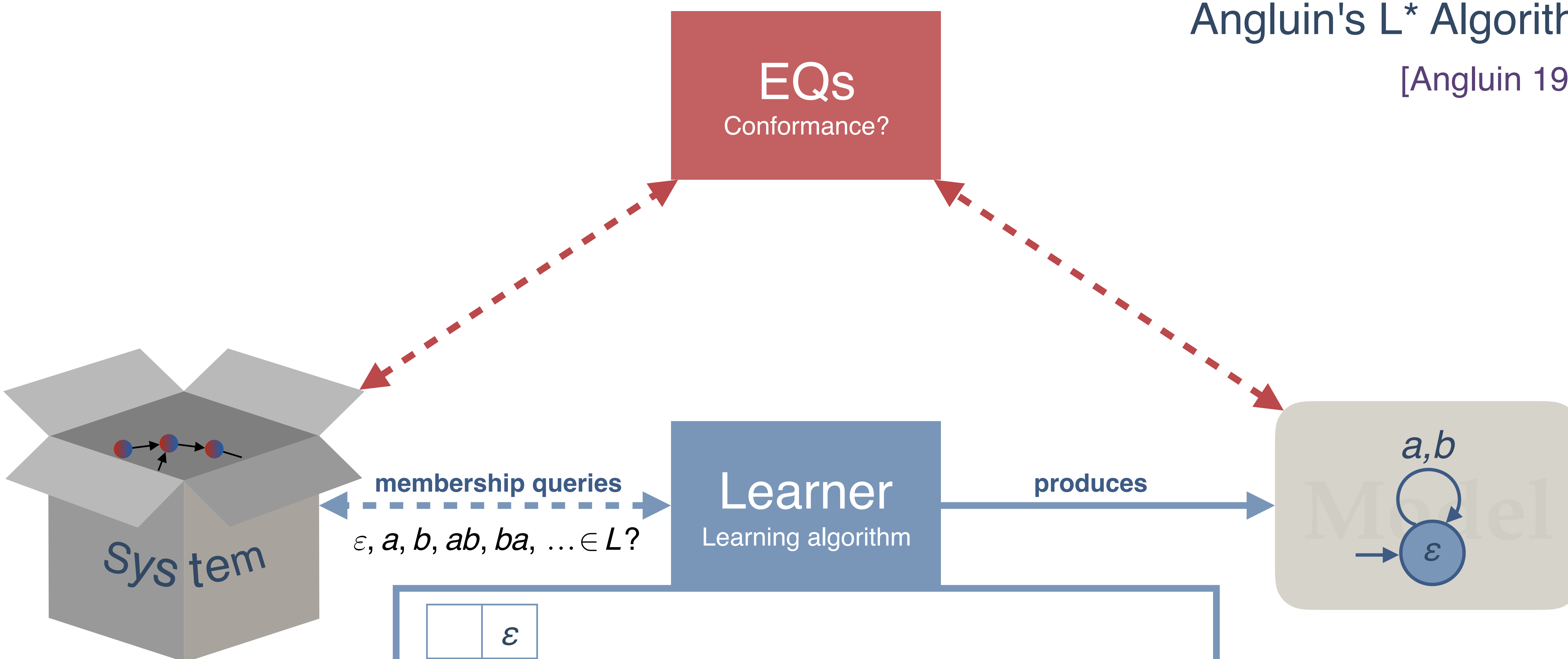
coherent ✔

$$f_L : \Sigma^* \to \{0,1\}$$

$$f_L(w) = 1 \text{ iff } w \in L$$

Angluin's L* Algorithm

[Angluin 1987]



$$L = (a+b)^* b (a+b)$$

$$f_L : \Sigma^* \to \{0,1\}$$

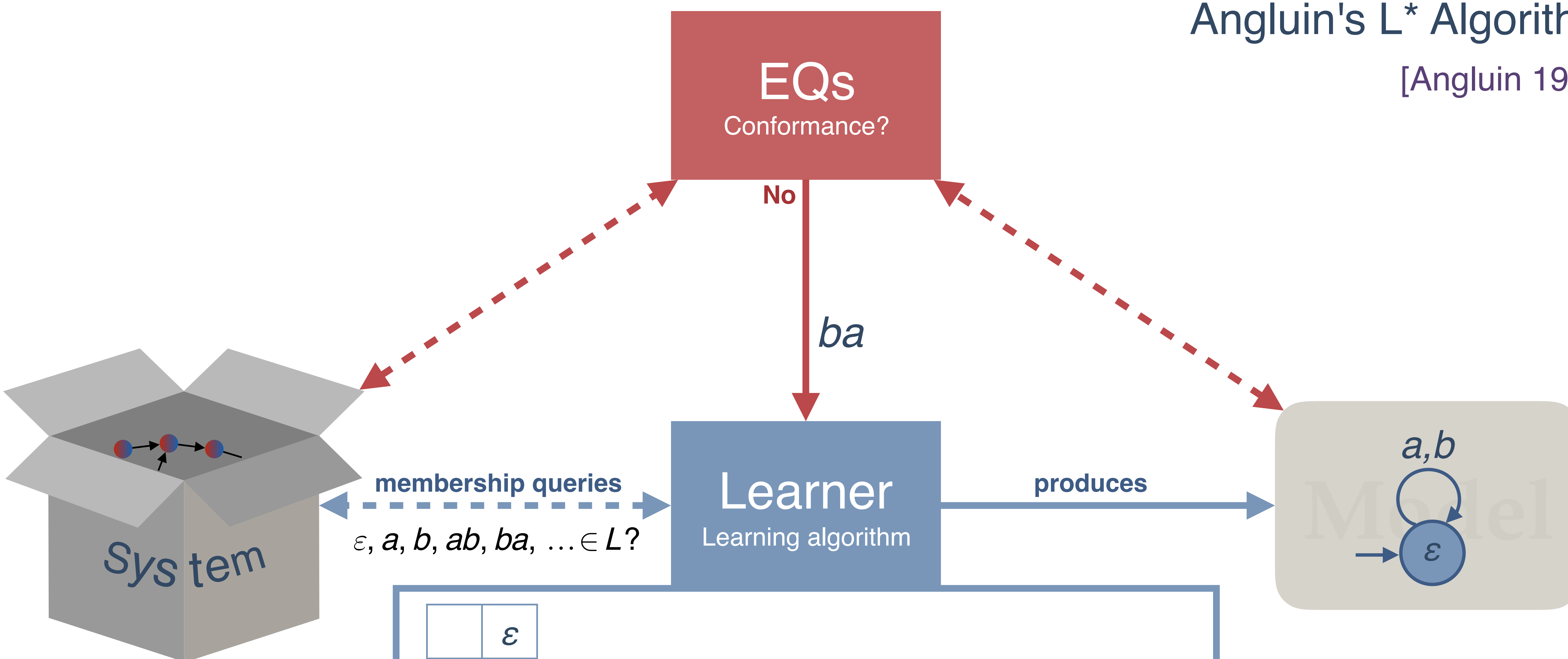
$$f_L(w) = 1 \text{ iff } w \in L$$

	ε
ε	0
a	0
b	0

coherent ✔

Angluin's L* Algorithm

[Angluin 1987]



$$L = (a+b)^* b (a+b)$$

$$f_L : \Sigma^* \to \{0,1\}$$

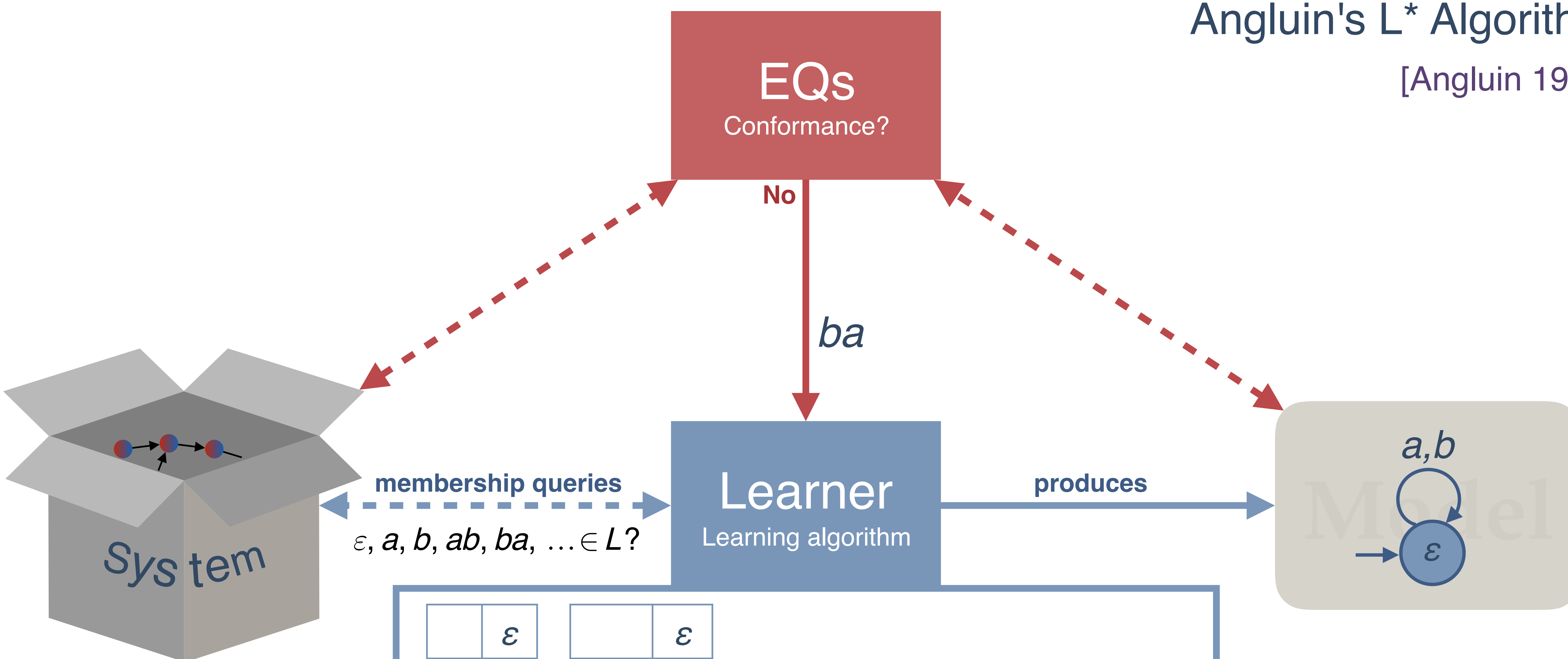
$$f_L(w) = 1 \text{ iff } w \in L$$

	ε
ε	0
a	0
b	0

coherent ✔

Angluin's L* Algorithm

[Angluin 1987]



$$L = (a+b)^* b (a+b)$$

$$f_L : \Sigma^* \rightarrow \{0,1\}$$

$$f_L(w) = 1 \text{ iff } w \in L$$

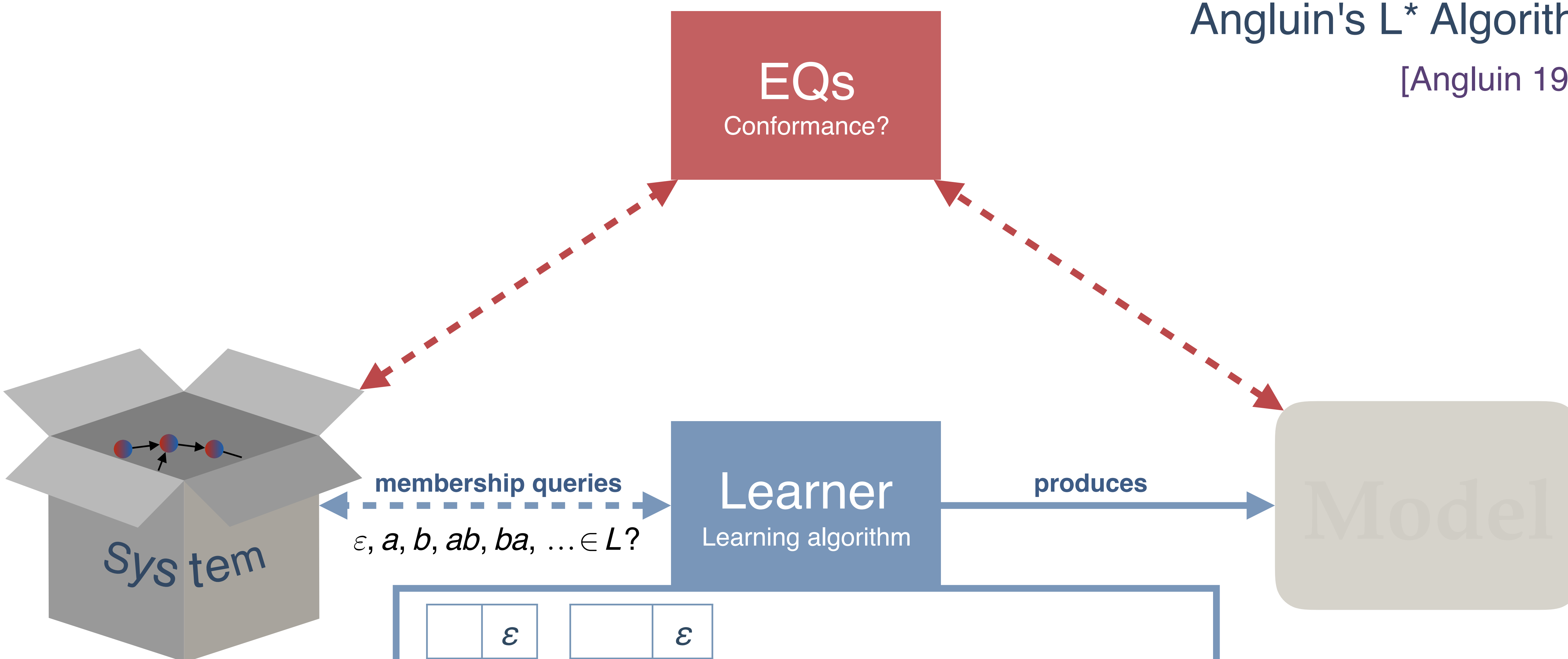
	ε
ε	0
a	0
b	0

coherent ✔

	ε
ε	0
b	0
ba	1
a	0
bb	1
baa	0
bab	0

Angluin's L* Algorithm

[Angluin 1987]



$$L = (a+b)^* b (a+b)$$

$$f_L : \Sigma^* \to \{0,1\}$$

$$f_L(w) = 1 \text{ iff } w \in L$$

	ε
ε	0
a	0
b	0

coherent ✔

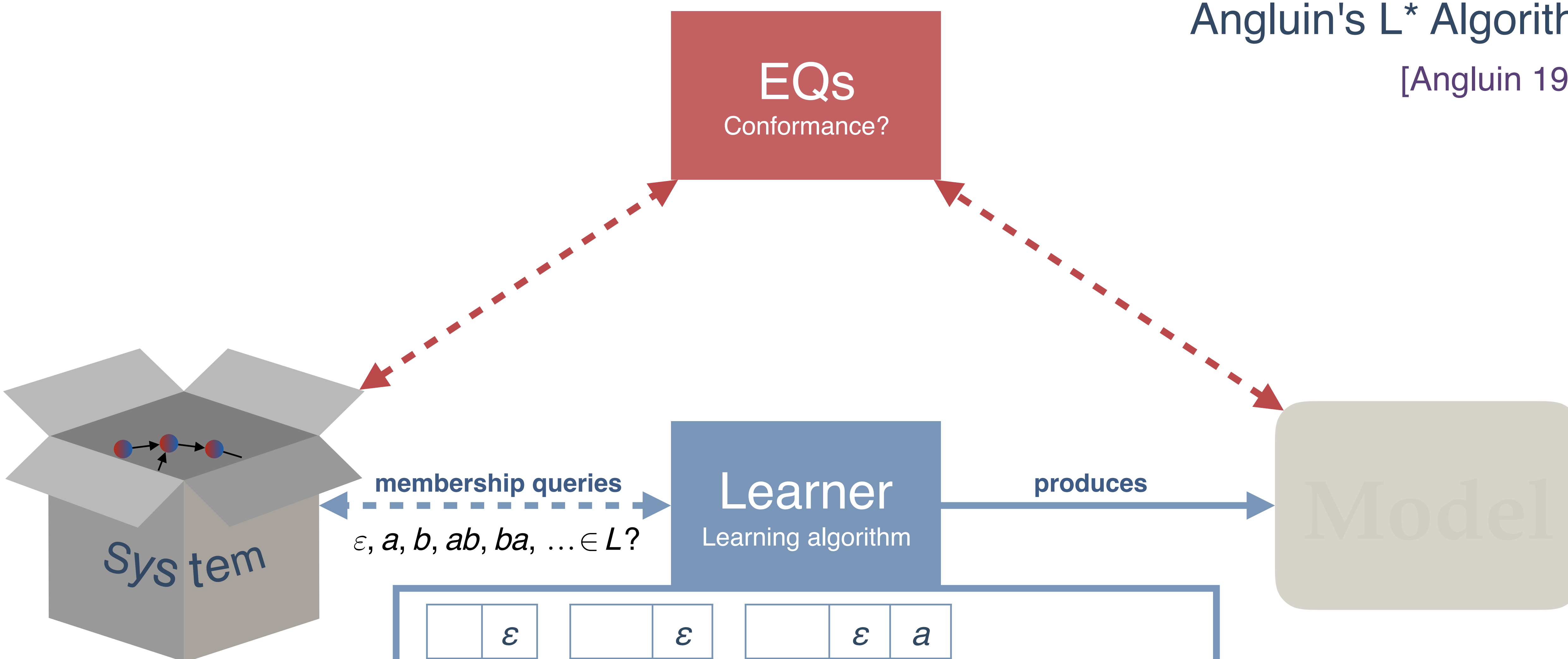
	ε
ε	0
b	0
ba	1
a	0
bb	1
baa	0
bab	0

consistent ✘

a a

Angluin's L* Algorithm

[Angluin 1987]



$$L = (a+b)^* b (a+b)$$

$$f_L : \Sigma^* \to \{0,1\}$$

$$f_L(w) = 1 \text{ iff } w \in L$$

	ε
ε	0
a	0
b	0

coherent ✔

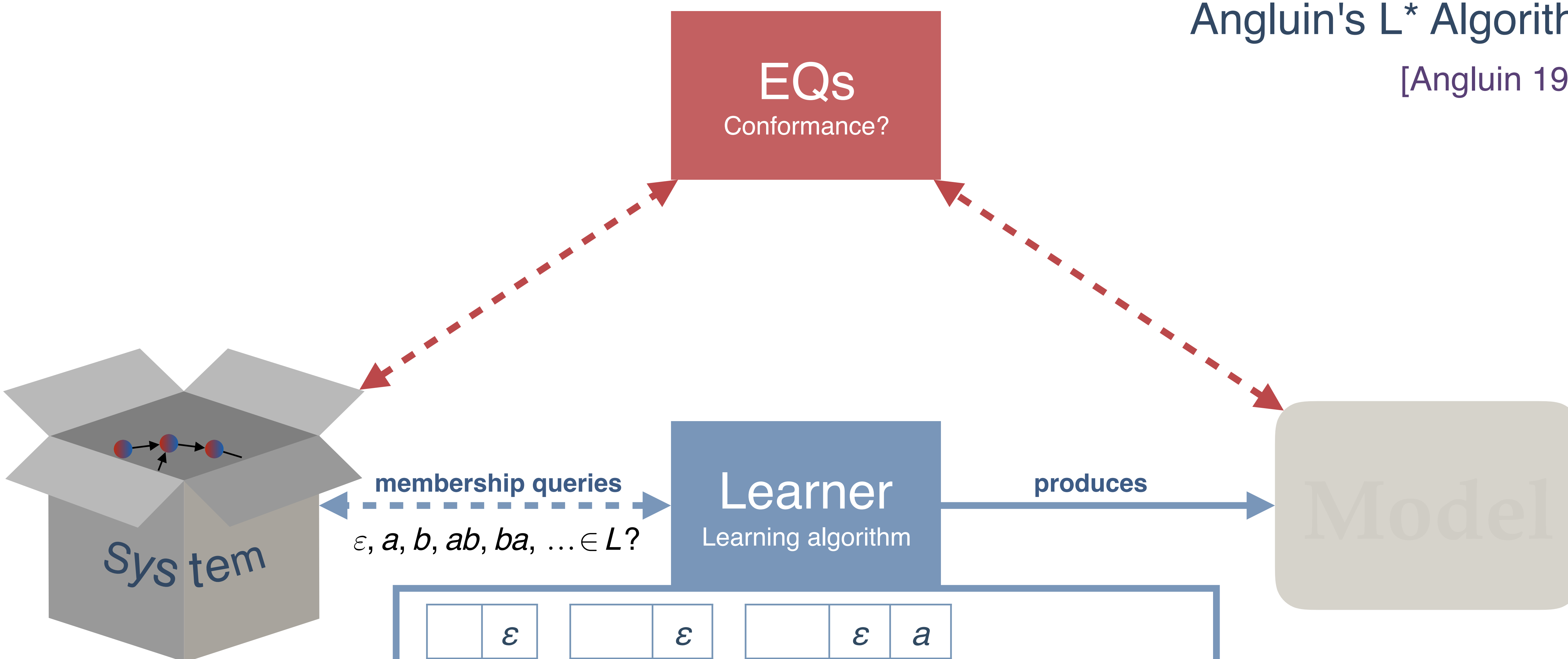
	ε
ε	0
b	0
ba	1
a	0
bb	1
baa	0
bab	0

consistent ✘

	ε	a
ε	0	0
b	0	1
ba	1	0
a	0	0
bb	1	1
baa	0	0
bab	0	1

Angluin's L* Algorithm

[Angluin 1987]



$$L = (a+b)^* b (a+b)$$

$$f_L : \Sigma^* \to \{0,1\}$$

$$f_L(w) = 1 \text{ iff } w \in L$$

	ε
ε	0
a	0
b	0

coherent ✔

	ε
ε	0
b	0
ba	1
a	0
bb	1
baa	0
bab	0

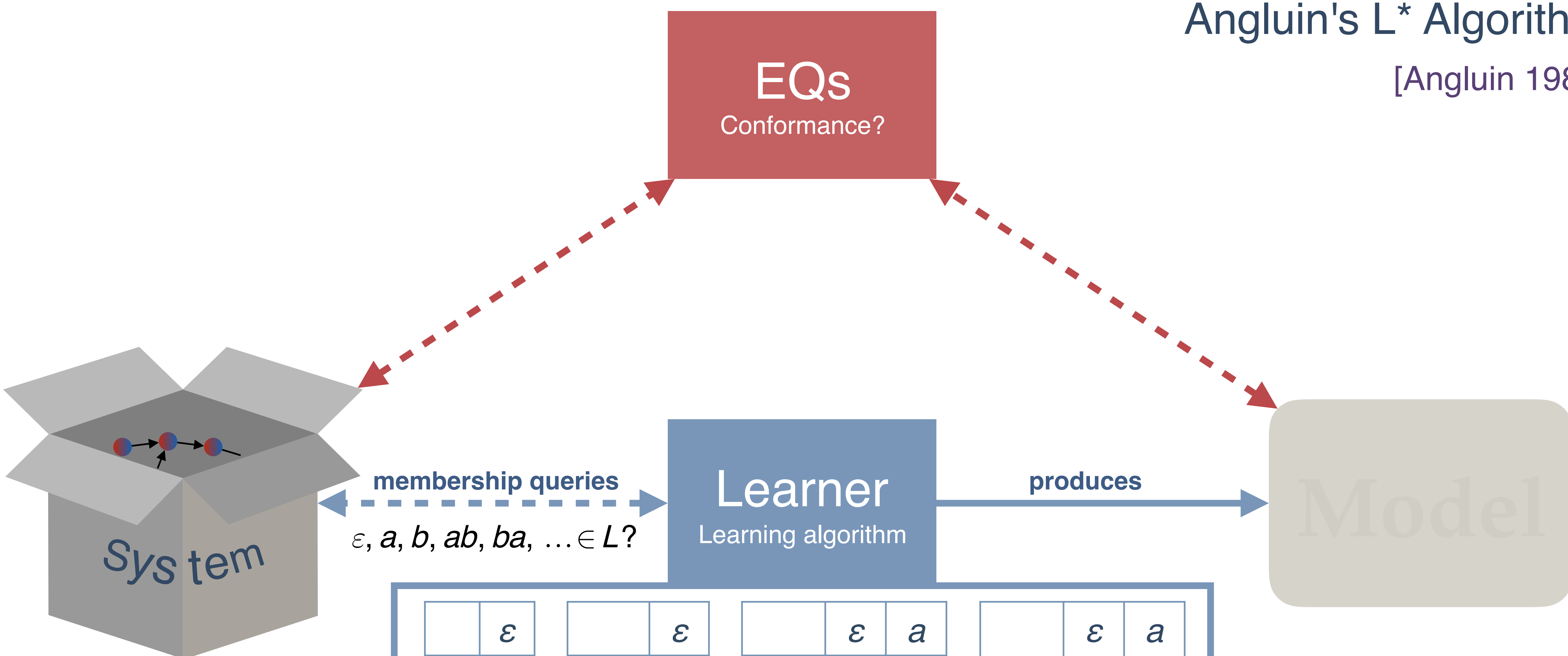
consistent ✘

	ε	a
ε	0	0
b	0	1
ba	1	0
a	0	0
bb	1	1
baa	0	0
bab	0	1

closed ✘

Angluin's L* Algorithm

[Angluin 1987]



$$L = (a+b)^* b (a+b)$$

$$f_L : \Sigma^* \to \{0,1\}$$

$$f_L(w) = 1 \ \text{iff} \ w \in L$$

	ε
ε	0
a	0
b	0

coherent ✔

	ε
ε	0
b	0
ba	1
a	0
bb	1
baa	0
bab	0

consistent ✘

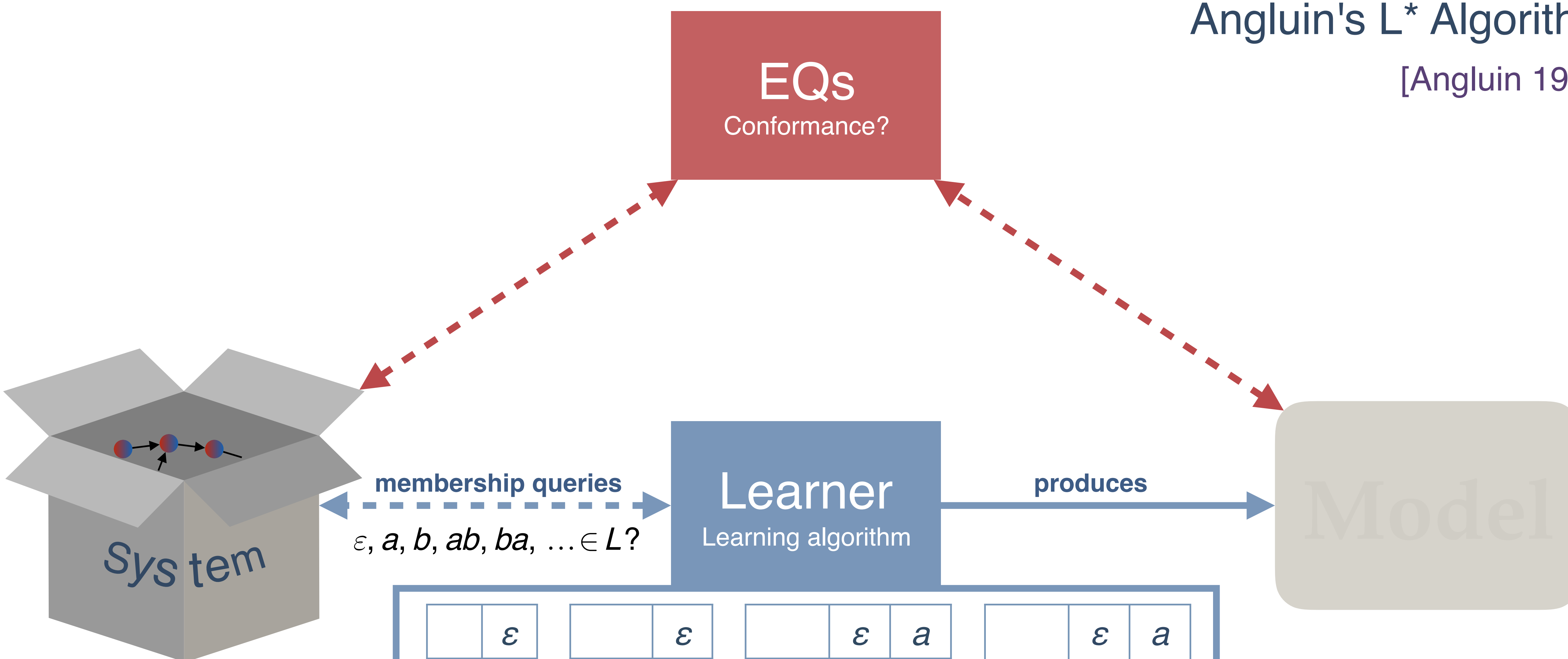
	ε	a
ε	0	0
b	0	1
ba	1	0
a	0	0
bb	1	1
baa	0	0
bab	0	1

closed ✘

	ε	a
ε	0	0
b	0	1
ba	1	0
bb	1	1
a	0	0
baa	0	0
bab	0	1
bba	1	0
bbb	1	1

Angluin's L* Algorithm

[Angluin 1987]



$$L = (a+b)^* b (a+b)$$

$$f_L : \Sigma^* \rightarrow \{0,1\}$$

$$f_L(w) = 1 \text{ iff } w \in L$$

	ε
ε	0
a	0
b	0

coherent ✔

	ε
ε	0
b	0
ba	1
a	0
bb	1
baa	0
bab	0

consistent ✘

	ε	a
ε	0	0
b	0	1
ba	1	0
a	0	0
bb	1	1
baa	0	0
bab	0	1

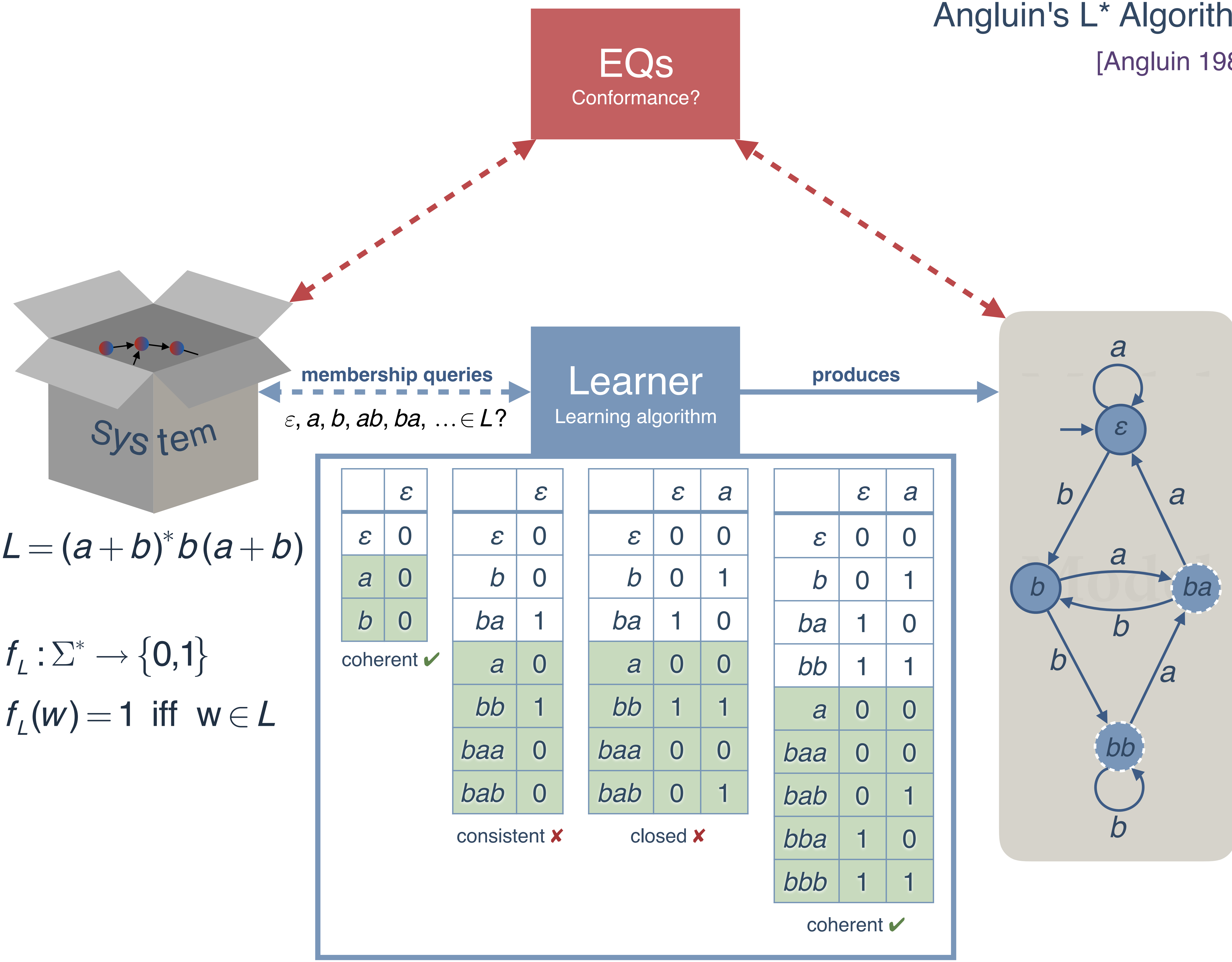
closed ✘

	ε	a
ε	0	0
b	0	1
ba	1	0
bb	1	1
a	0	0
baa	0	0
bab	0	1
bba	1	0
bbb	1	1

coherent ✔

Angluin's L* Algorithm

[Angluin 1987]



$$L = (a+b)^* b (a+b)$$

$$f_L : \Sigma^* \to \{0,1\}$$

$$f_L(w) = 1 \text{ iff } w \in L$$

	ε
ε	0
a	0
b	0

coherent ✔

	ε
ε	0
b	0
ba	1
a	0
bb	1
baa	0
bab	0

consistent ✘

	ε	a
ε	0	0
b	0	1
ba	1	0
a	0	0
bb	1	1
baa	0	0
bab	0	1

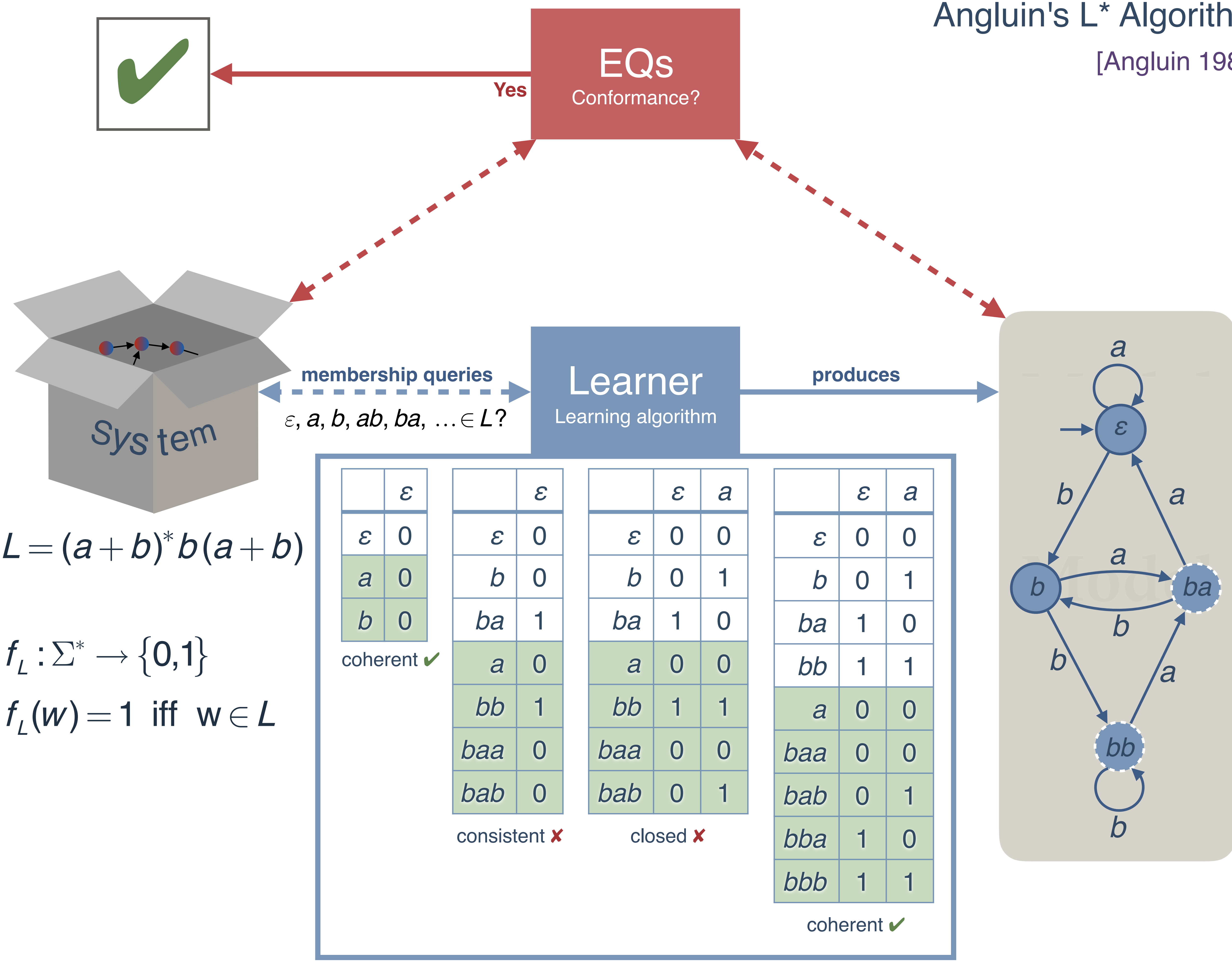
closed ✘

	ε	a
ε	0	0
b	0	1
ba	1	0
bb	1	1
a	0	0
baa	0	0
bab	0	1
bba	1	0
bbb	1	1

coherent ✔

Angluin's L* Algorithm

[Angluin 1987]



$$L = (a+b)^* b (a+b)$$

$$f_L : \Sigma^* \to \{0,1\}$$

$$f_L(w) = 1 \text{ iff } w \in L$$

	ε
ε	0
a	0
b	0

coherent ✔

	ε
ε	0
b	0
ba	1
a	0
bb	1
baa	0
bab	0

consistent ✘

	ε	a
ε	0	0
b	0	1
ba	1	0
a	0	0
bb	1	1
baa	0	0
bab	0	1

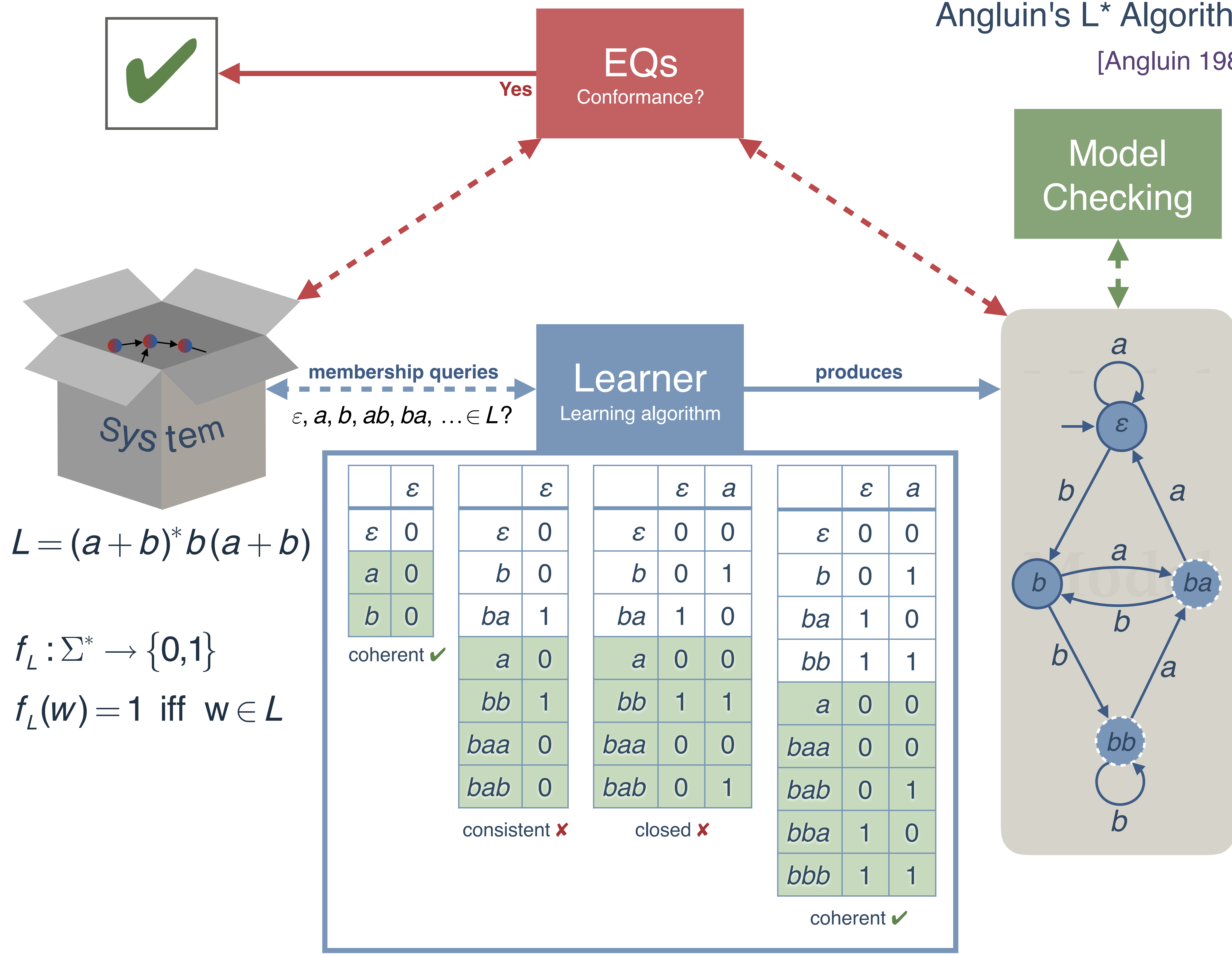
closed ✘

	ε	a
ε	0	0
b	0	1
ba	1	0
bb	1	1
a	0	0
baa	0	0
bab	0	1
bba	1	0
bbb	1	1

coherent ✔

Angluin's L* Algorithm

[Angluin 1987]



$$L = (a+b)^* b (a+b)$$

$$f_L : \Sigma^* \to \{0,1\}$$

$$f_L(w) = 1 \text{ iff } w \in L$$

	ε
ε	0
a	0
b	0

coherent ✔

	ε
ε	0
b	0
ba	1
a	0
bb	1
baa	0
bab	0

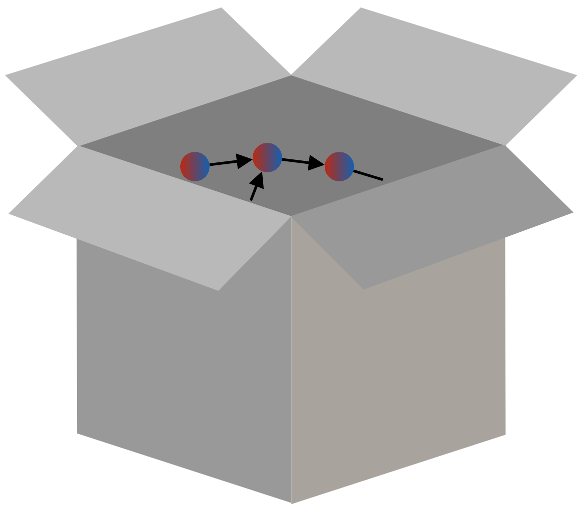
consistent ✘

	ε	a
ε	0	0
b	0	1
ba	1	0
a	0	0
bb	1	1
baa	0	0
bab	0	1

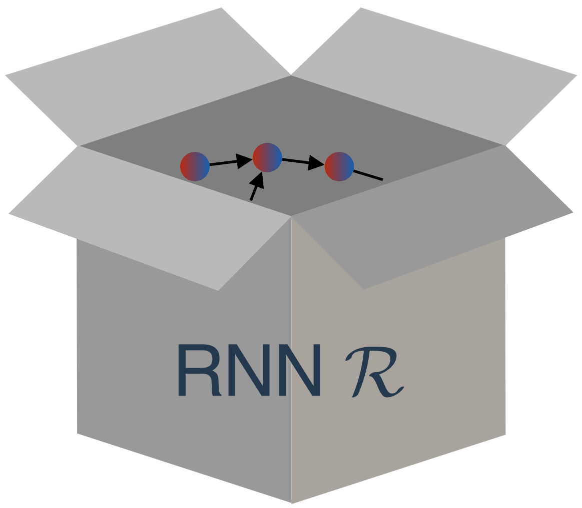
closed ✘

	ε	a
ε	0	0
b	0	1
ba	1	0
bb	1	1
a	0	0
baa	0	0
bab	0	1
bba	1	0
bbb	1	1

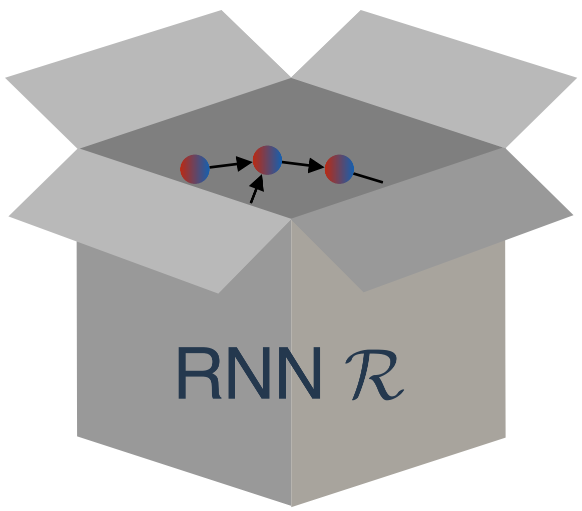
coherent ✔



- legacy software
- code not open source
- third-party software
- embedded systems

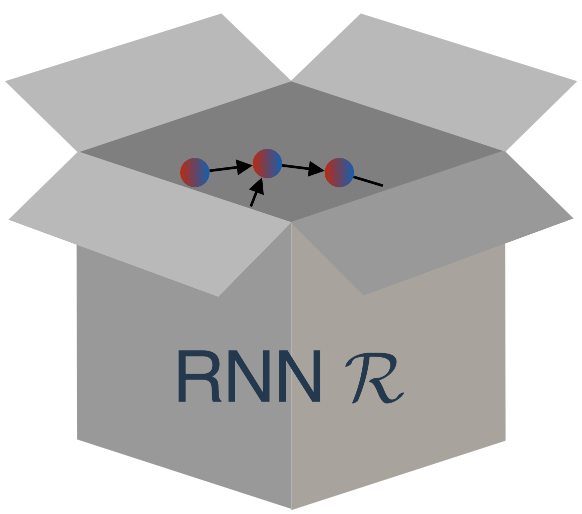


- legacy software
- code not open source
- third-party software
- embedded systems
- recurrent neural networks (RNNs)

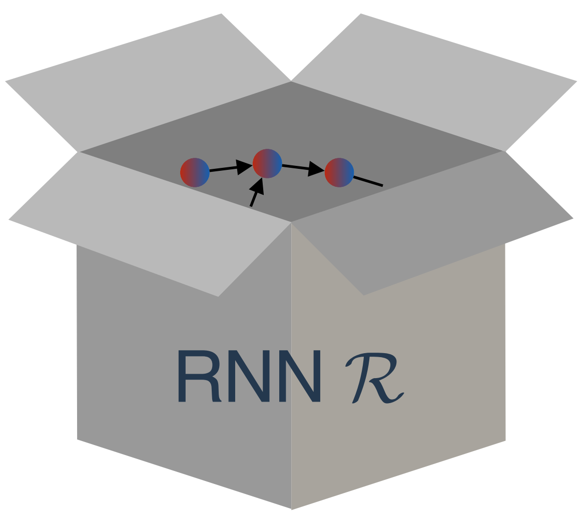


- legacy software
- code not open source
- third-party software
- embedded systems
- recurrent neural networks (RNNs)

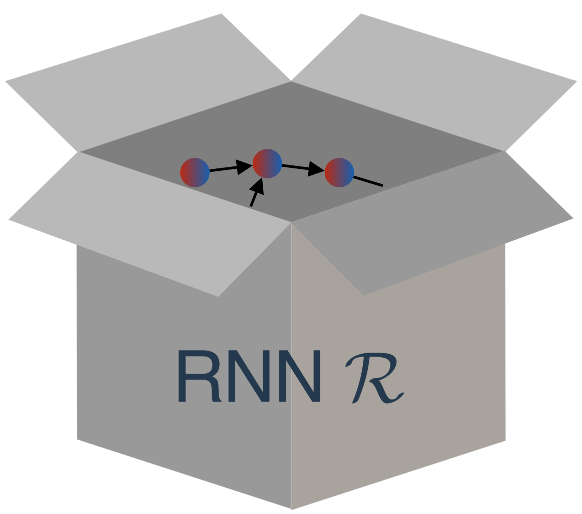
(approach works for binary classifiers)



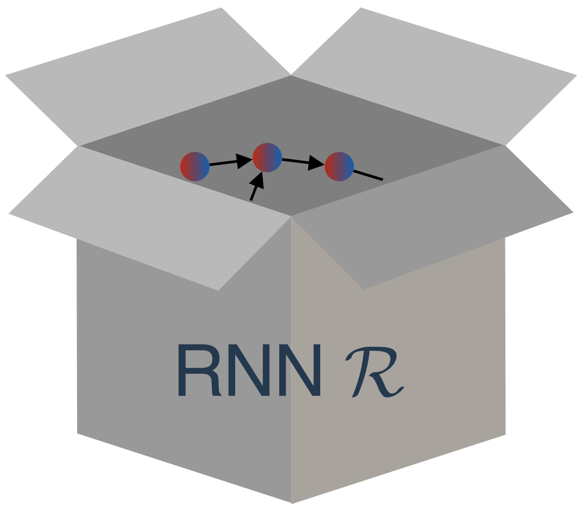
$$f : \underbrace{\mathbb{R}^k}_{\text{state}} \times \underbrace{\mathbb{R}^\ell}_{\text{input}} \longrightarrow \underbrace{\mathbb{R}^k}_{\text{state}}$$



$$f : \underbrace{\mathbb{R}^k}_{\text{state}} \times \underbrace{\Sigma}_{\text{input}} \longrightarrow \underbrace{\mathbb{R}^k}_{\text{state}}$$

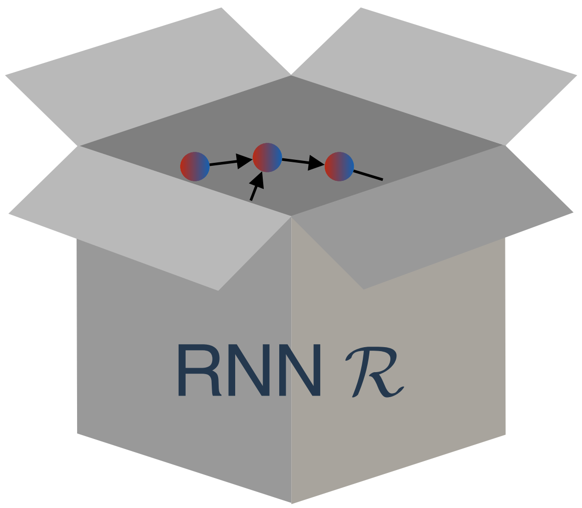


$$f : \underbrace{\mathbb{R}^k}_{\text{state}} \times \underbrace{\Sigma^*}_{\text{input}} \longrightarrow \underbrace{\mathbb{R}^k}_{\text{state}}$$



$$f : \underbrace{\mathbb{R}^k}_{\text{state}} \times \underbrace{\Sigma^*}_{\text{input}} \longrightarrow \underbrace{\mathbb{R}^k}_{\text{state}}$$

$$g : \underbrace{\mathbb{R}^k}_{\text{state}} \longrightarrow \{0,1\}$$



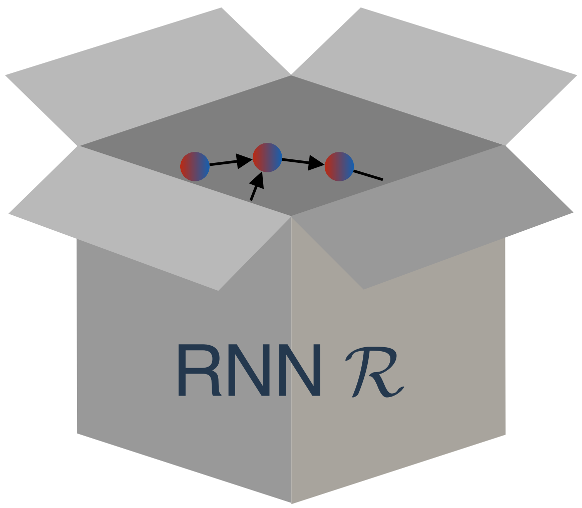
$$f : \underbrace{\mathbb{R}^k}_{\text{state}} \times \underbrace{\Sigma^*}_{\text{input}} \longrightarrow \underbrace{\mathbb{R}^k}_{\text{state}}$$

$$g : \underbrace{\mathbb{R}^k}_{\text{state}} \longrightarrow \{0,1\}$$

$\rightsquigarrow$ language acceptor

$$L(\mathcal{R}) = \{w \in \Sigma^* \mid g(f(init, w)) = 1\}$$

What does correctness for RNNs mean?



$$f : \underbrace{\mathbb{R}^k}_{\text{state}} \times \underbrace{\Sigma^*}_{\text{input}} \longrightarrow \underbrace{\mathbb{R}^k}_{\text{state}}$$

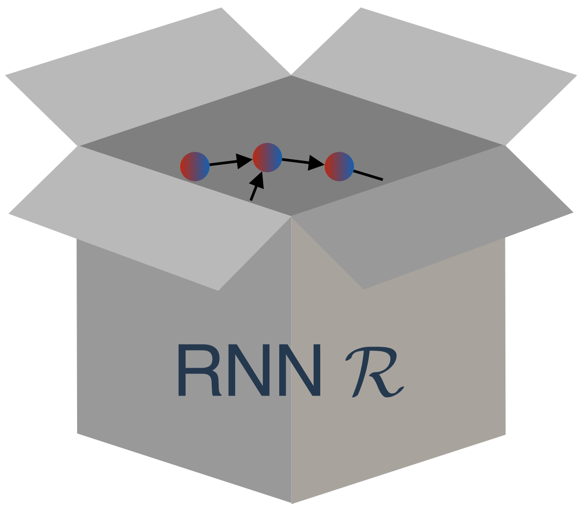
$$g : \underbrace{\mathbb{R}^k}_{\text{state}} \longrightarrow \{0,1\}$$

$\rightsquigarrow$ language acceptor

$$L(\mathcal{R}) = \{ w \in \Sigma^* \mid g(f(init, w)) = 1 \}$$

What does correctness for RNNs mean?

Specify correctness using finite automata $\mathcal{E}$ over Σ



$$f : \underbrace{\mathbb{R}^k}_{\text{state}} \times \underbrace{\Sigma^*}_{\text{input}} \longrightarrow \underbrace{\mathbb{R}^k}_{\text{state}}$$

$$g : \underbrace{\mathbb{R}^k}_{\text{state}} \longrightarrow \{0,1\}$$

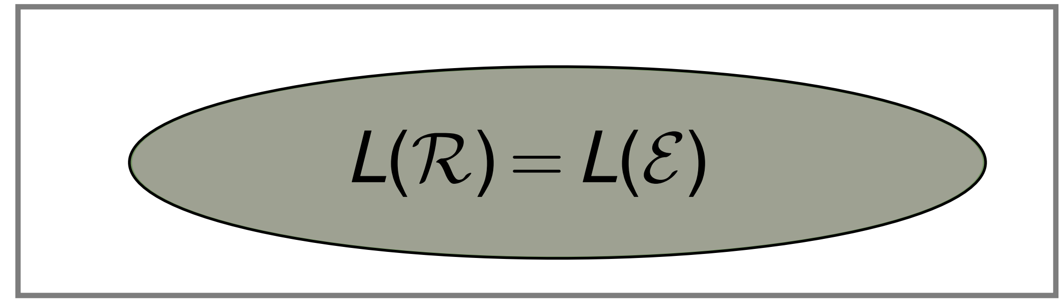
$\rightsquigarrow$ language acceptor

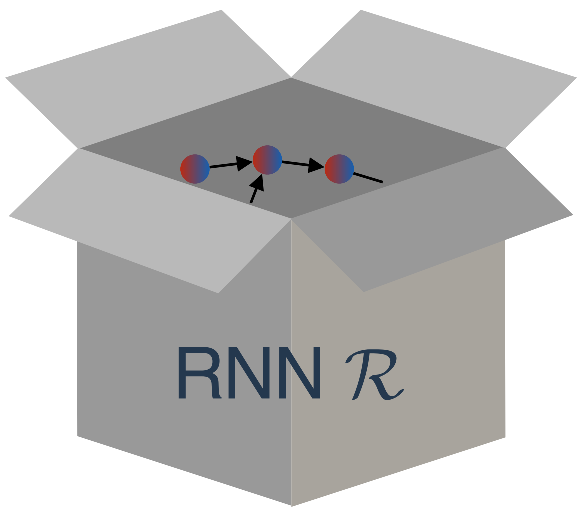
$$L(\mathcal{R}) = \{ w \in \Sigma^* \mid g(f(init, w)) = 1 \}$$

What does correctness for RNNs mean?

Specify correctness using finite automata $\mathcal{E}$ over Σ

$$L(\mathcal{R}) = L(\mathcal{E})$$





$$f : \underbrace{\mathbb{R}^k}_{\text{state}} \times \underbrace{\Sigma^*}_{\text{input}} \longrightarrow \underbrace{\mathbb{R}^k}_{\text{state}}$$

$$g : \underbrace{\mathbb{R}^k}_{\text{state}} \longrightarrow \{0,1\}$$

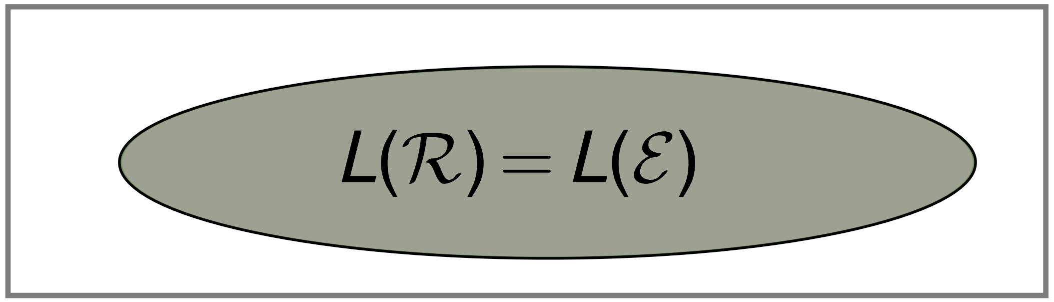
$\rightsquigarrow$ language acceptor

$$L(\mathcal{R}) = \{ w \in \Sigma^* \mid g(f(init, w)) = 1 \}$$

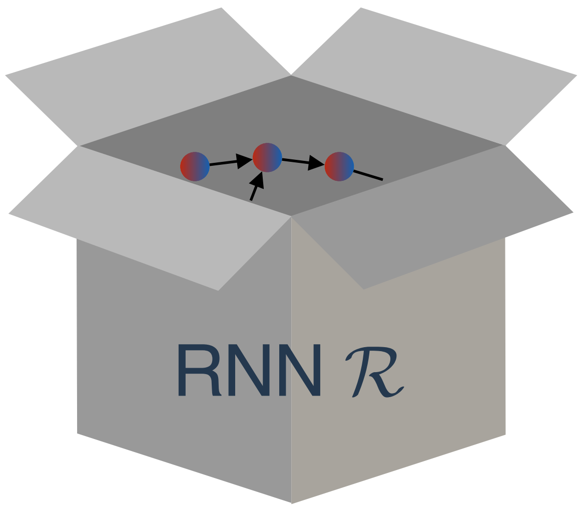
What does correctness for RNNs mean?

Specify correctness using finite automata $\mathcal{E}$ over Σ

$$L(\mathcal{R}) = L(\mathcal{E})$$



$\rightsquigarrow$ $L(\mathcal{R})$ is a regular language



$$f : \underbrace{\mathbb{R}^k}_{\text{state}} \times \underbrace{\Sigma^*}_{\text{input}} \longrightarrow \underbrace{\mathbb{R}^k}_{\text{state}}$$

$$g : \underbrace{\mathbb{R}^k}_{\text{state}} \longrightarrow \{0,1\}$$

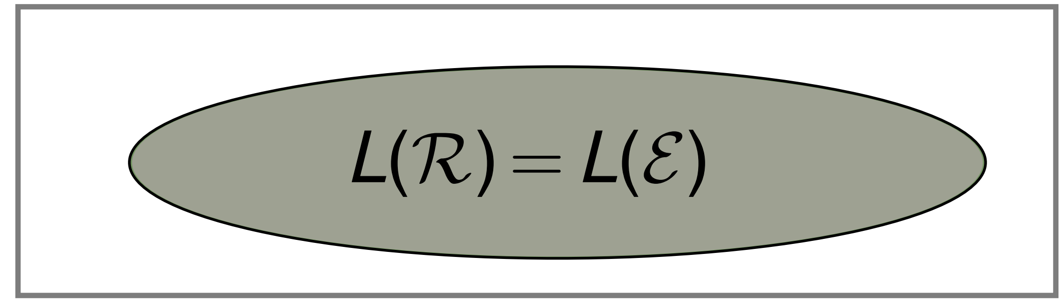
$\rightsquigarrow$ language acceptor

$$L(\mathcal{R}) = \{ w \in \Sigma^* \mid g(f(init, w)) = 1 \}$$

What does correctness for RNNs mean?

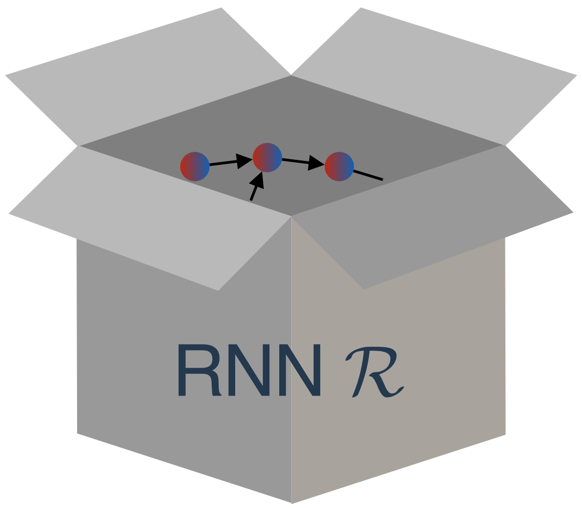
Specify correctness using finite automata $\mathcal{E}$ over Σ

$$L(\mathcal{R}) = L(\mathcal{E})$$



$\rightsquigarrow$ $L(\mathcal{R})$ is a regular language

restrictive: how about RNNs recognizing XML documents?



$$f : \underbrace{\mathbb{R}^k}_{\text{state}} \times \underbrace{\Sigma^*}_{\text{input}} \longrightarrow \underbrace{\mathbb{R}^k}_{\text{state}}$$

$$g : \underbrace{\mathbb{R}^k}_{\text{state}} \longrightarrow \{0,1\}$$

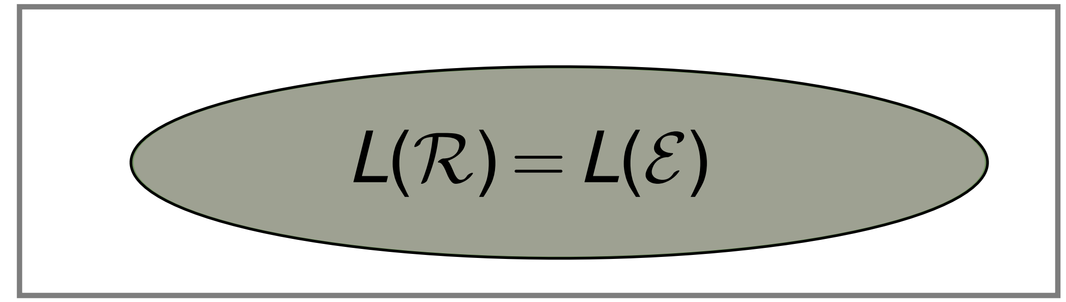
$\rightsquigarrow$ language acceptor

$$L(\mathcal{R}) = \left\{ w \in \Sigma^* \mid g(f(init, w)) = 1 \right\}$$

What does correctness for RNNs mean?

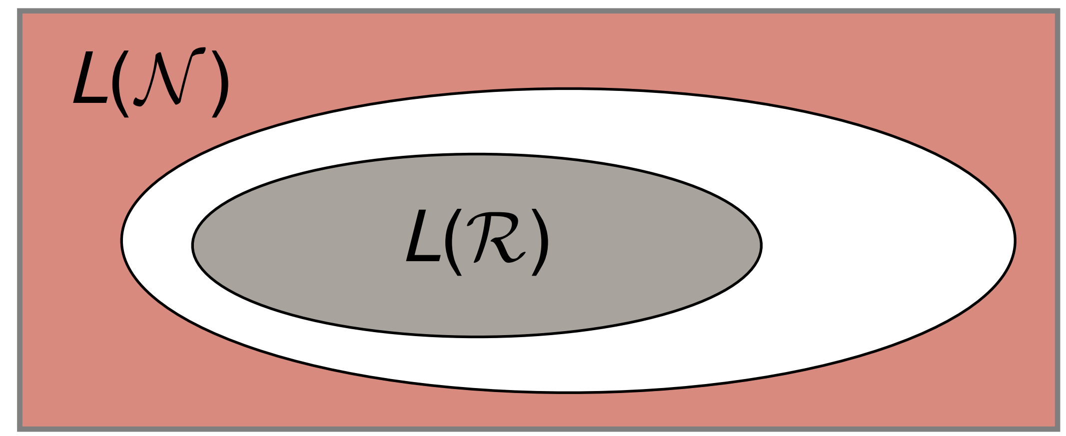
Specify correctness using finite automata $\mathcal{E}$, $\mathcal{N}$ over Σ

$L(\mathcal{R}) = L(\mathcal{E})$



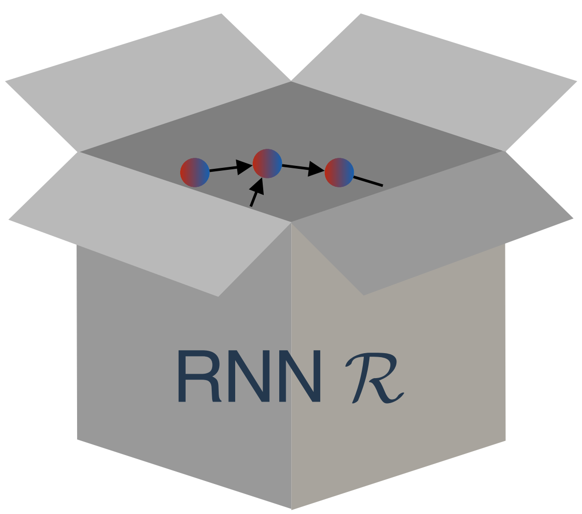
$\leadsto$ $L(\mathcal{R})$ is a regular language

restrictive: how about RNNs recognizing XML documents?



each $w \in L(\mathcal{N})$ must be classified as negative

$\mathcal{R}$ does not produce false positives



$$f : \underbrace{\mathbb{R}^k}_{\text{state}} \times \underbrace{\Sigma^*}_{\text{input}} \rightarrow \underbrace{\mathbb{R}^k}_{\text{state}}$$

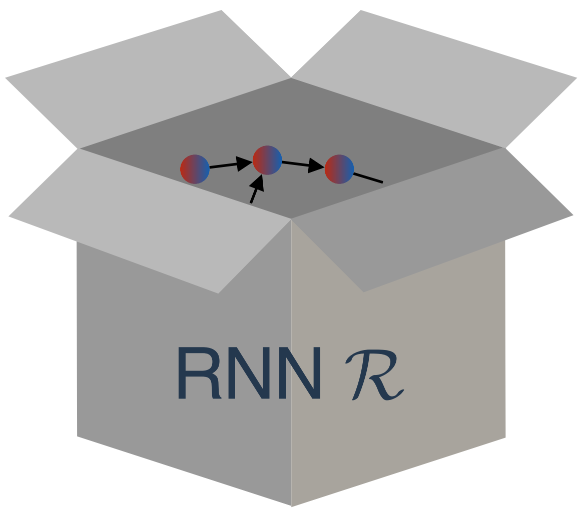
$$g : \underbrace{\mathbb{R}^k}_{\text{state}} \rightarrow \{0,1\}$$

$\leadsto$ language acceptor

$$L(\mathcal{R}) = \{w \in \Sigma^* \mid g(f(init, w)) = 1\}$$

What does correctness for RNNs mean?

Specify correctness using finite automata $\mathcal{E}, \mathcal{N}$ over Σ



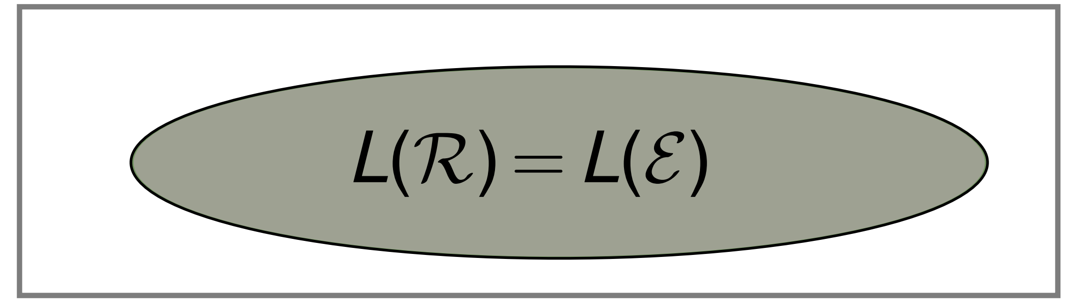
$$f : \underbrace{\mathbb{R}^k}_{\text{state}} \times \underbrace{\Sigma^*}_{\text{input}} \rightarrow \underbrace{\mathbb{R}^k}_{\text{state}}$$

$$g : \underbrace{\mathbb{R}^k}_{\text{state}} \rightarrow \{0,1\}$$

$\rightsquigarrow$ language acceptor

$$L(\mathcal{R}) = \{w \in \Sigma^* \mid g(f(init, w)) = 1\}$$

$$L(\mathcal{R}) = L(\mathcal{E})$$

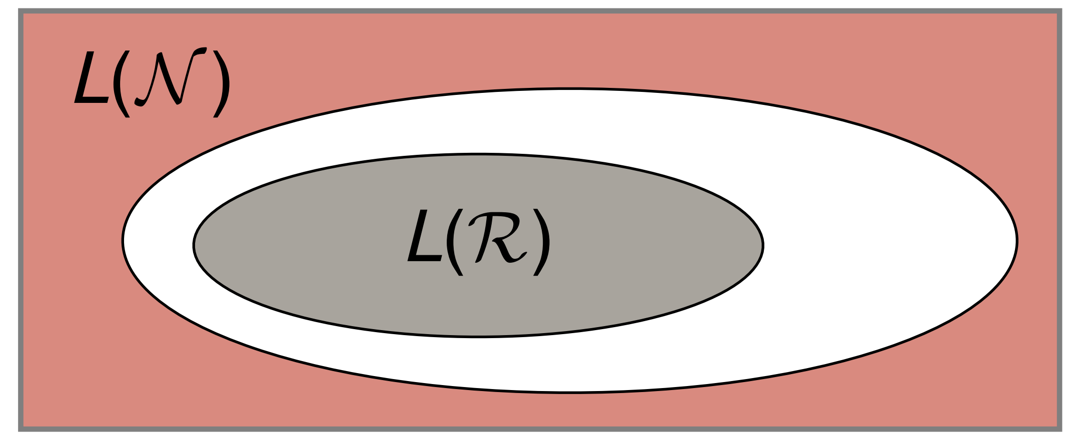


$\rightsquigarrow$ $L(\mathcal{R})$ is a regular language

restrictive: how about RNNs recognizing XML documents?

complement

$$L(\mathcal{R}) \subseteq L(\overline{\mathcal{N}})$$

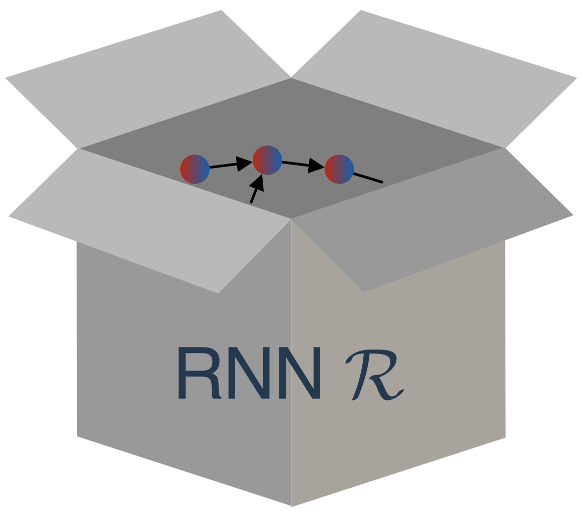


each $w \in L(\mathcal{N})$ must be classified as negative

$\mathcal{R}$ does not produce false positives

What does correctness for RNNs mean?

Specify correctness using finite automata $\mathcal{E}, \mathcal{N}, \mathcal{P}$ over Σ



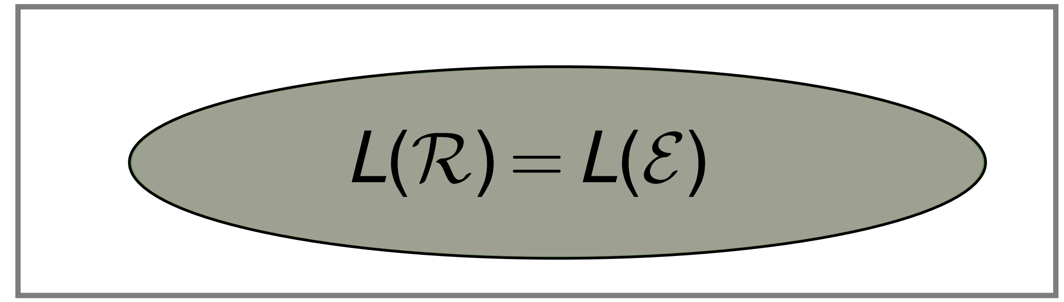
$$f : \underbrace{\mathbb{R}^k}_{\text{state}} \times \underbrace{\Sigma^*}_{\text{input}} \to \underbrace{\mathbb{R}^k}_{\text{state}}$$

$$g : \underbrace{\mathbb{R}^k}_{\text{state}} \to \{0,1\}$$

$\rightsquigarrow$ language acceptor

$$L(\mathcal{R}) = \{ w \in \Sigma^* \mid g(f(init, w)) = 1 \}$$

$$L(\mathcal{R}) = L(\mathcal{E})$$

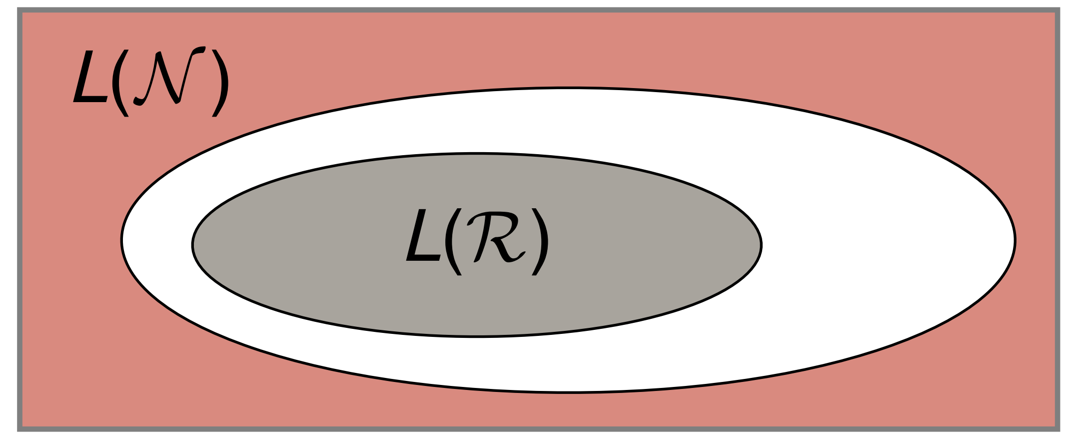


$\rightsquigarrow$ $L(\mathcal{R})$ is a regular language

restrictive: how about RNNs recognizing XML documents?

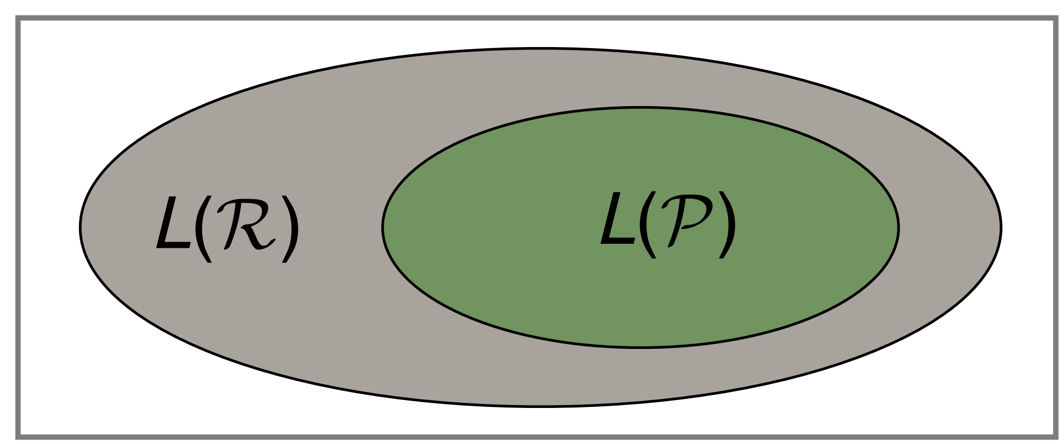
complement

$$L(\mathcal{R}) \subseteq L(\overline{\mathcal{N}})$$



each $w \in L(\mathcal{N})$ must be classified as negative

$\mathcal{R}$ does not produce false positives

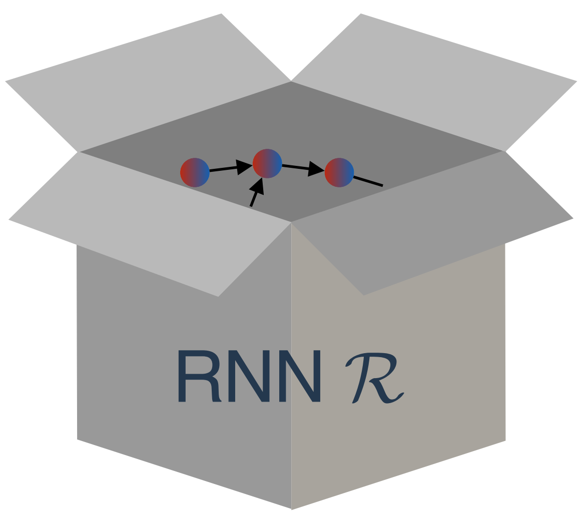


each $w \in L(\mathcal{P})$ must be classified as positive

$\mathcal{R}$ does not produce false negatives

What does correctness for RNNs mean?

Specify correctness using finite automata $\mathcal{E}, \mathcal{N}, \mathcal{P}$ over Σ



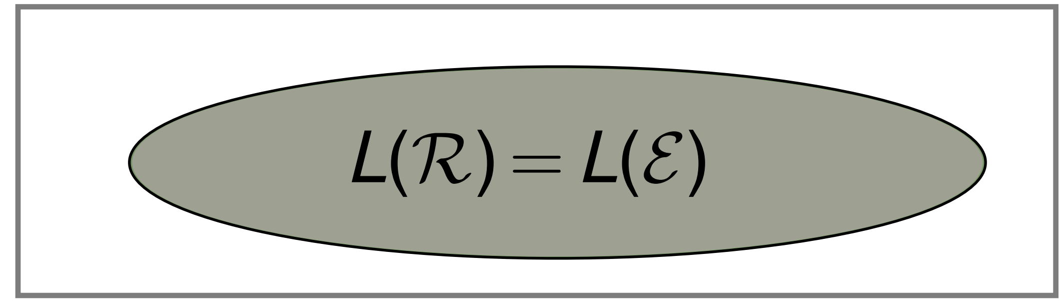
$$f : \underbrace{\mathbb{R}^k}_{\text{state}} \times \underbrace{\Sigma^*}_{\text{input}} \to \underbrace{\mathbb{R}^k}_{\text{state}}$$

$$g : \underbrace{\mathbb{R}^k}_{\text{state}} \to \{0,1\}$$

$\rightsquigarrow$ language acceptor

$$L(\mathcal{R}) = \{w \in \Sigma^* \mid g(f(init, w)) = 1\}$$

$L(\mathcal{R}) = L(\mathcal{E})$

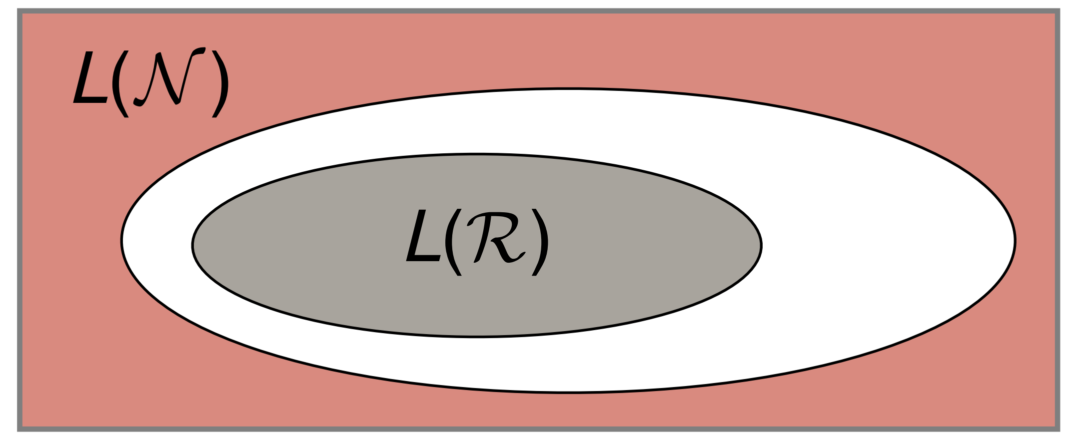


$\rightsquigarrow$ $L(\mathcal{R})$ is a regular language

restrictive: how about RNNs recognizing XML documents?

complement

$L(\mathcal{R}) \subseteq L(\overline{\mathcal{N}})$

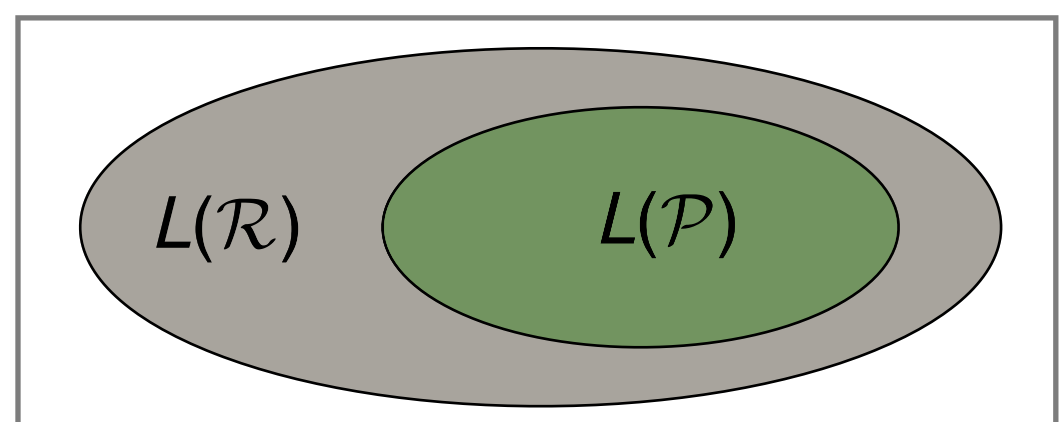


each $w \in L(\mathcal{N})$ must be classified as negative

$\mathcal{R}$ does not produce false positives

$L(\overline{\mathcal{R}}) \subseteq L(\overline{\mathcal{P}})$

complement

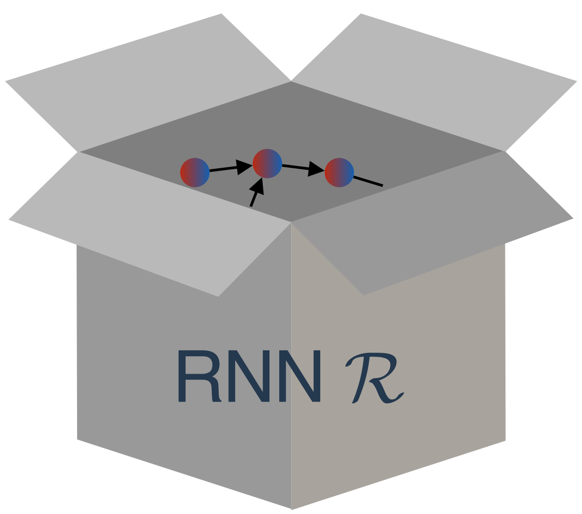


each $w \in L(\mathcal{P})$ must be classified as positive

$\mathcal{R}$ does not produce false negatives

What does correctness for RNNs mean?

Specify correctness using finite automata $\mathcal{E}, \mathcal{N}, \mathcal{P}$ over Σ



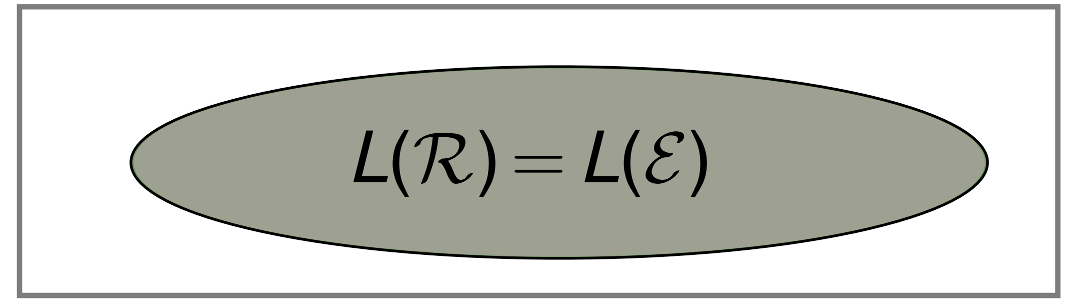
$$f : \underbrace{\mathbb{R}^k}_{\text{state}} \times \underbrace{\Sigma^*}_{\text{input}} \to \underbrace{\mathbb{R}^k}_{\text{state}}$$

$$g : \underbrace{\mathbb{R}^k}_{\text{state}} \to \{0,1\}$$

$\rightsquigarrow$ language acceptor

$$L(\mathcal{R}) = \{ w \in \Sigma^* \mid g(f(init, w)) = 1 \}$$

$L(\mathcal{R}) = L(\mathcal{E})$

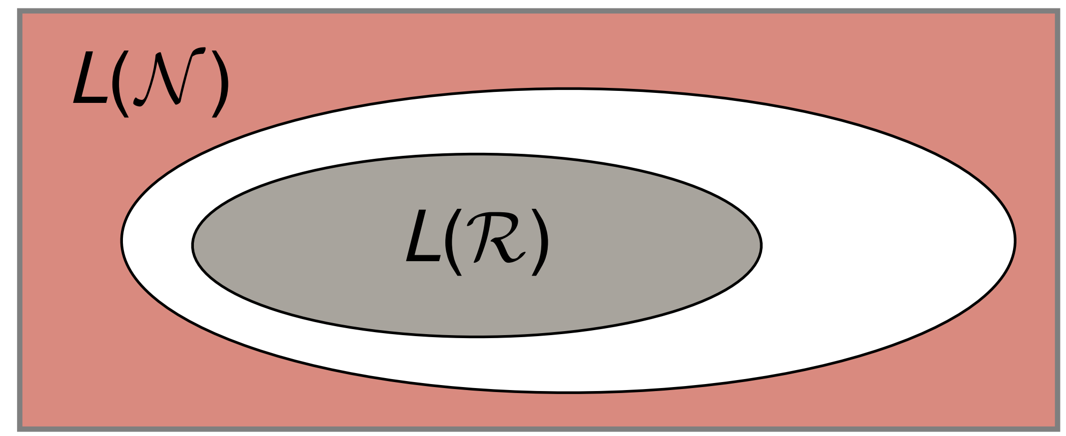


$\rightsquigarrow$ $L(\mathcal{R})$ is a regular language

restrictive: how about RNNs recognizing XML documents?

complement

$L(\mathcal{R}) \subseteq L(\overline{\mathcal{N}})$

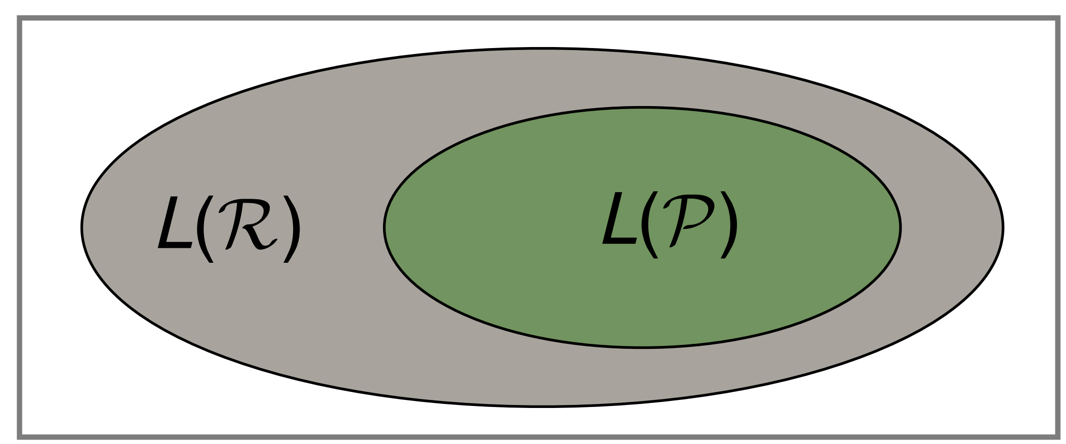


each $w \in L(\mathcal{N})$ must be classified as negative

$\mathcal{R}$ does not produce false positives

inclusion checking

$L(\overline{\mathcal{R}}) \subseteq L(\overline{\mathcal{P}})$



complement

each $w \in L(\mathcal{P})$ must be classified as positive

$\mathcal{R}$ does not produce false negatives

Assume $\mathcal{R}$ is supposed to recognize XML documents over

$$\Sigma = \left\{ \text{<list>, </list>, <item>, </item>} \right\}.$$

```
<list>
    <item>
        <list>
            <item>
            </item>
            <item>
            </item>
        </list>
    </item>
    <item>
    </item>
</list>
```

<list>
<item>
<list>
<item>
</item>
<item>
</item>
</list>
</item>
<item>
</item>
</list>

Assume $\mathcal{R}$ is supposed to recognize XML documents over

$$\Sigma = \{ \text{<list>}, \text{</list>}, \text{<item>}, \text{</item>} \}.$$

Regular negative specification $\mathcal{N}$:

<list>
<item>
<list>
<item>
</item>
<item>
</item>
</list>
</item>
<item>
</item>
</list>

Assume $\mathcal{R}$ is supposed to recognize XML documents over

$$\Sigma = \left\{ \text{<list>, </list>, <item>, </item>} \right\}.$$

Regular negative specification $\mathcal{N}$:

"there is an opening tag that is not eventually followed by a corresponding closing tag"

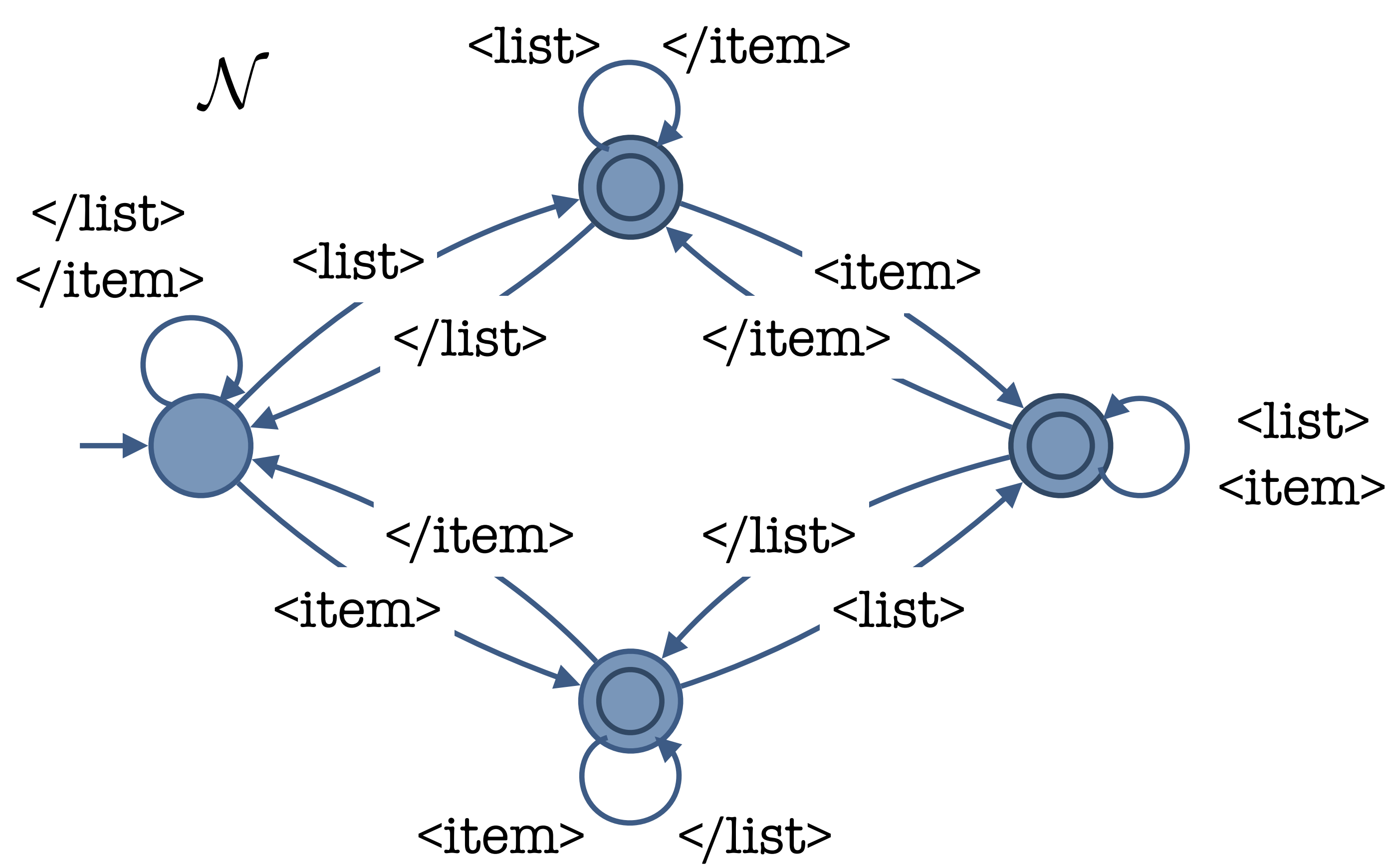
```
<list>
    <item>
        <list>
            <item>
            </item>
            <item>
            </item>
        </list>
    </item>
    <item>
    </item>
</list>
```

Assume $\mathcal{R}$ is supposed to recognize XML documents over

$$\Sigma = \left\{\text{<list>, </list>, <item>, </item>}\right\}.$$

Regular negative specification $\mathcal{N}$:

"there is an opening tag that is not eventually followed by a corresponding closing tag"



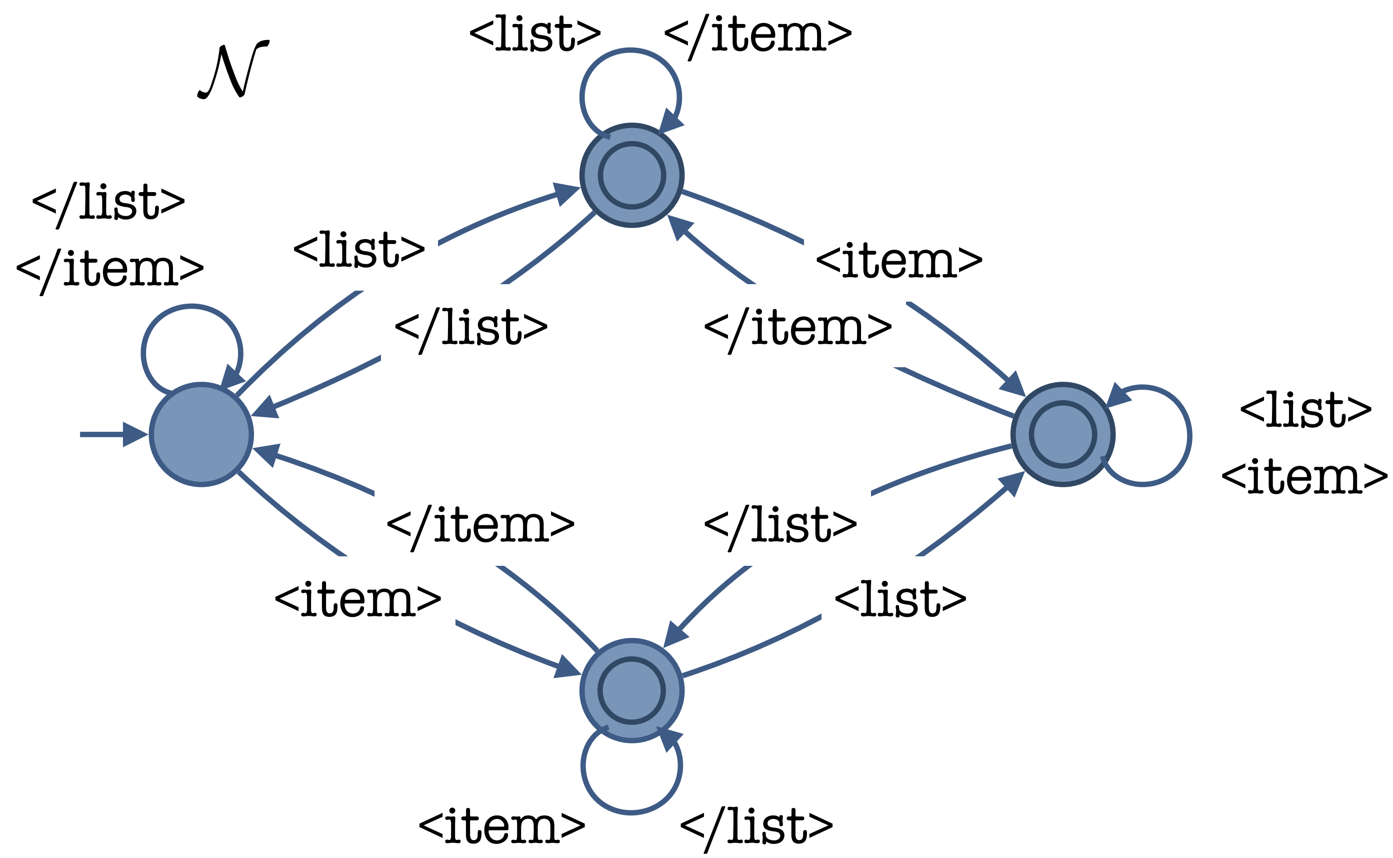
```
<list>
    <item>
        <list>
            <item>
            </item>
            <item>
            </item>
        </list>
    </item>
    <item>
    </item>
</list>
```

Assume $\mathcal{R}$ is supposed to recognize XML documents over

$$\Sigma = \{ \text{<list>, </list>, <item>, </item>} \}.$$

Regular negative specification $\mathcal{N}$:

"there is an opening tag that is not eventually followed by a corresponding closing tag"



<list> <item> </item> <item> </list> $\in L(\mathcal{N})$

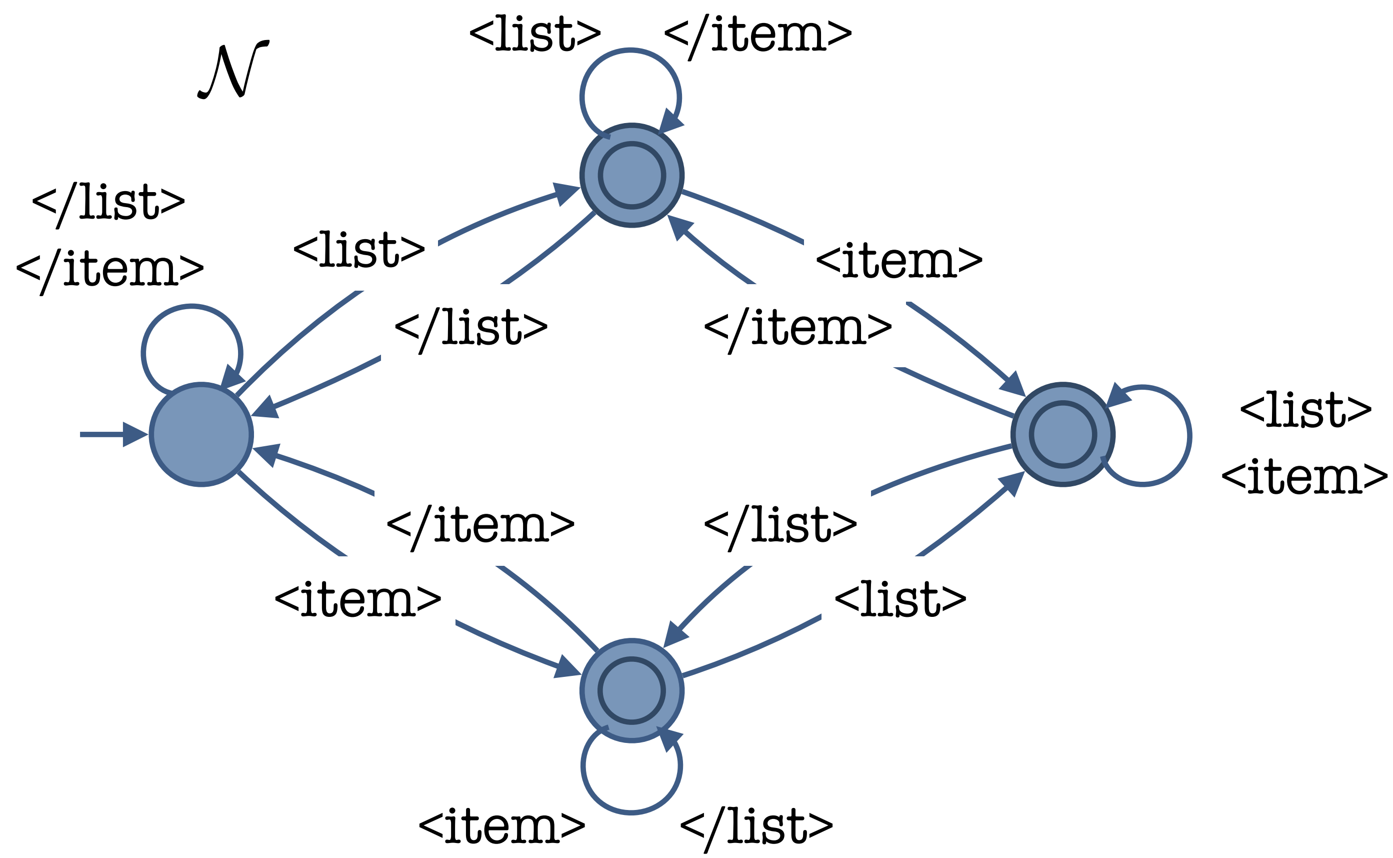
```
<list>
    <item>
        <list>
            <item>
            </item>
            <item>
            </item>
        </list>
    </item>
    <item>
    </item>
</list>
```

Assume $\mathcal{R}$ is supposed to recognize XML documents over

$$\Sigma = \{ \text{<list>}, \text{</list>}, \text{<item>}, \text{</item>} \}.$$

Regular negative specification $\mathcal{N}$:

"there is an opening tag that is not eventually followed by a corresponding closing tag"



<list> <item> </item> <item> </list> $\in L(\mathcal{N})$

<list> <item> <item> </item> </list> $\notin L(\mathcal{N})$

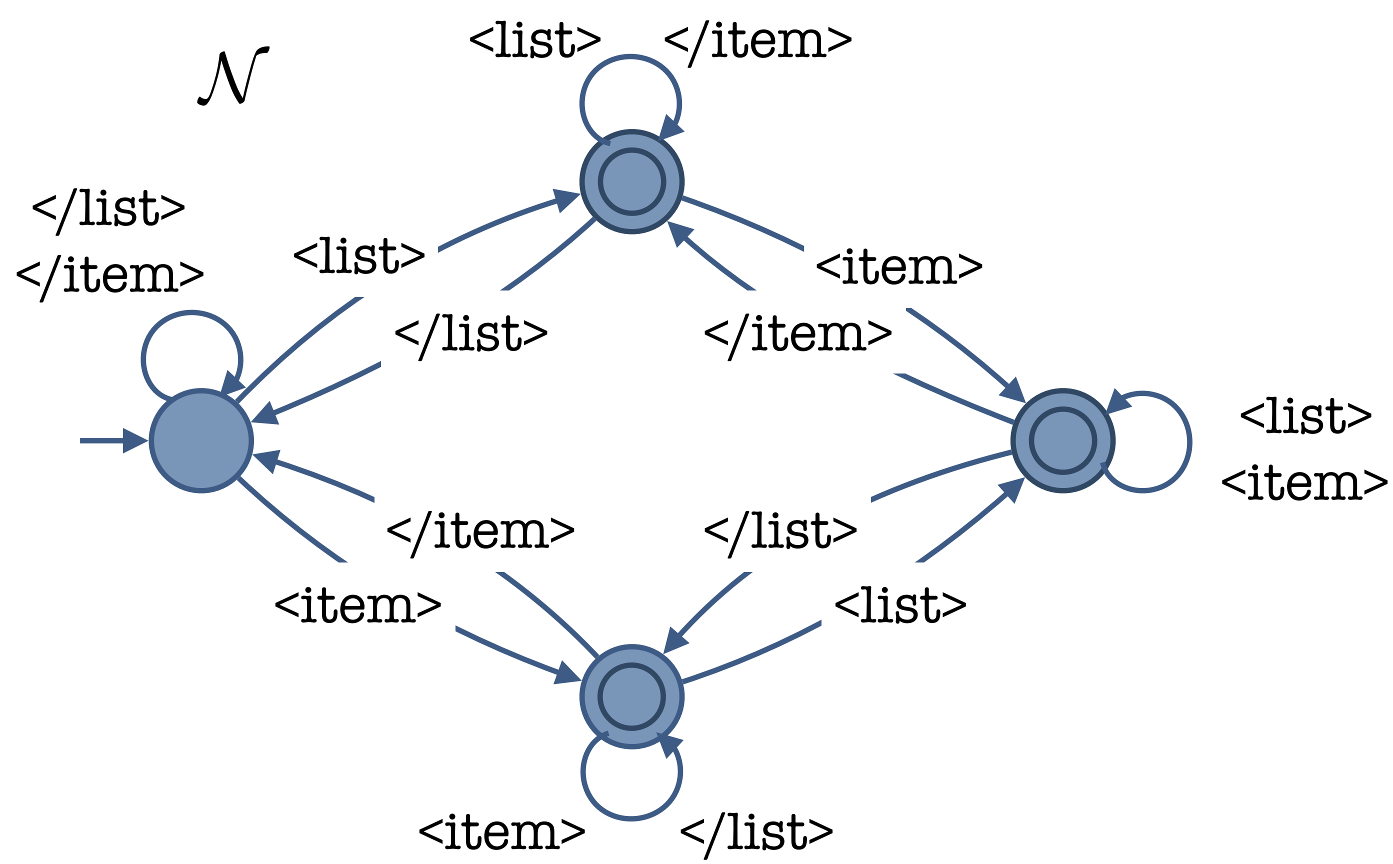
```
<list>
    <item>
        <list>
            <item>
            </item>
            <item>
            </item>
        </list>
    </item>
    <item>
    </item>
</list>
```

Assume $\mathcal{R}$ is supposed to recognize XML documents over

$$\Sigma = \{\, \texttt{<list>}, \texttt{</list>}, \texttt{<item>}, \texttt{</item>} \,\}.$$

Regular negative specification $\mathcal{N}$:

"there is an opening tag that is not eventually followed by a corresponding closing tag"



We check $L(\mathcal{R}) \subseteq L(\overline{\mathcal{N}})$.

$\texttt{<list> <item> </item> <item> </list>} \in L(\mathcal{N})$

$\texttt{<list> <item> <item> </item> </list>} \notin L(\mathcal{N})$

```
<list>
    <item>
        <list>
            <item>
            </item>
            <item>
            </item>
        </list>
    </item>
    <item>
    </item>
</list>
```

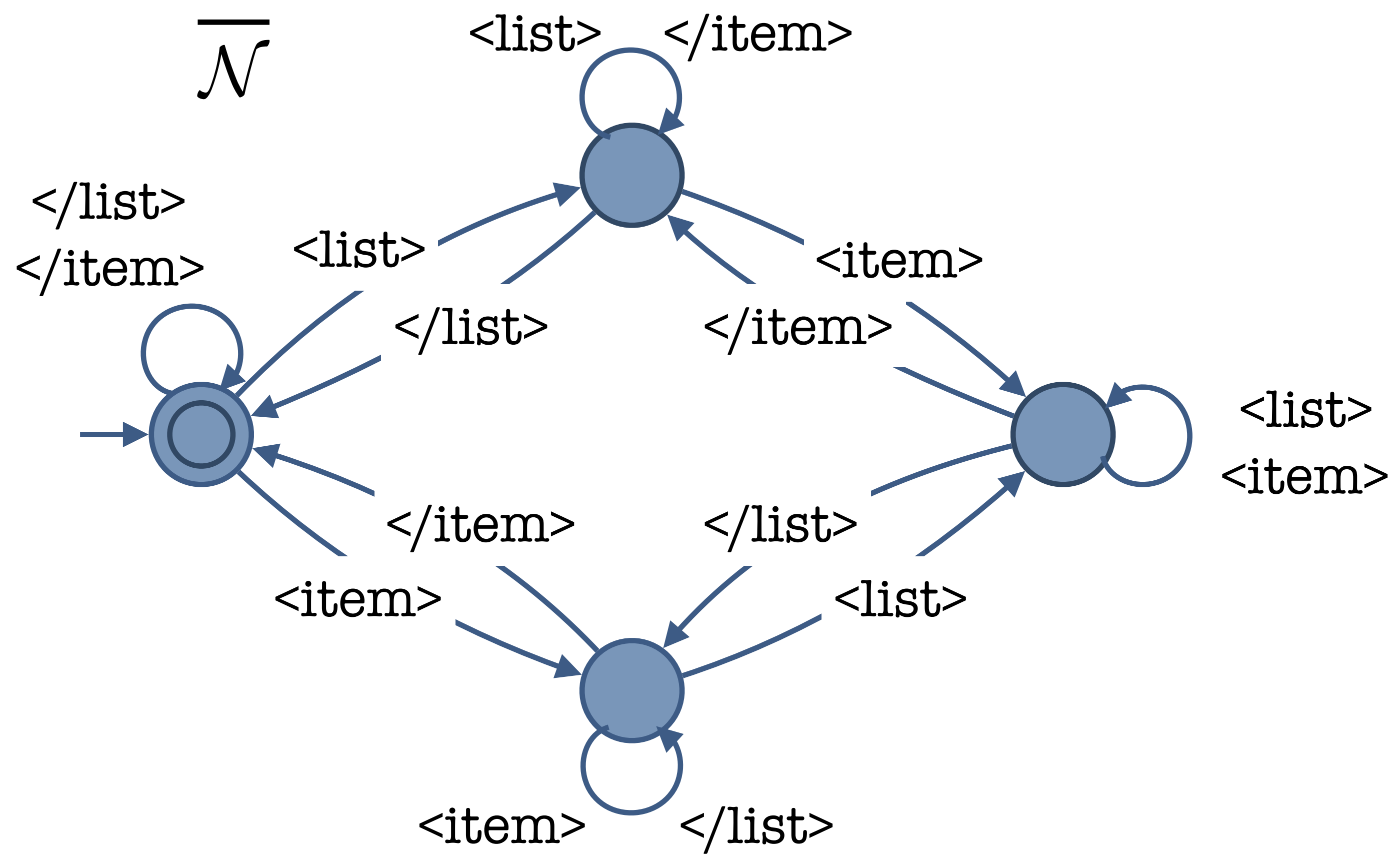
Assume $\mathcal{R}$ is supposed to recognize XML documents over

$$\Sigma = \{\text{<list>, </list>, <item>, </item>}\}.$$

Regular negative specification $\mathcal{N}$:

"there is an opening tag that is not eventually followed by a corresponding closing tag"

$\overline{\mathcal{N}}$



We check $L(\mathcal{R}) \subseteq L(\overline{\mathcal{N}})$.

<list> <item> </item> <item> </list> $\in L(\mathcal{N})$

<list> <item> <item> </item> </list> $\notin L(\mathcal{N})$

Checking $L(\mathcal{R}) \subseteq L(\mathcal{A})$ for deterministic finite automaton $\mathcal{A}$

Checking $L(\mathcal{R}) \subseteq L(\mathcal{A})$ for deterministic finite automaton $\mathcal{A}$

- Statistical model checking (SMC)

Checking $L(\mathcal{R}) \subseteq L(\mathcal{A})$ for deterministic finite automaton $\mathcal{A}$

- Statistical model checking (SMC)
- Automaton abstraction & model checking (AAMC)
  (model-learning approach)

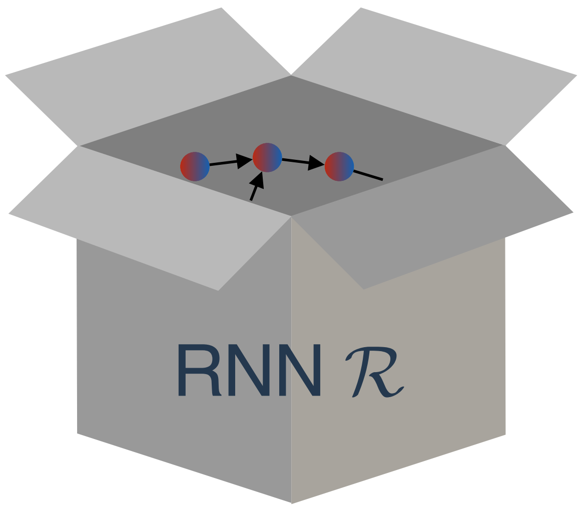
Checking $L(\mathcal{R}) \subseteq L(\mathcal{A})$ for deterministic finite automaton $\mathcal{A}$

- Statistical model checking (SMC)
- Automaton abstraction & model checking (AAMC)
  (model-learning approach)
- Property-directed verification (PDV)

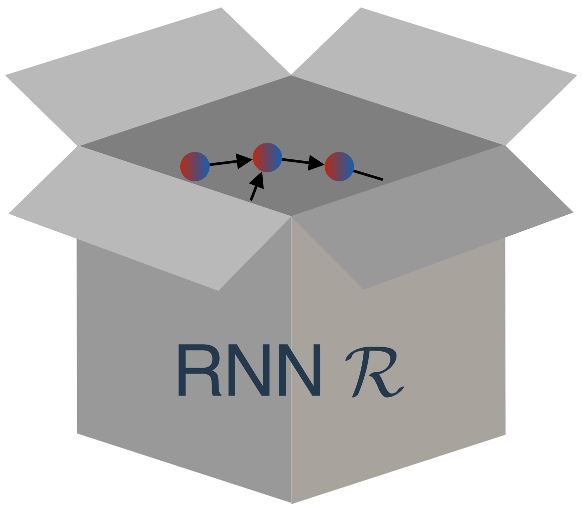
Statistical model checking (SMC)

Fix $\varepsilon, \gamma > 0$.

Sample $\log(2/\varepsilon)/(2\gamma^2)$ words over Σ.



Statistical model checking (SMC)



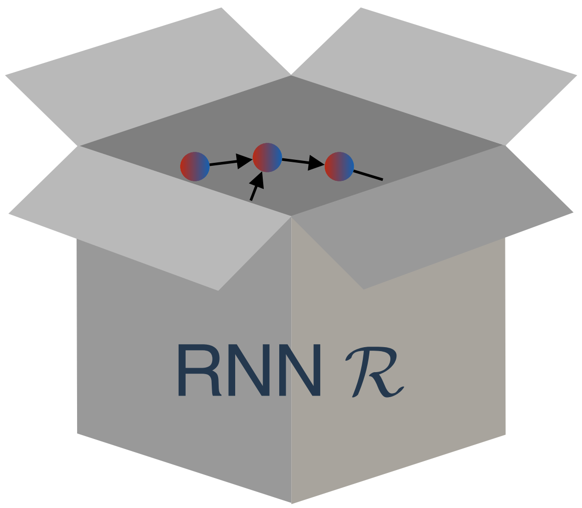
Fix $\varepsilon, \gamma > 0$.

Sample $\log(2/\varepsilon)/(2\gamma^2)$ words over Σ.

If, for some word w, we have $w \in L(\mathcal{R})$ and $w \notin L(\mathcal{A})$:

Property not satisfied.

Statistical model checking (SMC)



Fix $\varepsilon, \gamma > 0$.

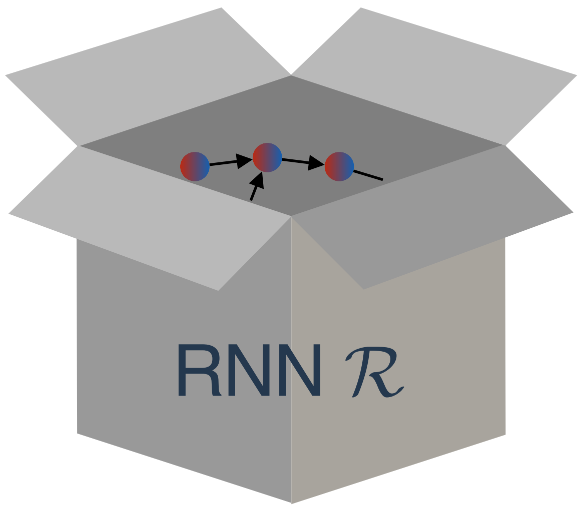
Sample $\log(2/\varepsilon)/(2\gamma^2)$ words over Σ.

If, for some word w, we have $w \in L(\mathcal{R})$ and $w \notin L(\mathcal{A})$:

Property not satisfied.

Else, $\mathcal{R}$ is ε-approximately correct with probability at least $1-\gamma$.

Statistical model checking (SMC)



Fix $\varepsilon, \gamma > 0$.

Sample $\log(2/\varepsilon)/(2\gamma^2)$ words over Σ.

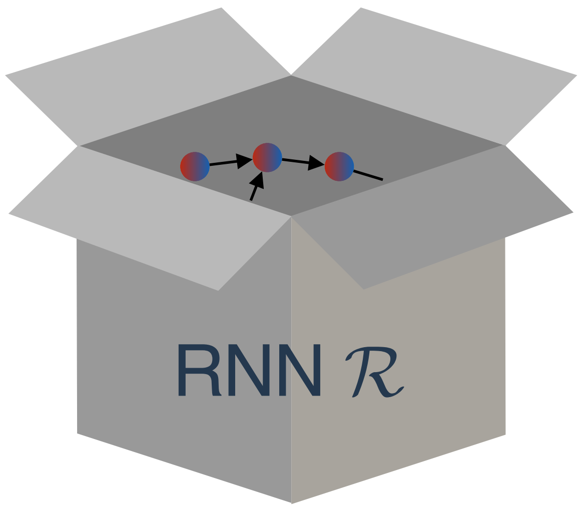
If, for some word w, we have $w \in L(\mathcal{R})$ and $w \notin L(\mathcal{A})$:

Property not satisfied.

Else, $\mathcal{R}$ is ε-approximately correct with probability at least $1-\gamma$.

$$\Pr\left(L(\mathcal{R}) \setminus L(\mathcal{A})\right) < \varepsilon$$

Statistical model checking (SMC)



Fix $\varepsilon, \gamma > 0$.

Sample $\log(2/\varepsilon)/(2\gamma^2)$ words over Σ.

If, for some word w, we have $w \in L(\mathcal{R})$ and $w \notin L(\mathcal{A})$:

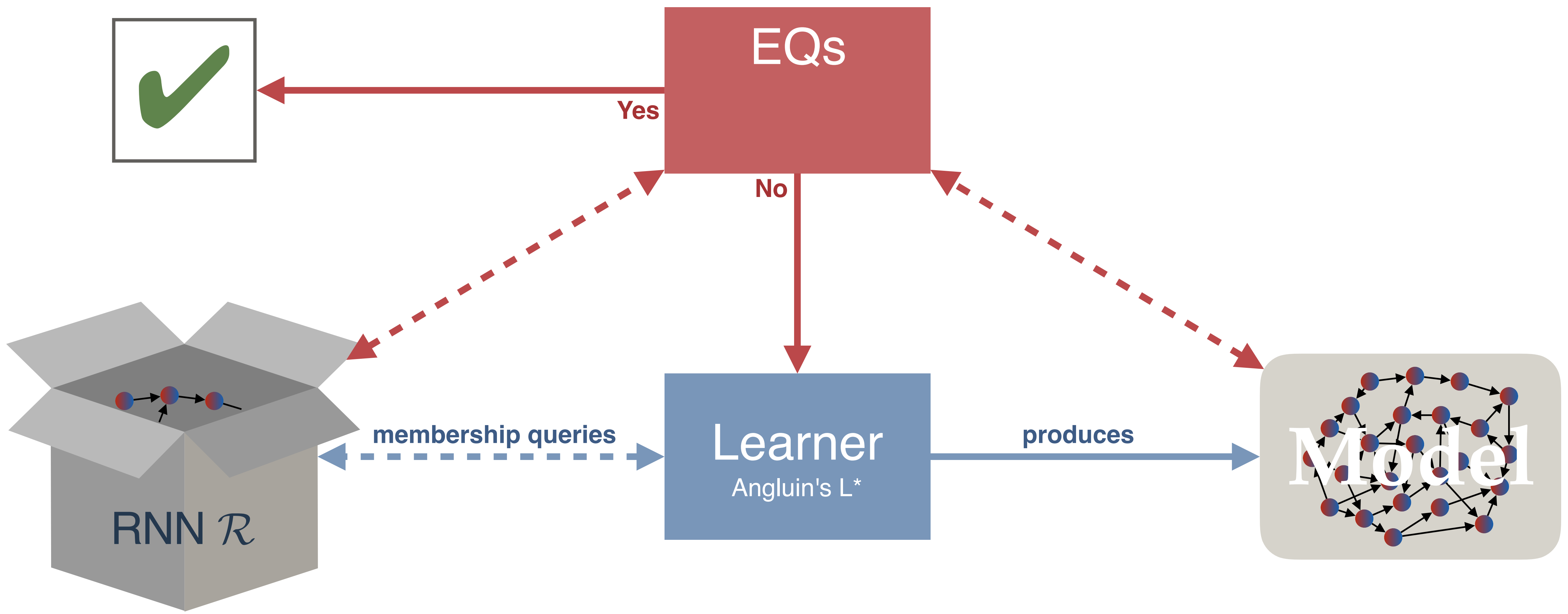
Property not satisfied.

Else, $\mathcal{R}$ is ε-approximately correct with probability at least $1-\gamma$.

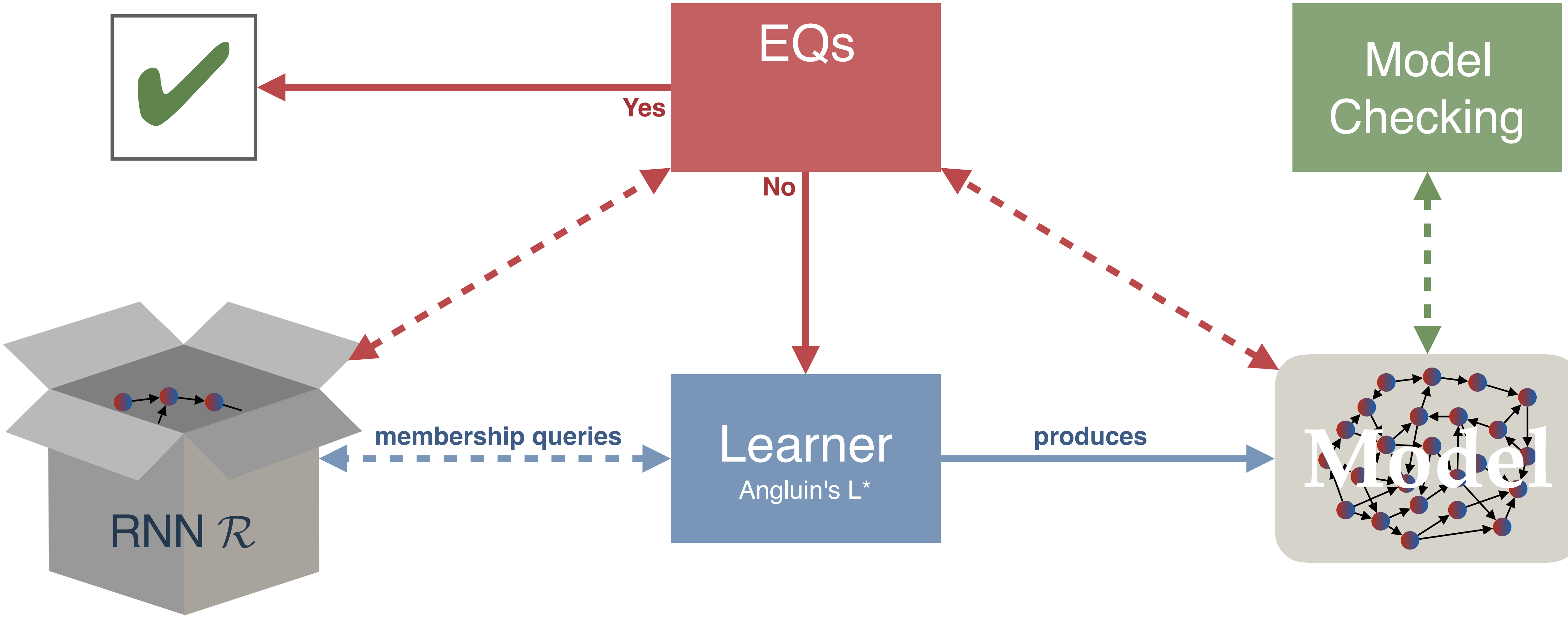
$$\Pr\left(L(\mathcal{R}) \setminus L(\mathcal{A})\right) < \varepsilon$$

- Relies on probability distribution.
- May require many queries.

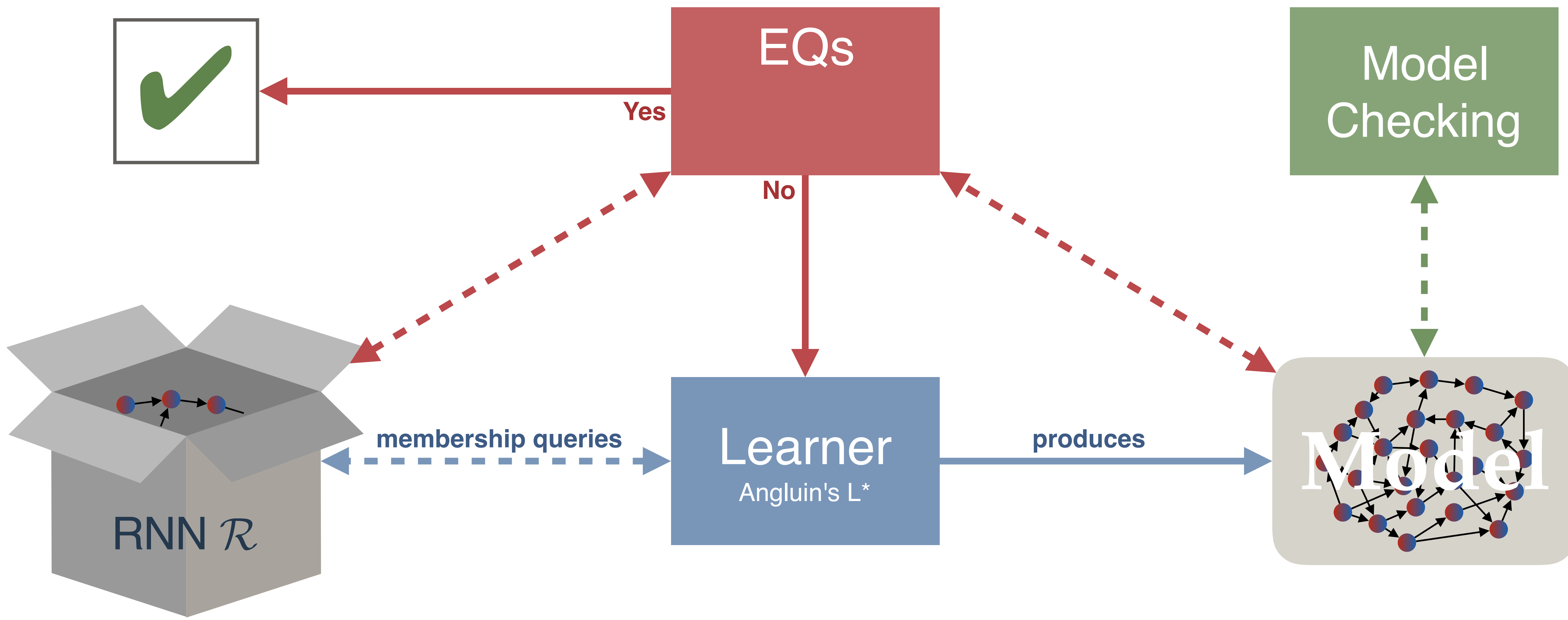
Automaton abstraction & model checking (AAMC)



Automaton abstraction & model checking (AAMC)

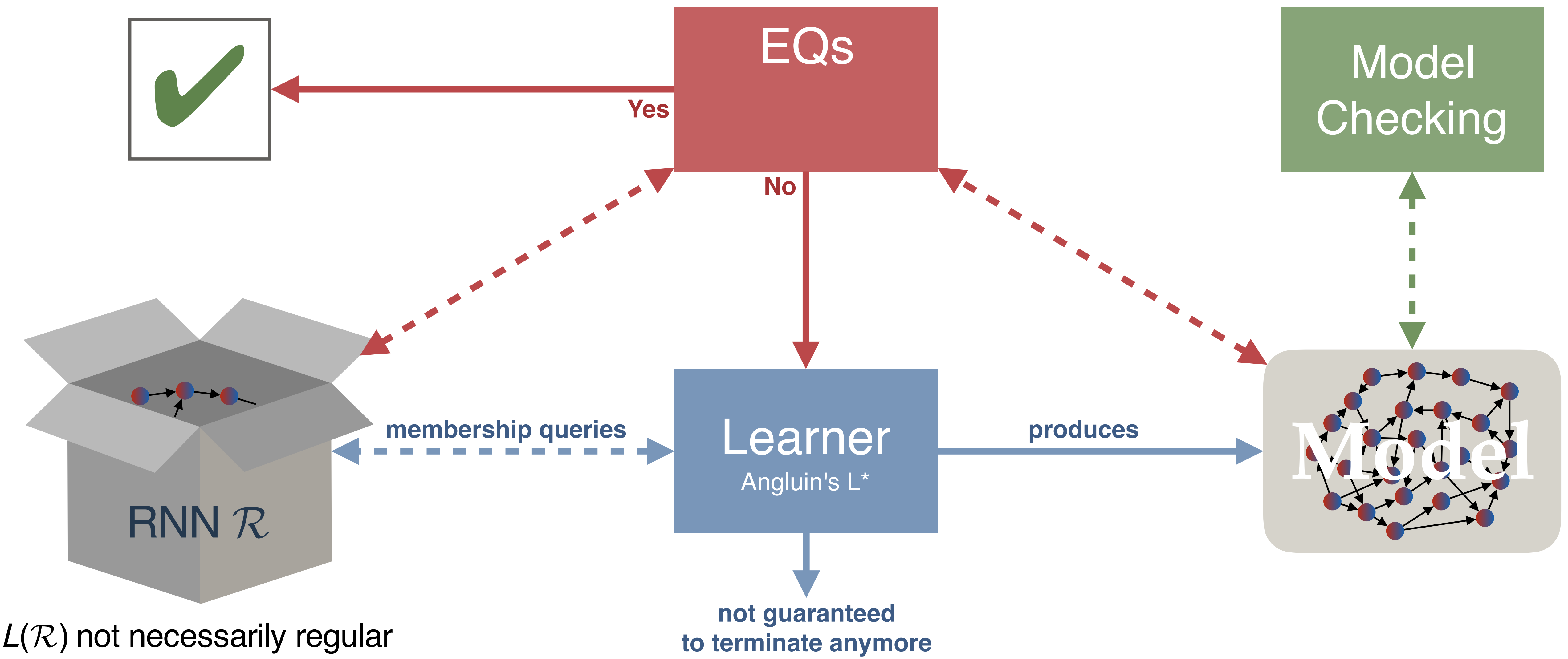


Automaton abstraction & model checking (AAMC)



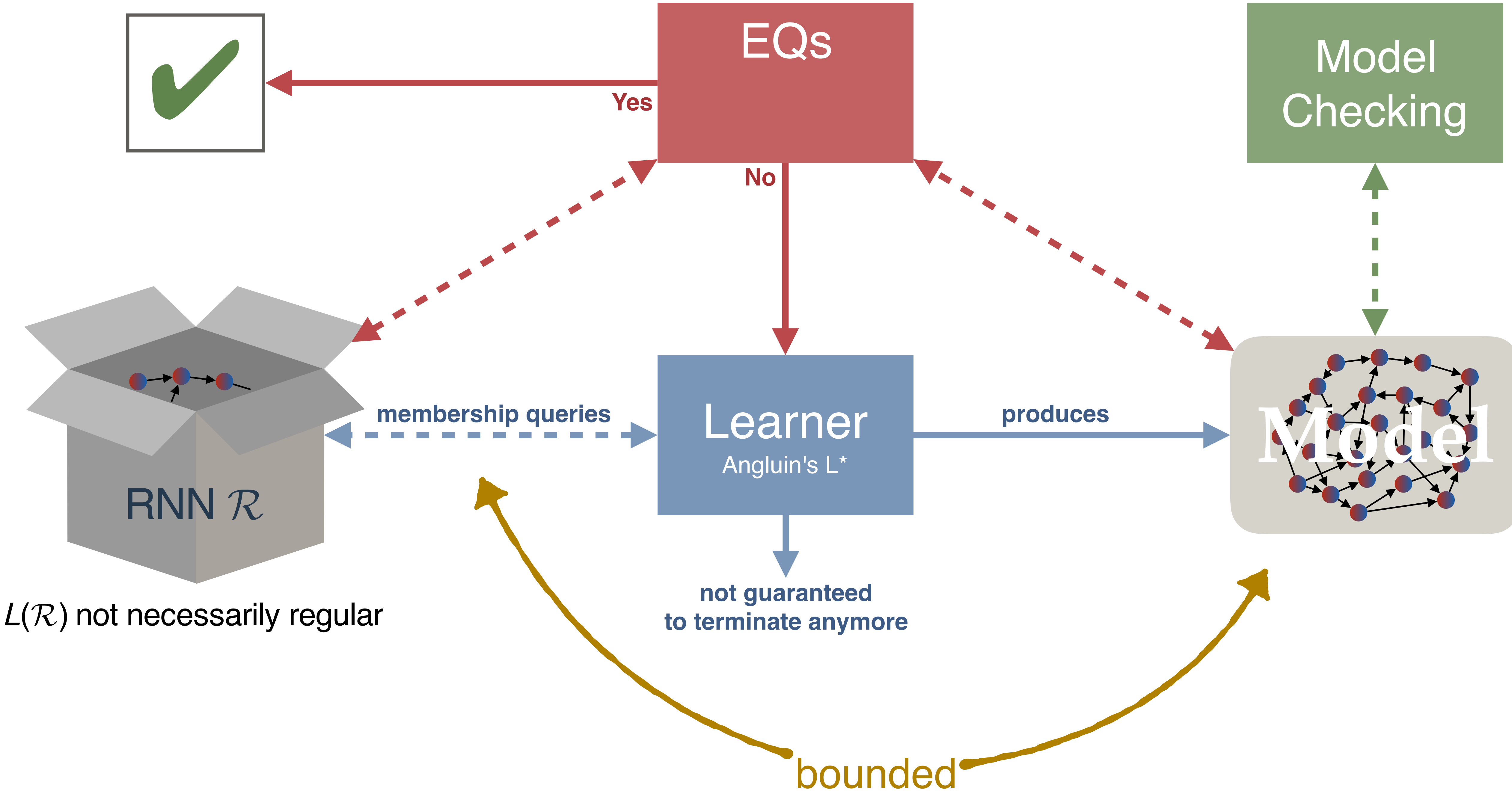
$L(\mathcal{R})$ not necessarily regular

Automaton abstraction & model checking (AAMC)



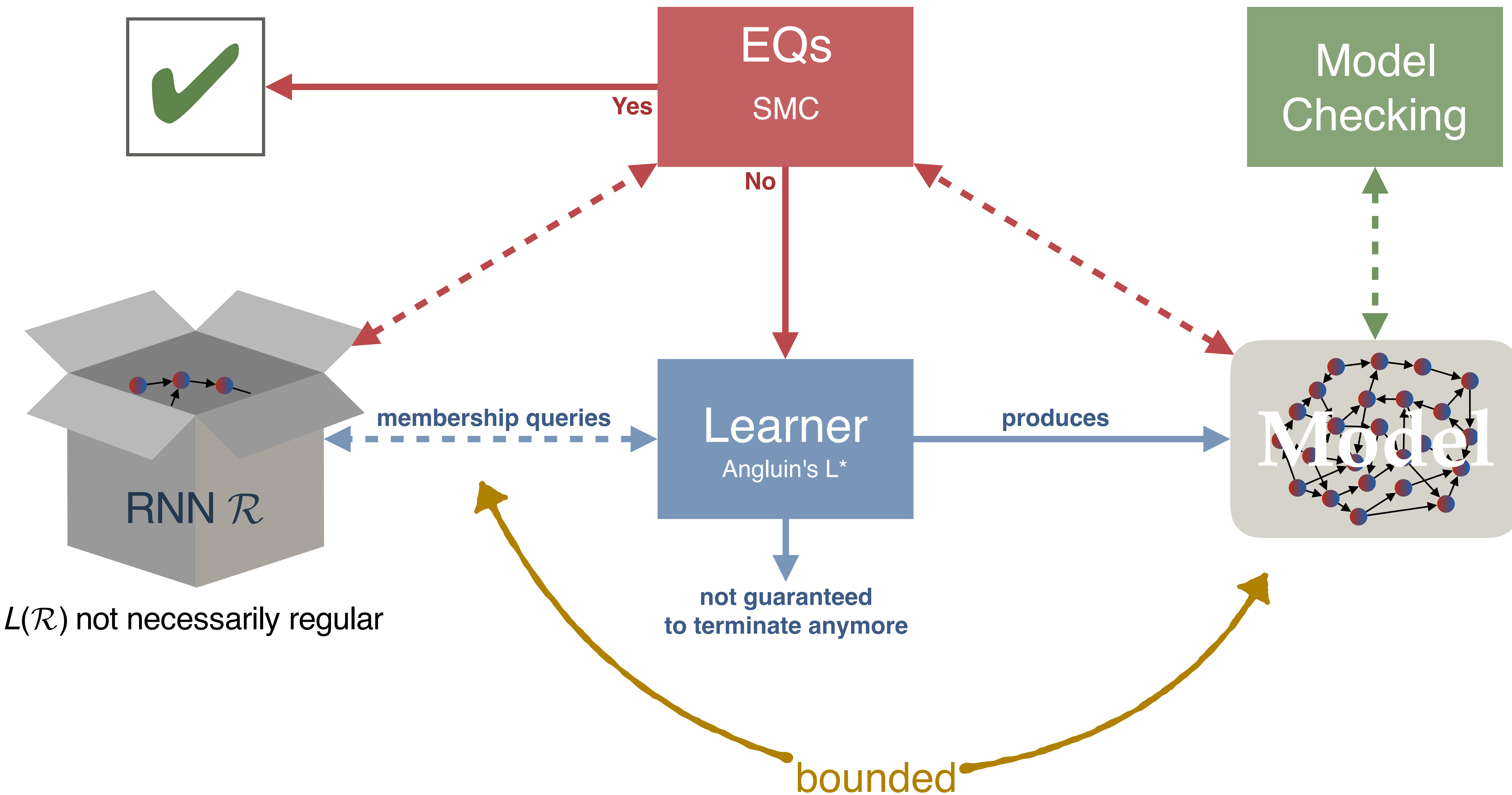
$L(\mathcal{R})$ not necessarily regular

Automaton abstraction & model checking (AAMC)



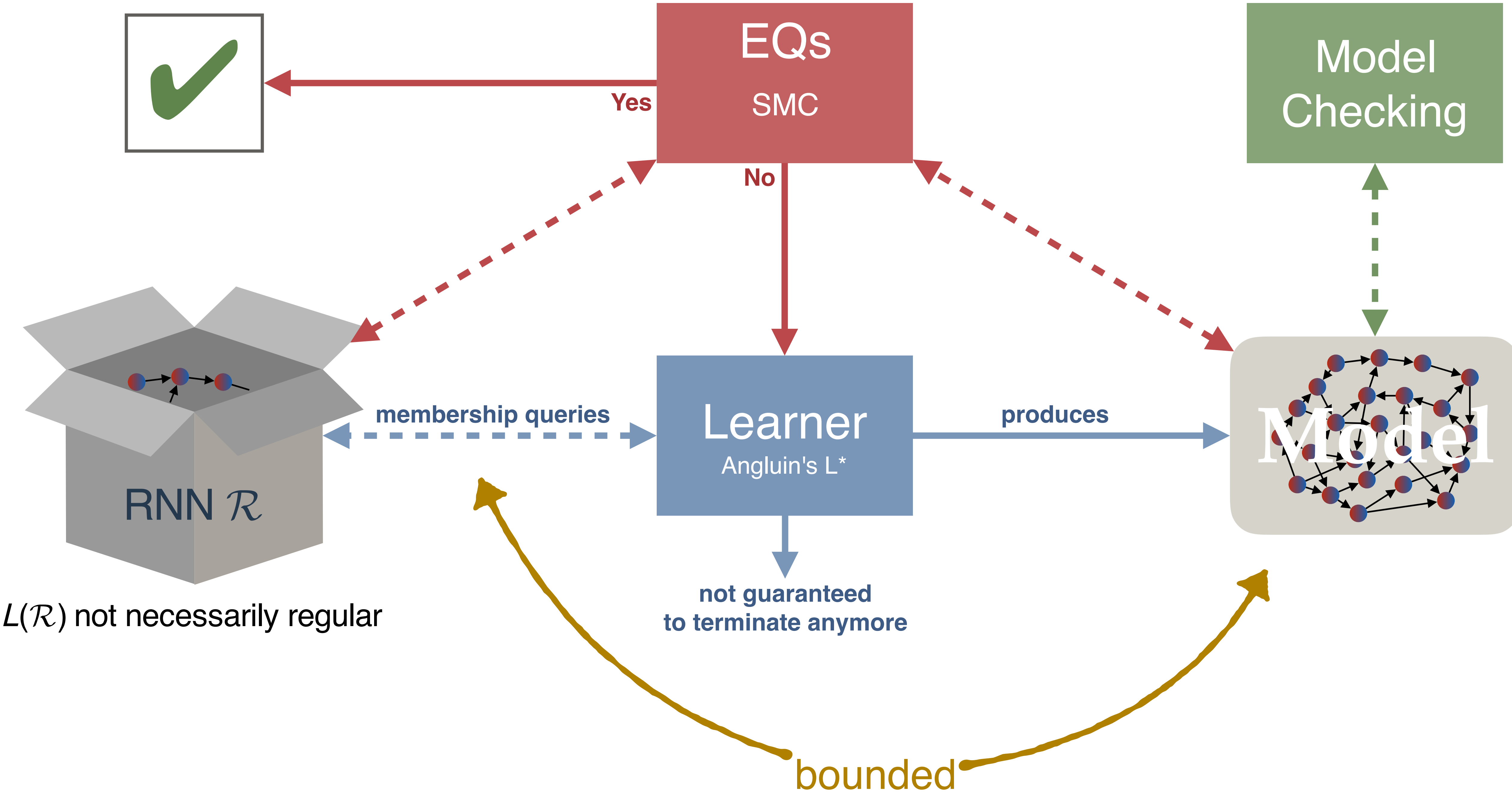
[Mayr, Yovine: Regular inference on artificial neural networks. CD-MAKE 2018]

Automaton abstraction & model checking (AAMC)



[Mayr, Yovine: Regular inference on artificial neural networks. CD-MAKE 2018]

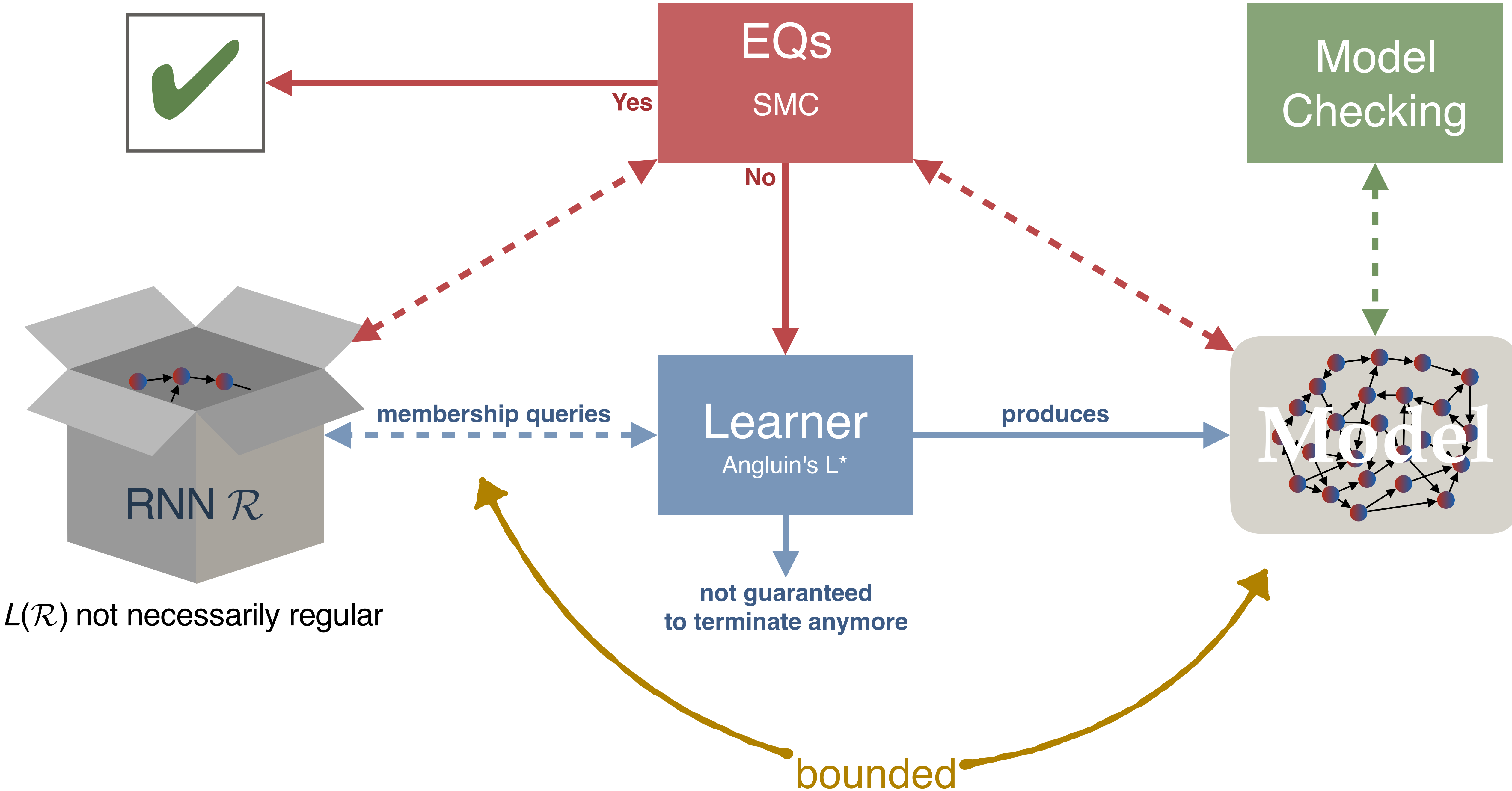
Automaton abstraction & model checking (AAMC)



- Relies on probability distribution.

[Mayr, Yovine: Regular inference on artificial neural networks. CD-MAKE 2018]

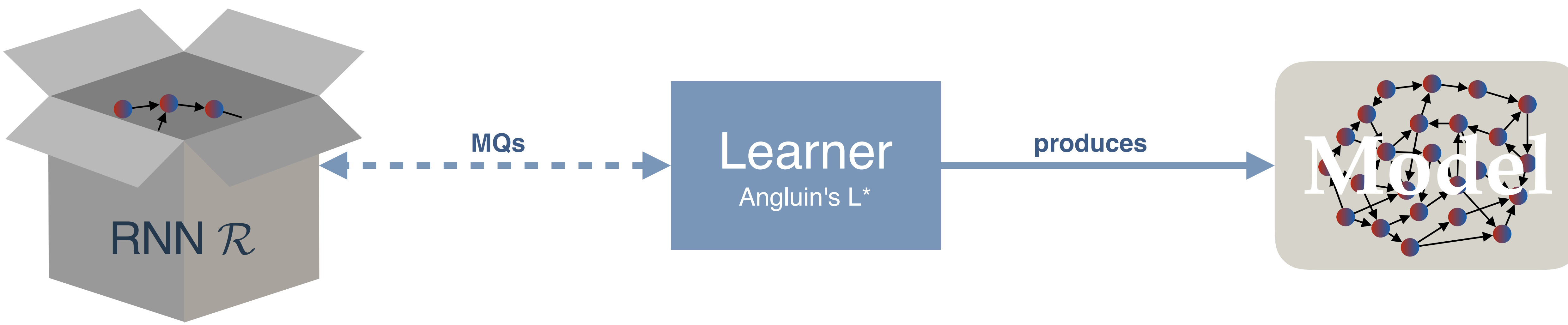
Automaton abstraction & model checking (AAMC)



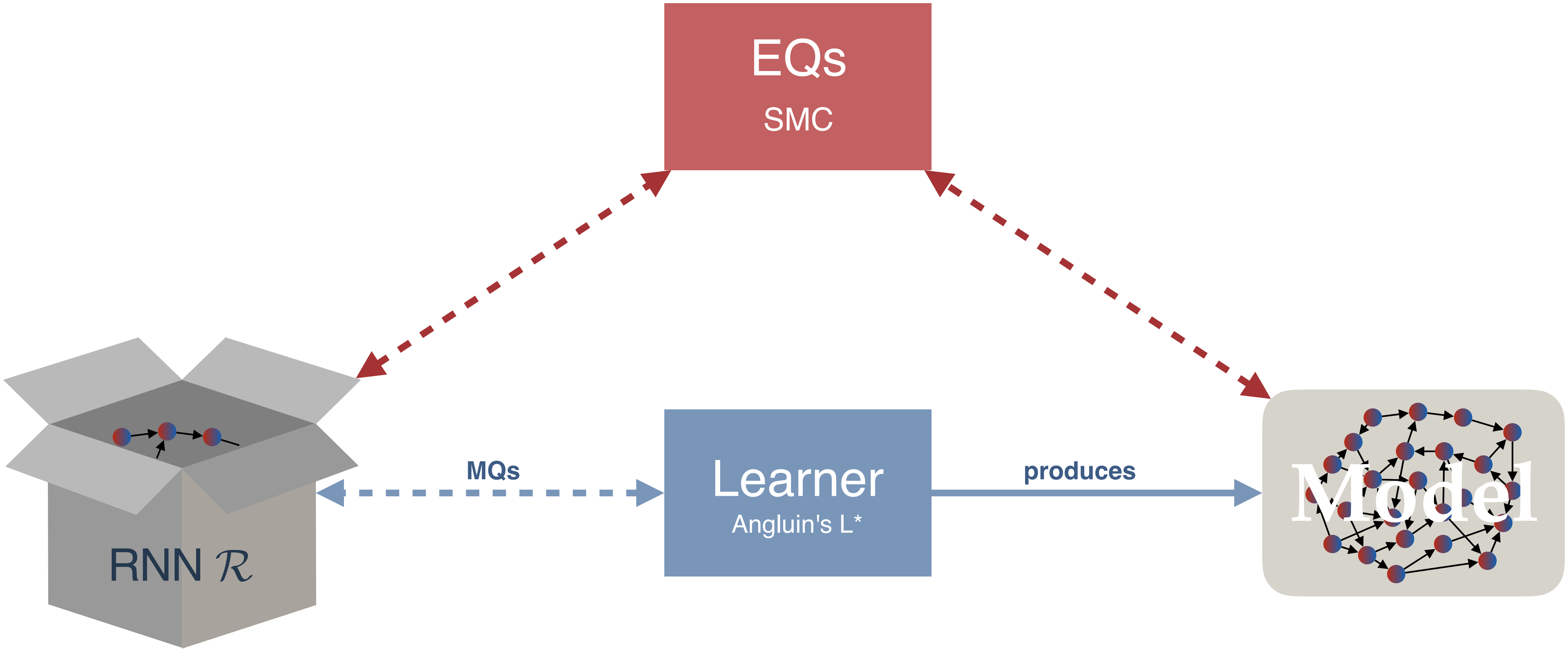
- Relies on probability distribution.
- Counterexample may be spurious.

As EQs are implemented by SMC, a "counterexample" may be classified by RNN as negative.

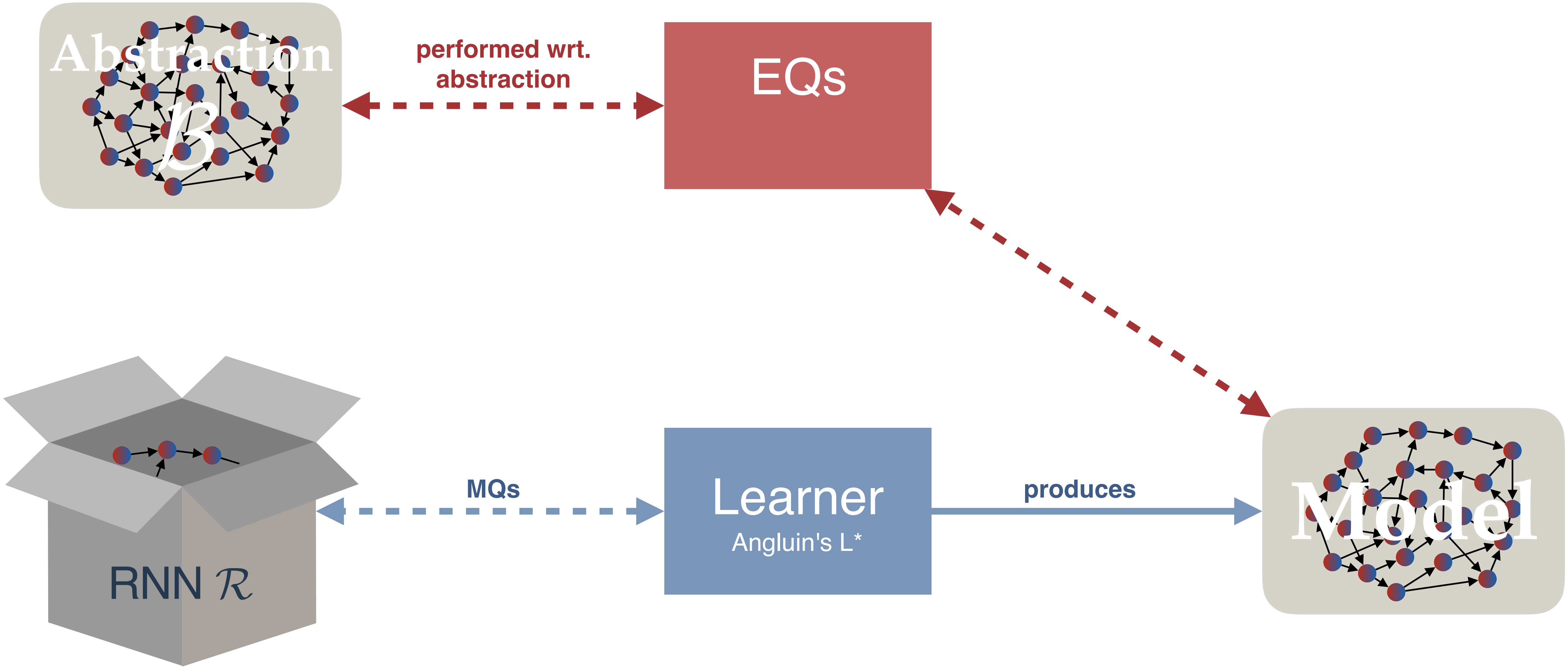
[Mayr, Yovine: Regular inference on artificial neural networks. CD-MAKE 2018]



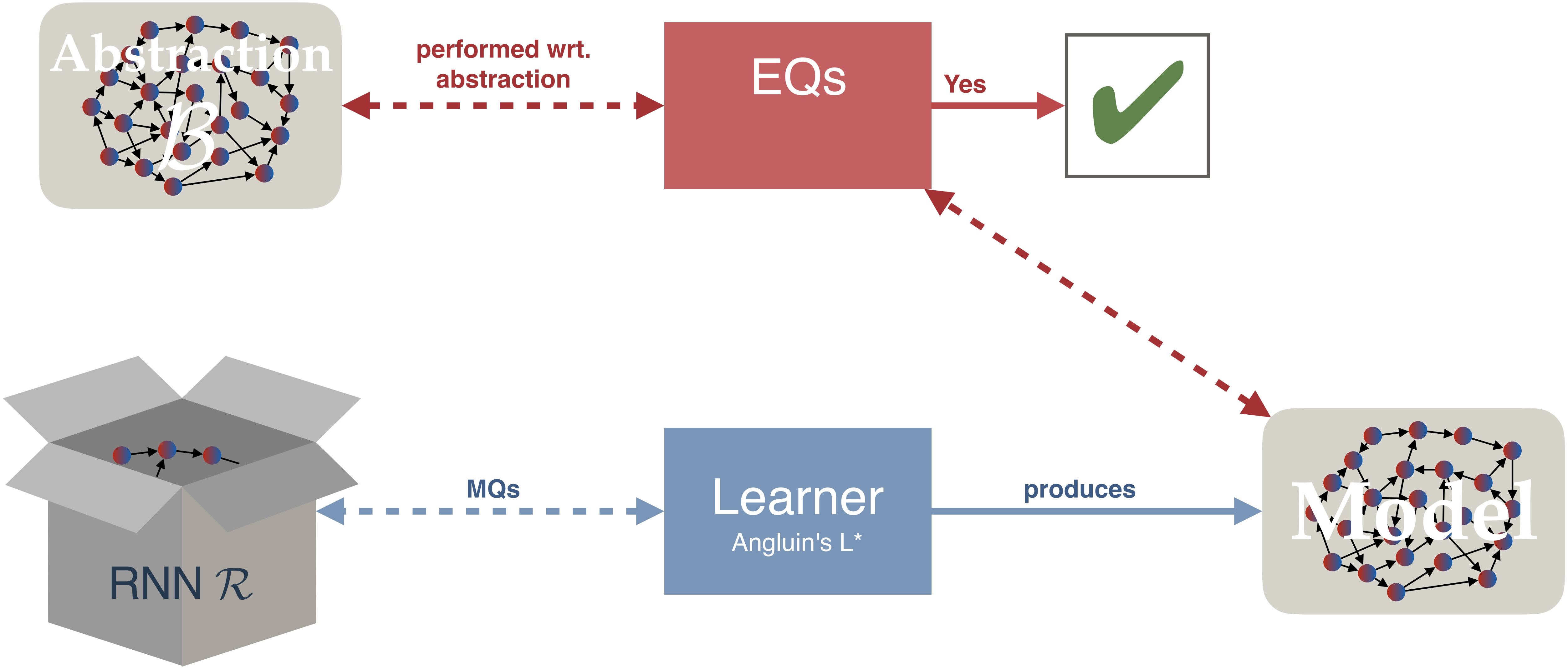
[Weiss, Goldberg, Yahav: *Extracting Automata from Recurrent Neural Networks Using Queries and Counterexamples*. ICML 2018]



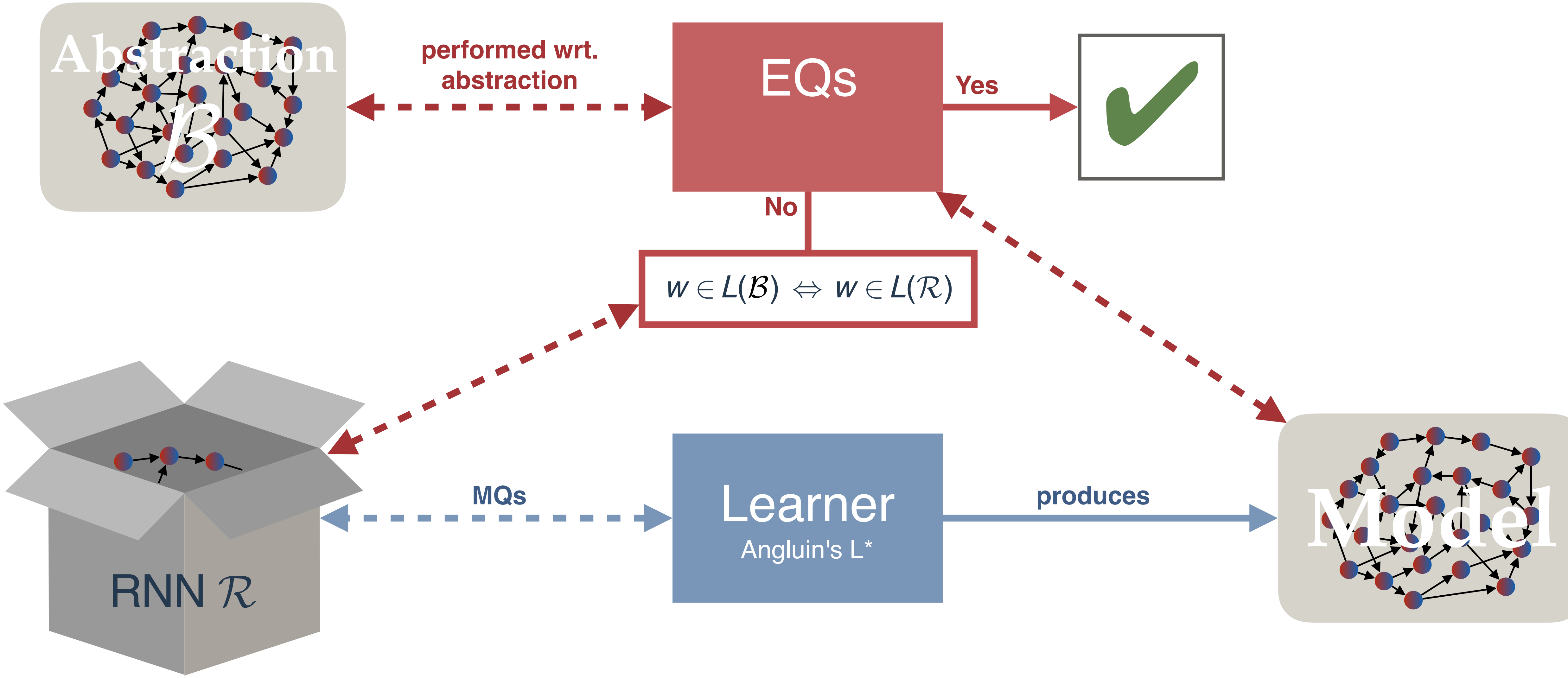
[Weiss, Goldberg, Yahav: *Extracting Automata from Recurrent Neural Networks Using Queries and Counterexamples*. ICML 2018]



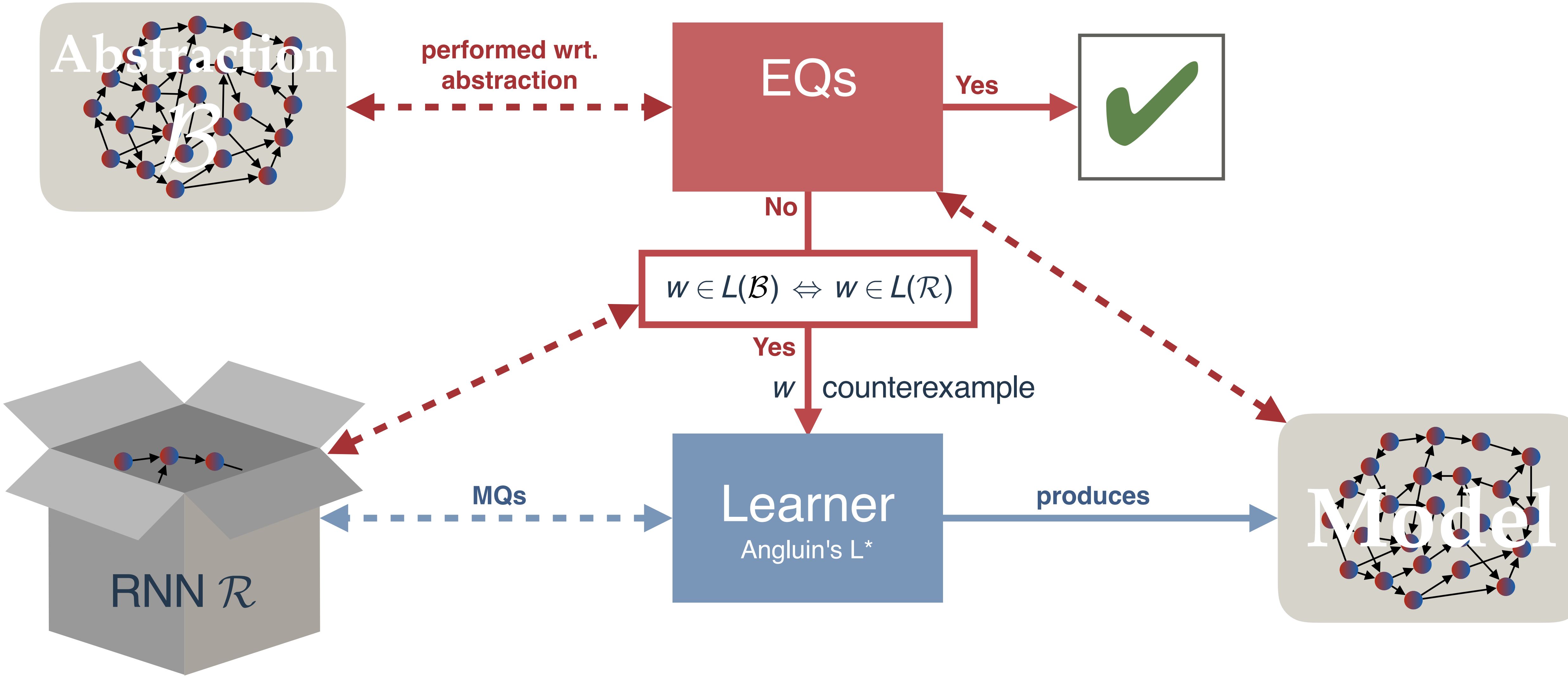
[Weiss, Goldberg, Yahav: *Extracting Automata from Recurrent Neural Networks Using Queries and Counterexamples*. ICML 2018]



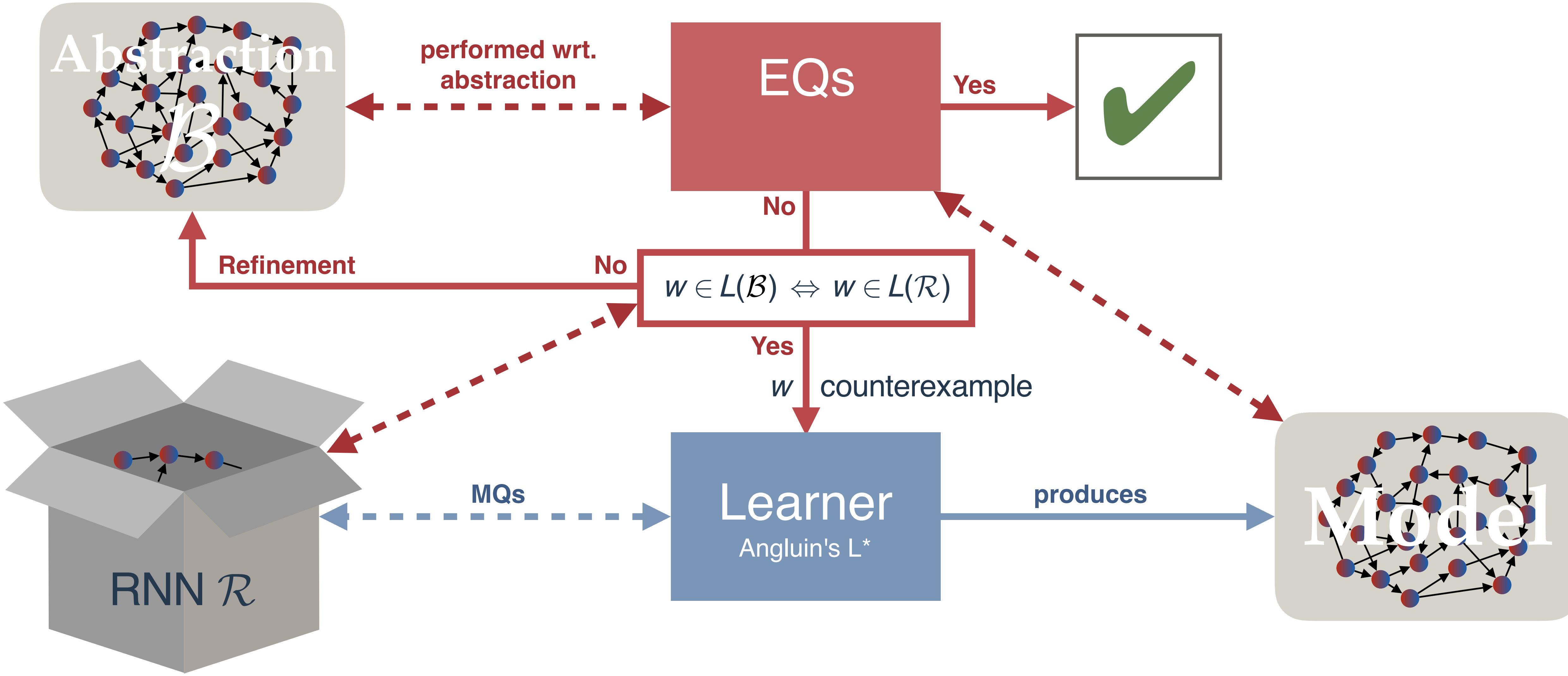
[Weiss, Goldberg, Yahav: *Extracting Automata from Recurrent Neural Networks Using Queries and Counterexamples*. ICML 2018]



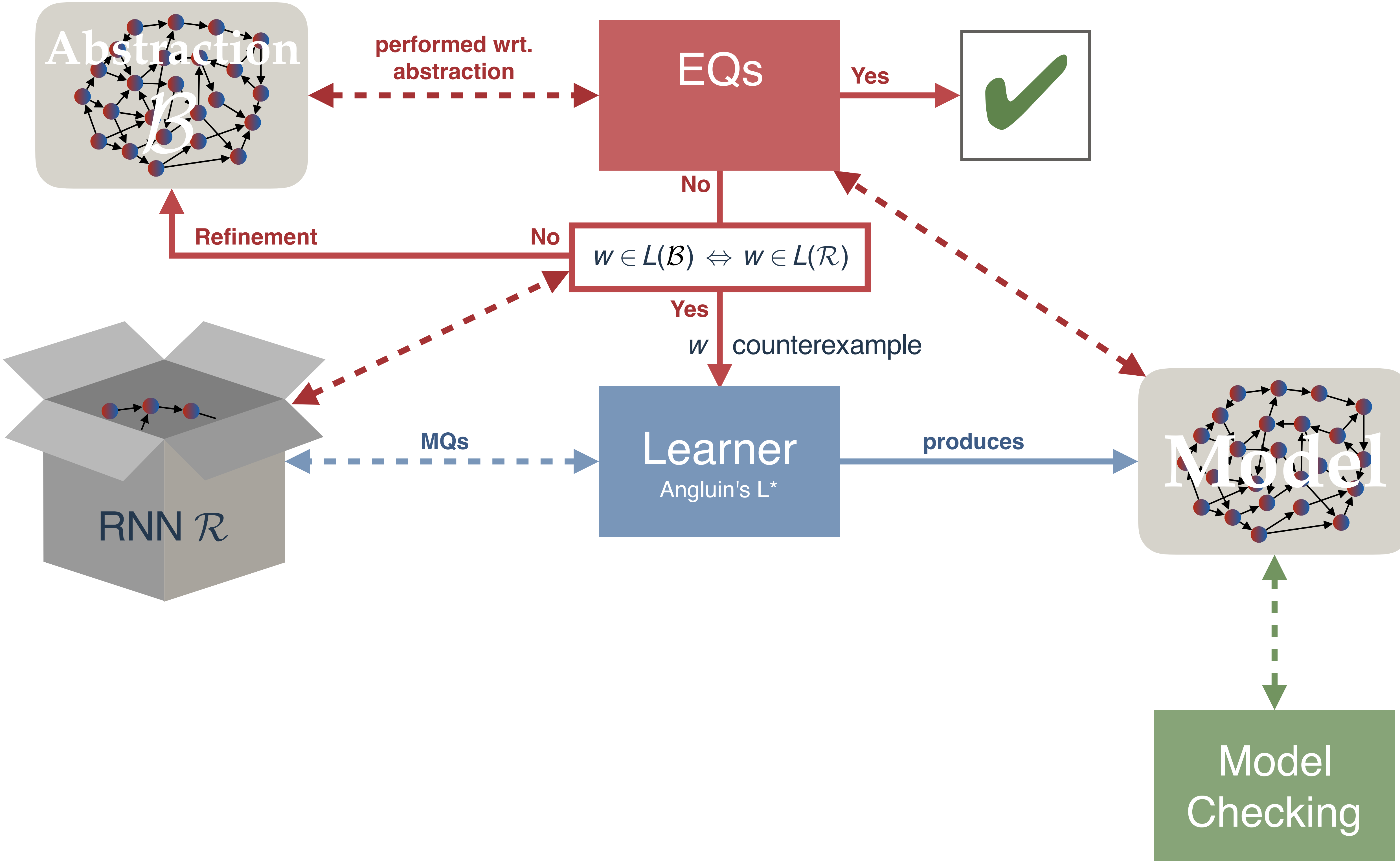
[Weiss, Goldberg, Yahav: *Extracting Automata from Recurrent Neural Networks Using Queries and Counterexamples*. ICML 2018]



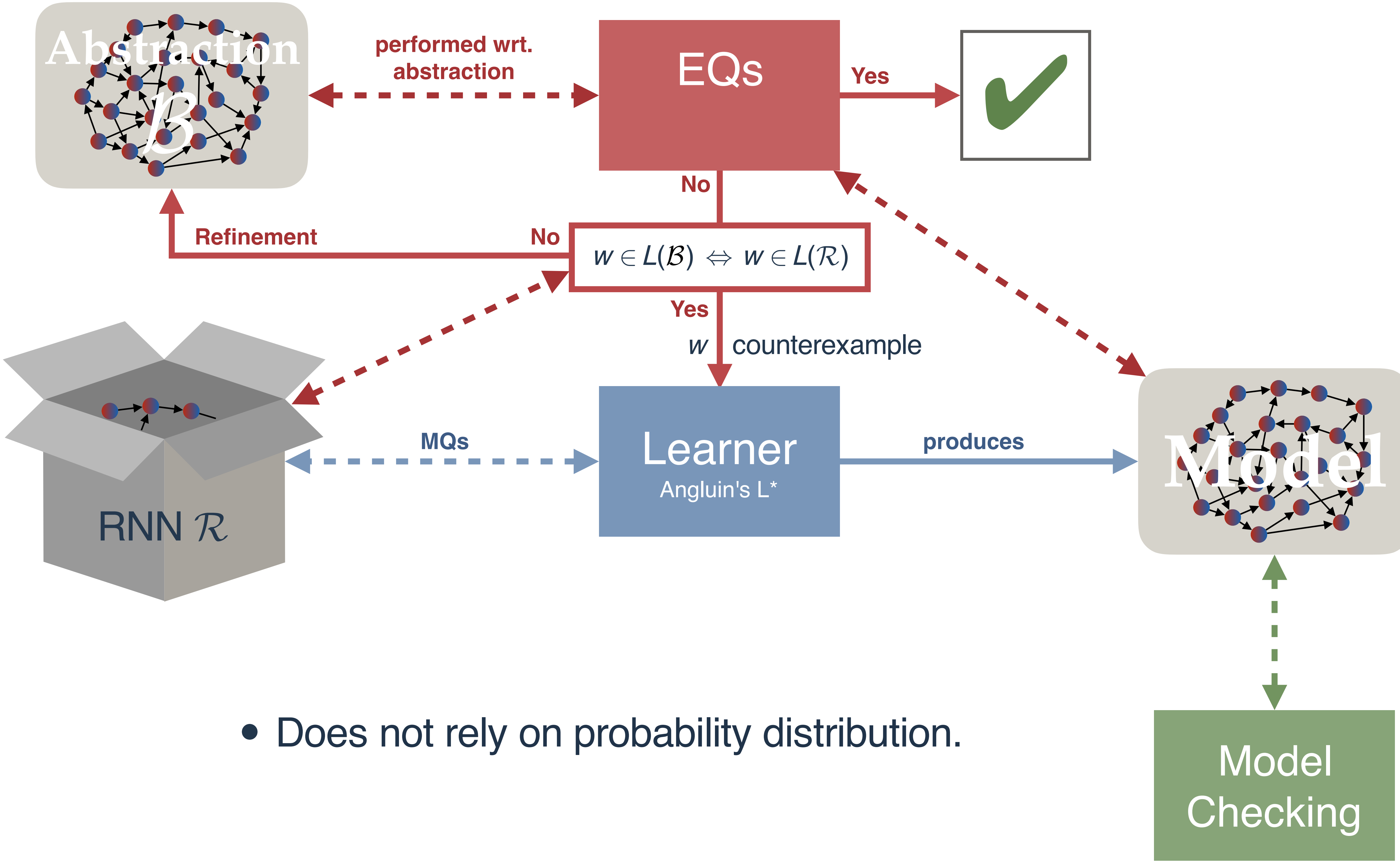
[Weiss, Goldberg, Yahav: *Extracting Automata from Recurrent Neural Networks Using Queries and Counterexamples*. ICML 2018]



[Weiss, Goldberg, Yahav: *Extracting Automata from Recurrent Neural Networks Using Queries and Counterexamples*. ICML 2018]

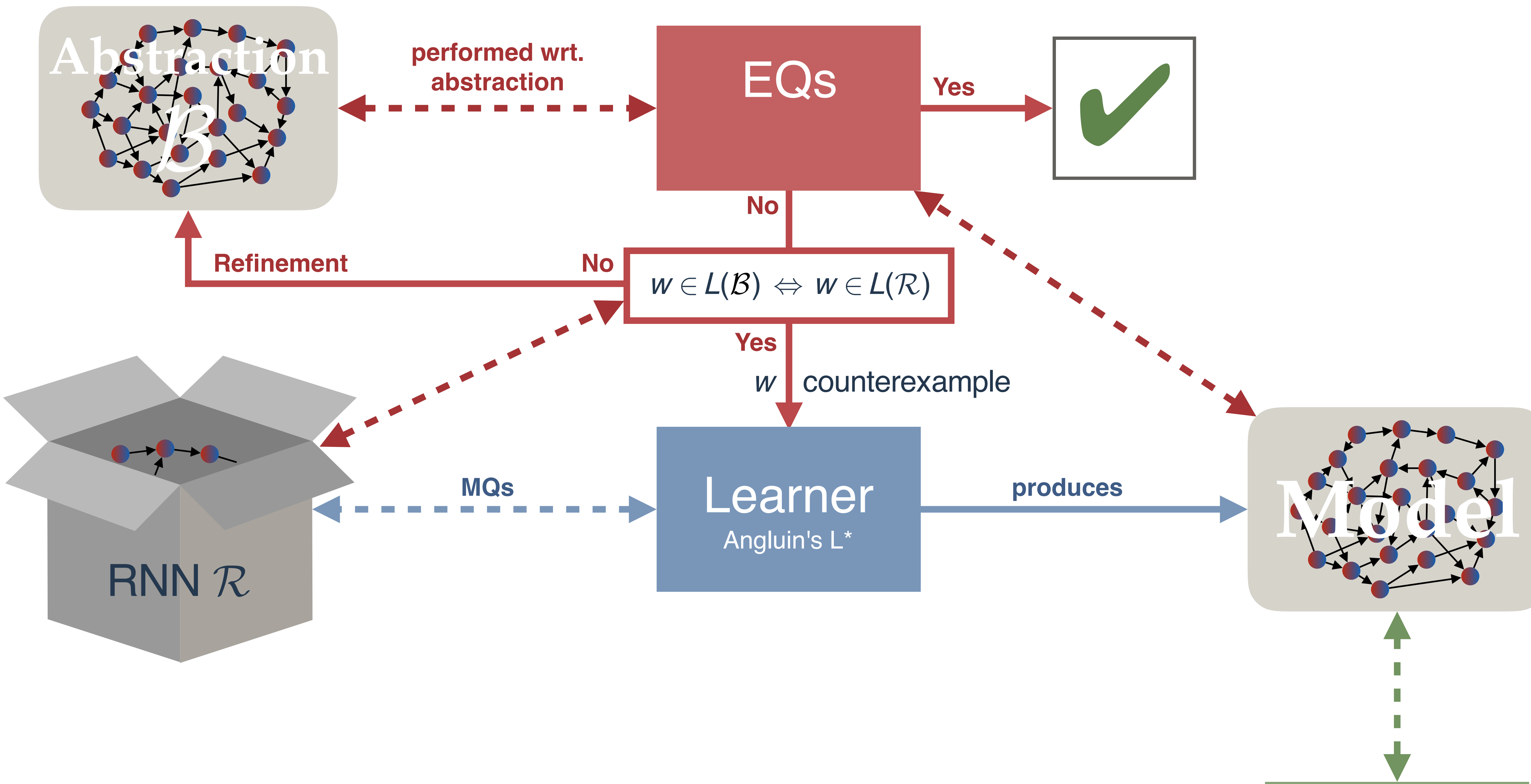


[Weiss, Goldberg, Yahav: *Extracting Automata from Recurrent Neural Networks Using Queries and Counterexamples*. ICML 2018]



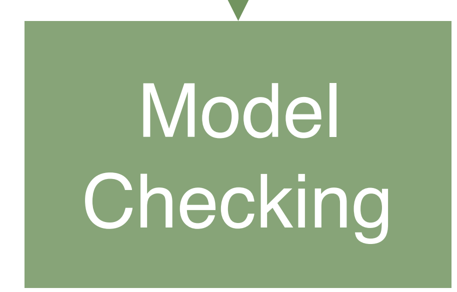
- Does not rely on probability distribution.

[Weiss, Goldberg, Yahav: *Extracting Automata from Recurrent Neural Networks Using Queries and Counterexamples*. ICML 2018]



- Does not rely on probability distribution.
- But counterexample may be spurious.

  Depends on the quality of the abstraction.



[Weiss, Goldberg, Yahav: *Extracting Automata from Recurrent Neural Networks Using Queries and Counterexamples*. ICML 2018]

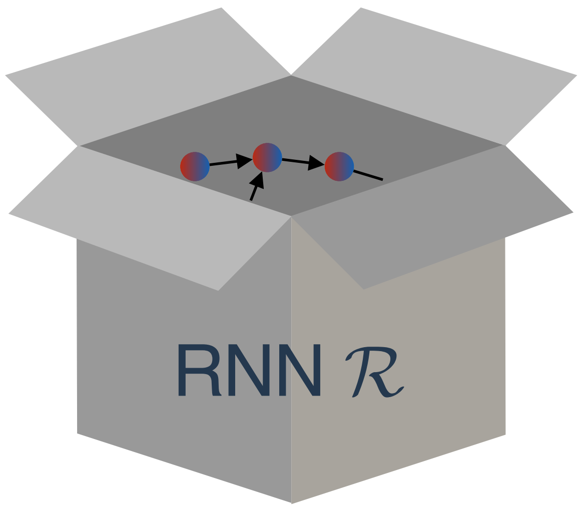
Checked on 30 DFAs / RNNs and 138 specifications:

Type	*Avg time* (s)	*Avg len*	*# Mistakes*	*Avg MQs*
SMC	92	111	**122**	286063
AAMC	444	**7**	30	3701916

Checked on 30 DFAs / RNNs and 138 specifications:

Type	*Avg time* (s)	*Avg len*	# *Mistakes*	*Avg MQs*
SMC	92	111	**122**	286063
AAMC	444	**7**	30	3701916

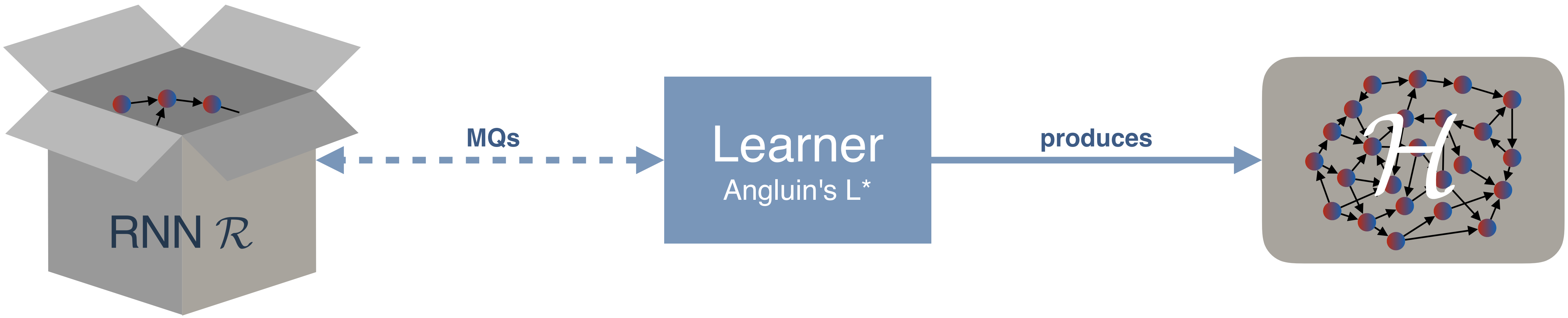
→ Property-directed verification (PDV)



inspired by black-box checking: interweave learning and model checking

[Peled, Vardi, Yannakakis 1999]

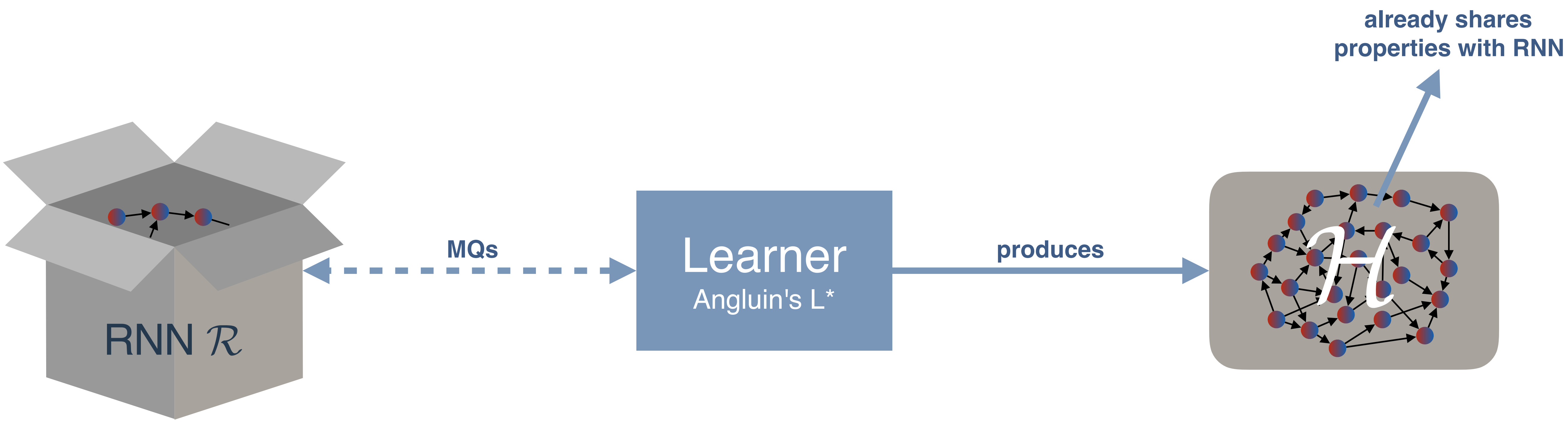
Property-directed verification (PDV)



inspired by black-box checking: interweave learning and model checking

[Peled, Vardi, Yannakakis 1999]

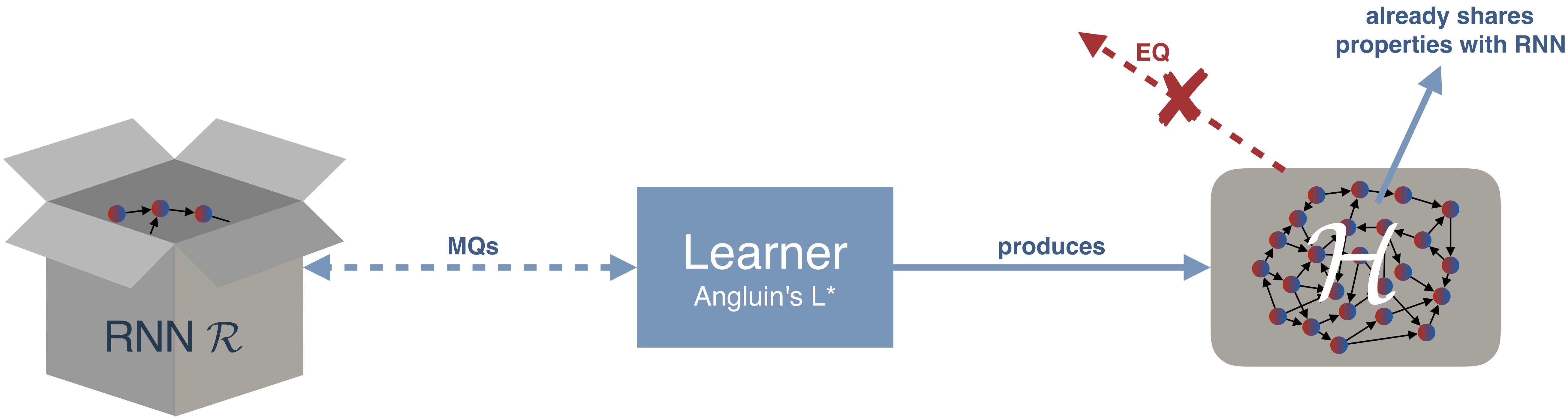
Property-directed verification (PDV)



inspired by black-box checking: interweave learning and model checking

[Peled, Vardi, Yannakakis 1999]

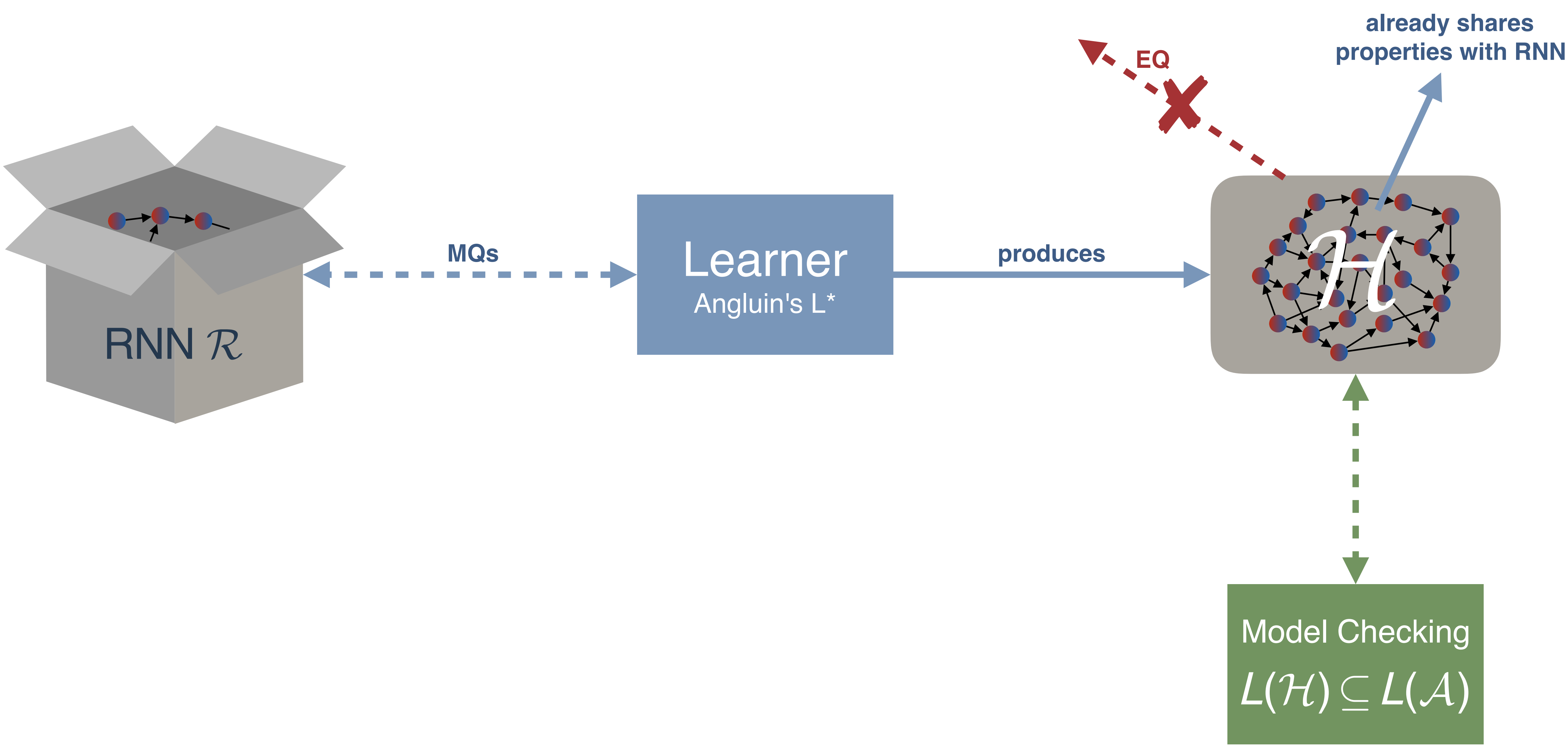
Property-directed verification (PDV)



inspired by black-box checking: interweave learning and model checking

[Peled, Vardi, Yannakakis 1999]

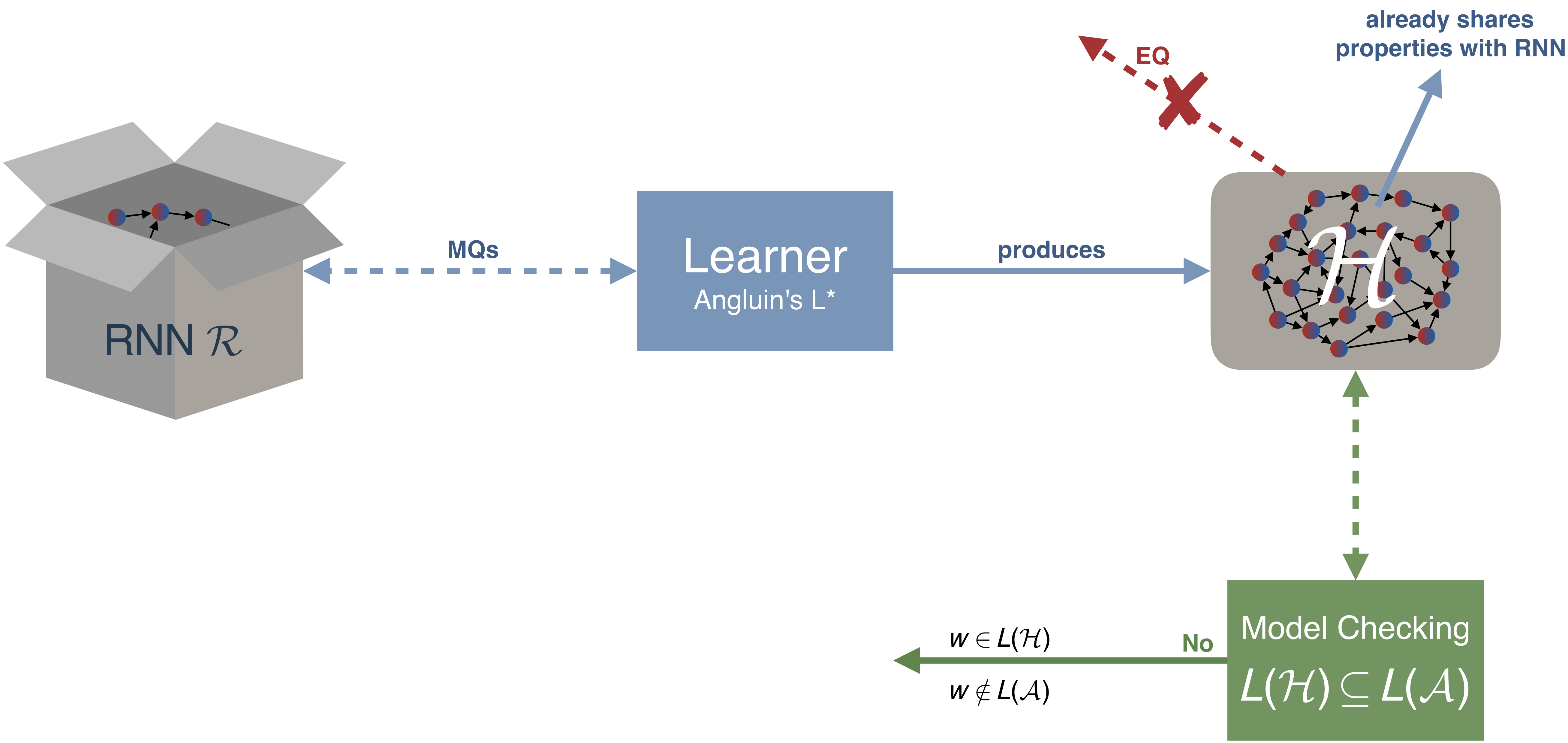
Property-directed verification (PDV)



inspired by black-box checking: interweave learning and model checking

[Peled, Vardi, Yannakakis 1999]

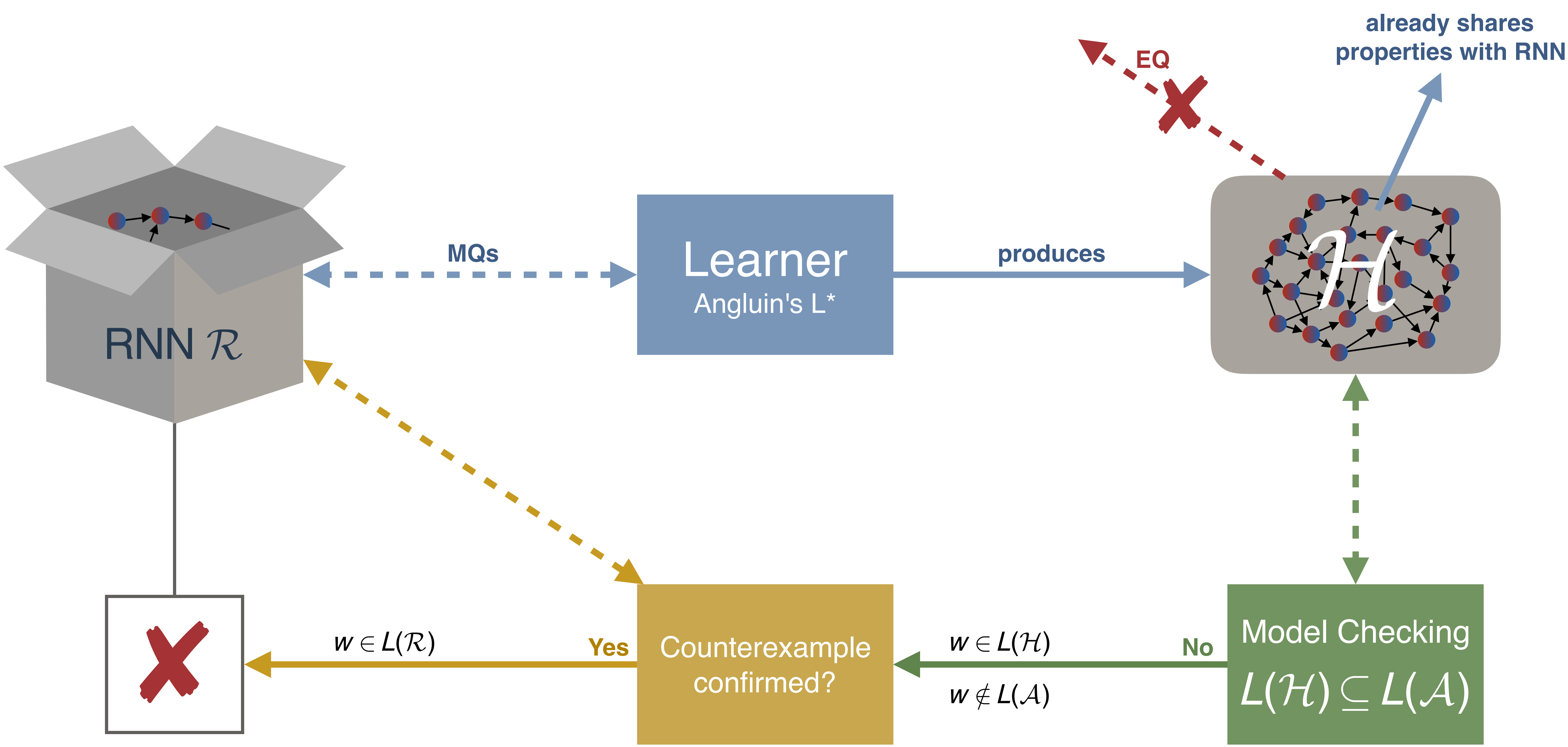
Property-directed verification (PDV)



inspired by black-box checking: interweave learning and model checking

[Peled, Vardi, Yannakakis 1999]

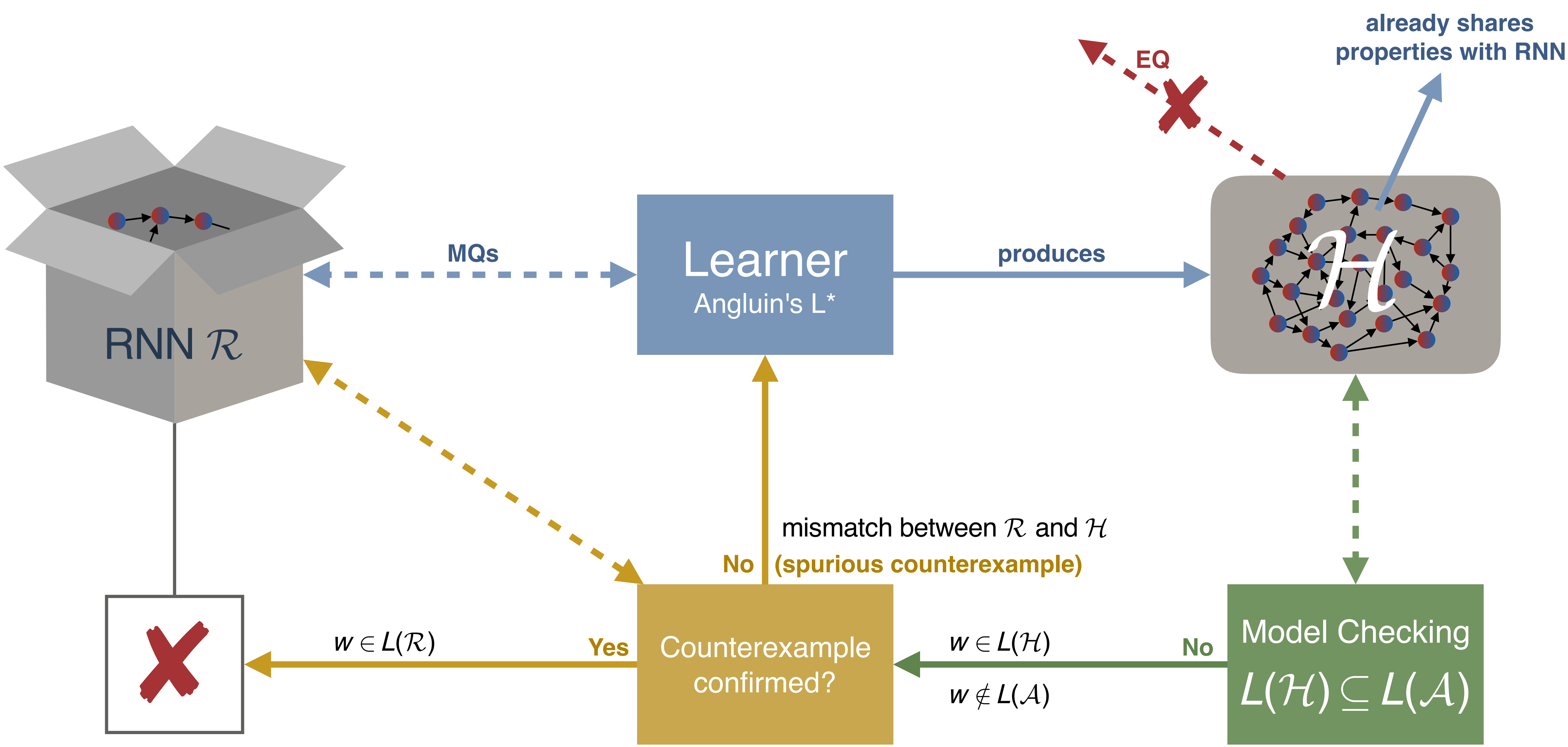
Property-directed verification (PDV)



inspired by black-box checking: interweave learning and model checking

[Peled, Vardi, Yannakakis 1999]

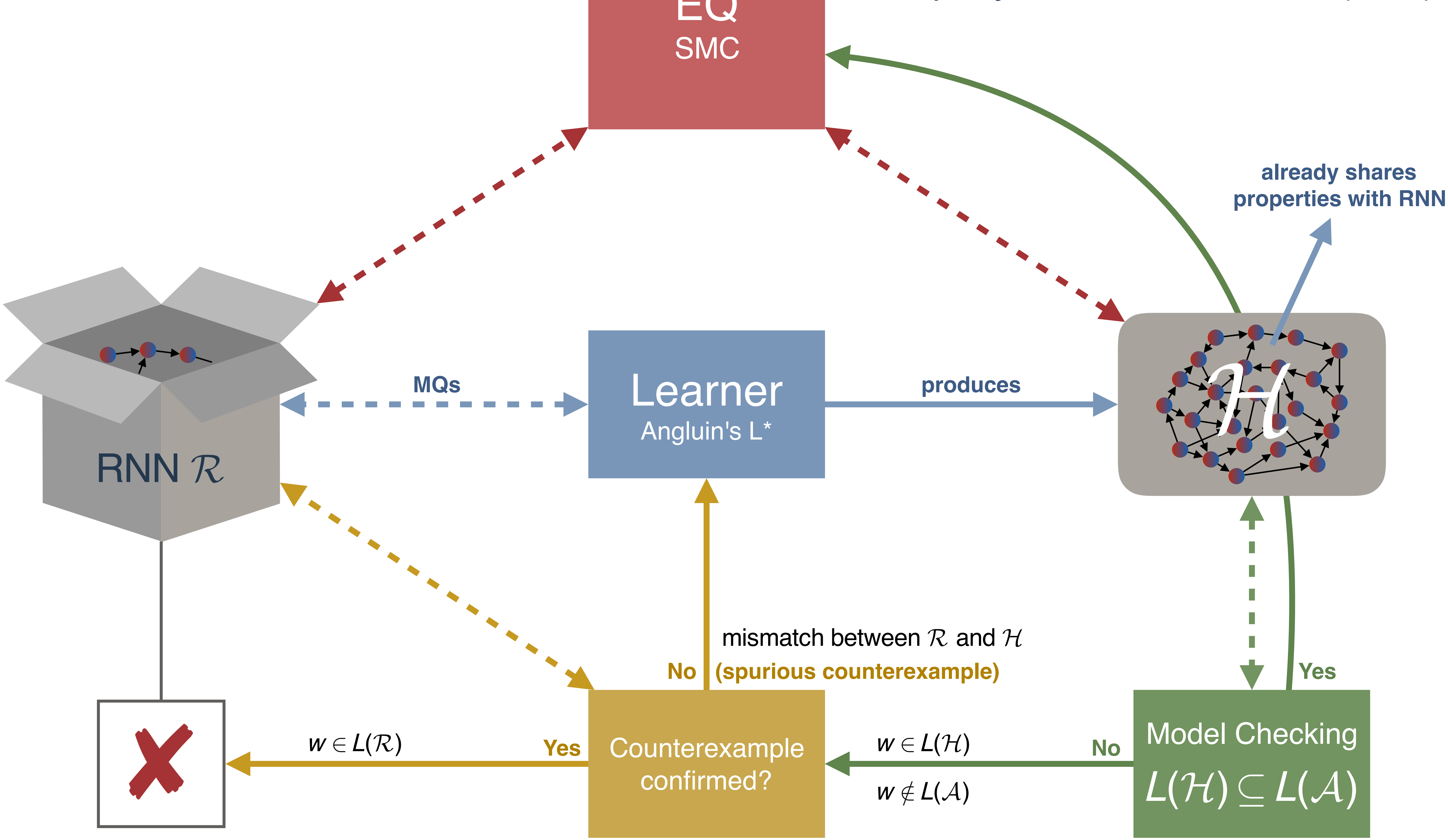
Property-directed verification (PDV)



inspired by black-box checking: interweave learning and model checking

[Peled, Vardi, Yannakakis 1999]

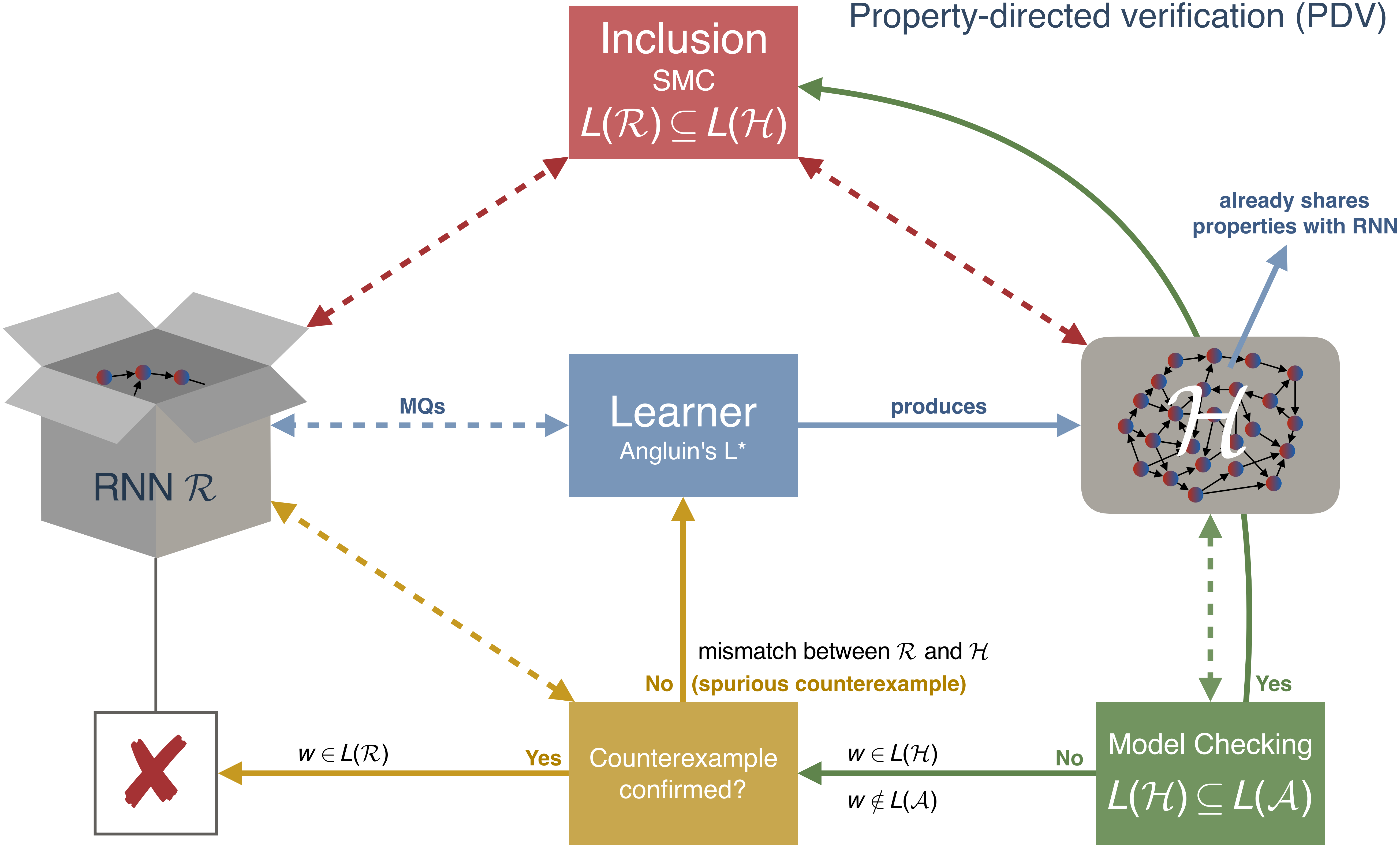
Property-directed verification (PDV)



inspired by black-box checking: interweave learning and model checking

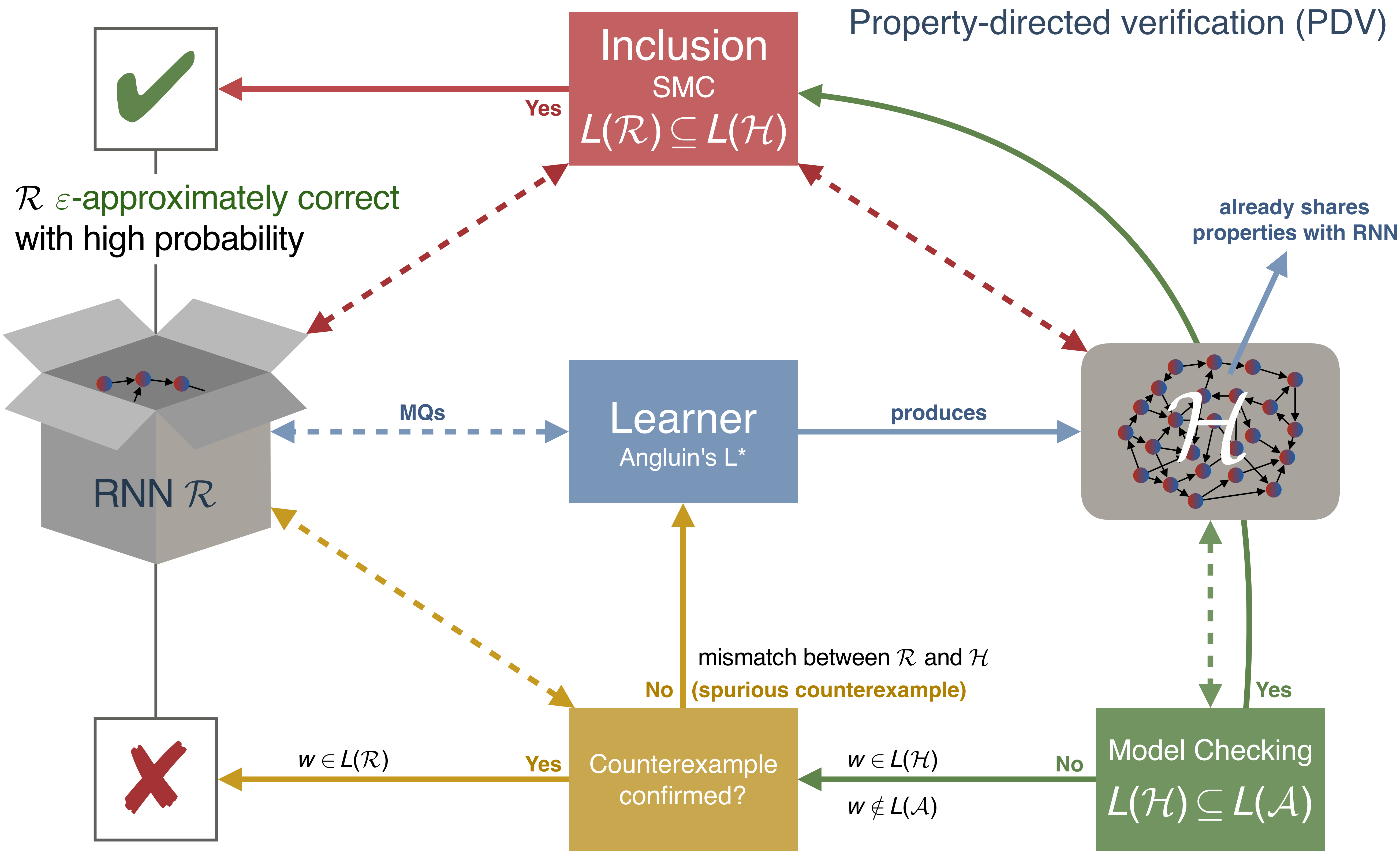
[Peled, Vardi, Yannakakis 1999]

Property-directed verification (PDV)



inspired by black-box checking: interweave learning and model checking

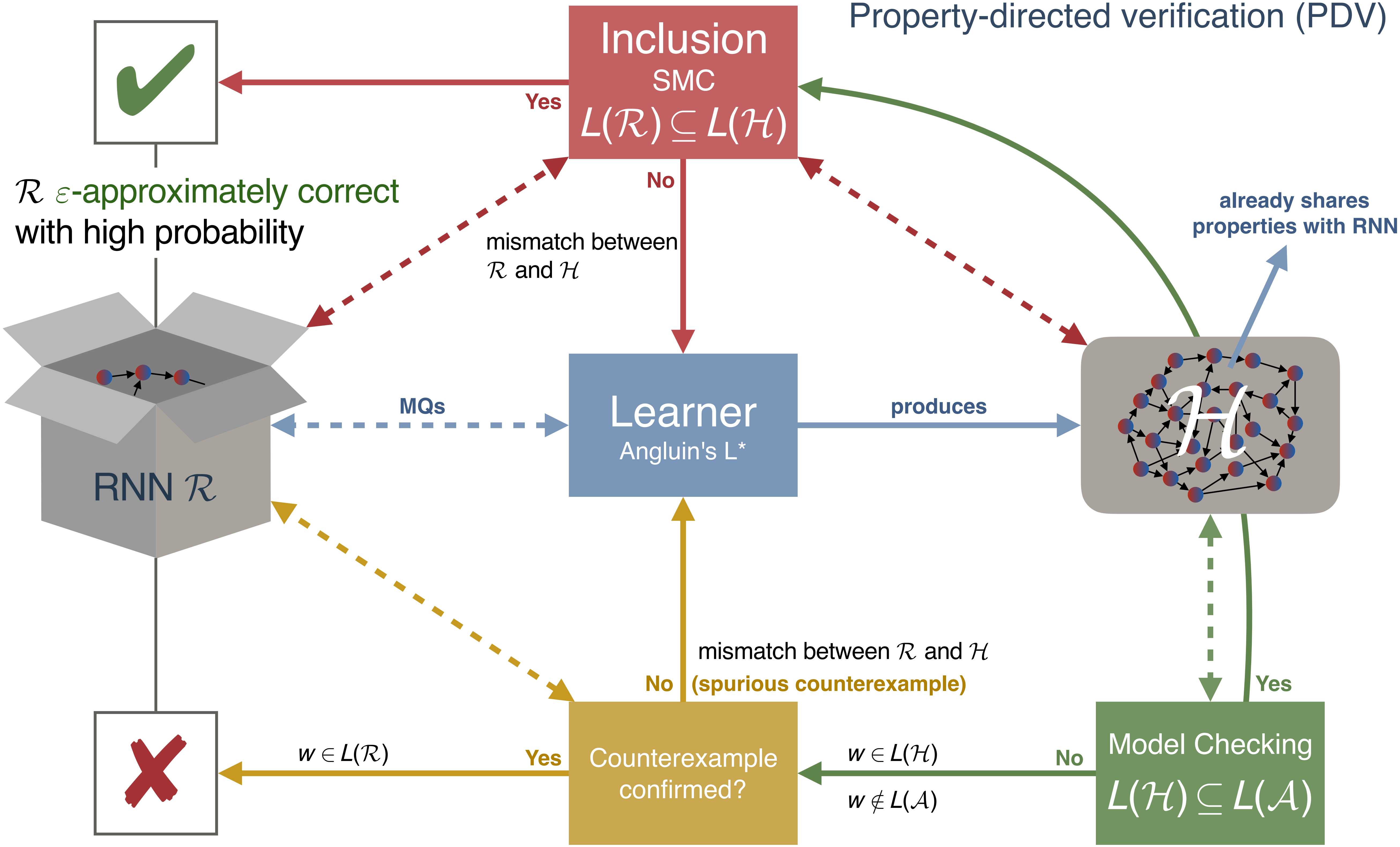
[Peled, Vardi, Yannakakis 1999]



inspired by black-box checking: interweave learning and model checking

[Peled, Vardi, Yannakakis 1999]

Property-directed verification (PDV)



inspired by black-box checking: interweave learning and model checking

[Peled, Vardi, Yannakakis 1999]

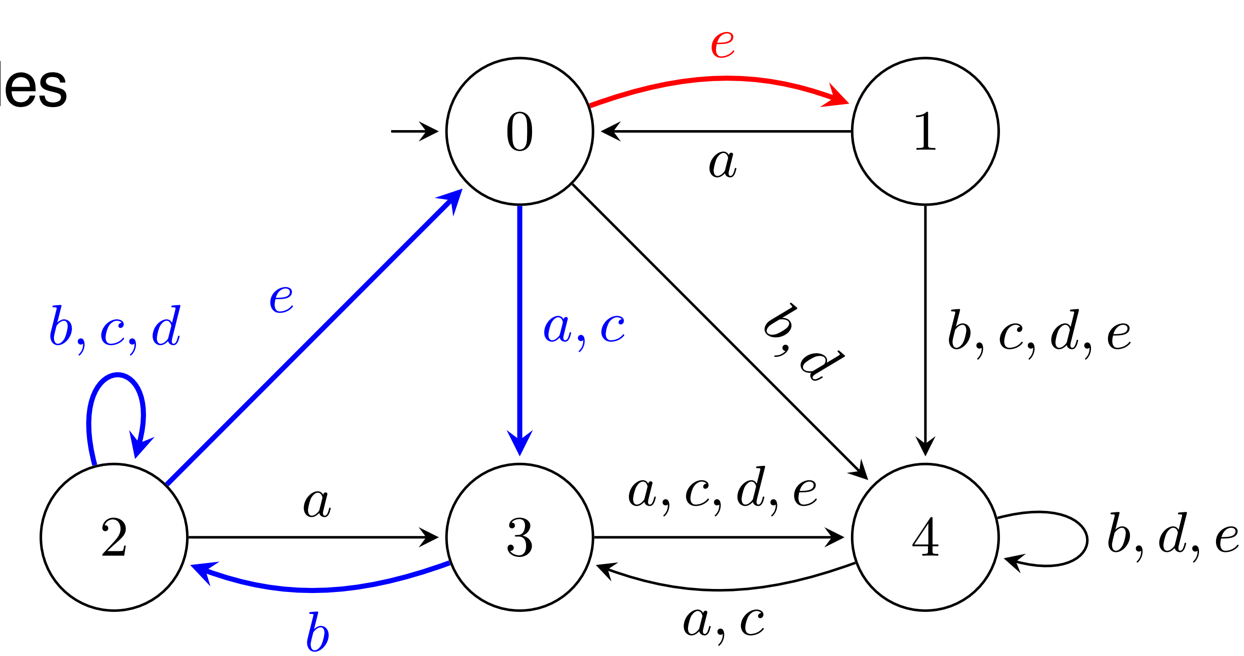
Checked on 30 DFAs / RNNs and 138 specifications:

Type	*Avg time* (s)	*Avg len*	# *Mistakes*	*Avg MQs*
SMC	92	111	**122**	286063
AAMC	444	**7**	30	3701916
PDV	**21**	11	109	**28318**

Checked on 30 DFAs / RNNs and 138 specifications:

Type	*Avg time* (s)	*Avg len*	*# Mistakes*	*Avg MQs*
SMC	92	111	**122**	286063
AAMC	444	**7**	30	3701916
PDV	**21**	11	109	**28318**

PDV detected "faulty flows"
in 81/ 109 of the counterexamples



Conclusion

Conclusion

1. Extraction of finite-state information from RNNs contributes to their understanding and verification.

Conclusion

1. Extraction of finite-state information from RNNs contributes to their understanding and verification.
2. Property-directed verification (PDV) is able to find short counterexamples fast.

Conclusion

1. Extraction of finite-state information from RNNs contributes to their understanding and verification.
2. Property-directed verification (PDV) is able to find short counterexamples fast.

Conclusion and Outlook

1. Practical applications of PDV?

Conclusion

1. Extraction of finite-state information from RNNs contributes to their understanding and verification.
2. Property-directed verification (PDV) is able to find short counterexamples fast.

Conclusion and Outlook

1. Practical applications of PDV?
2. Verification of RNN-based agent environment systems?

Conclusion

1. Extraction of finite-state information from RNNs contributes to their understanding and verification.
2. Property-directed verification (PDV) is able to find short counterexamples fast.

Conclusion and Outlook

1. Practical applications of PDV?
2. Verification of RNN-based agent environment systems?
3. Going beyond regular specifications: Inference of context-free languages.

Conclusion

1. Extraction of finite-state information from RNNs contributes to their understanding and verification.
2. Property-directed verification (PDV) is able to find short counterexamples fast.

Conclusion and Outlook

1. Practical applications of PDV?
2. Verification of RNN-based agent environment systems?
3. Going beyond regular specifications: Inference of context-free languages.

Thank you!