Certificates for automata in a hostile environment

Sebastian Muskalla

May 11, 2023

PhD defense

Table of contents

Certificates for automata
in a hostile environment

1

Certificates for automata
in a hostile environment

② ①

Certificates for automata

in a hostile environment

The title

② ①

Certificates for automata
in a hostile environment

③

Automata theory

Automata theory

Theoretical computer science:

Which problems can be solved by computers in principle?

Automata theory

Theoretical computer science:

Which problems can be solved by computers in principle?

Concept of self-application

Theoretical computer science:

Which problems can be solved by computers in principle?

Concept of self-application

Study verification:

Which problems about computer (programs)
can be solved by computer (programs)?

Verification problem

Verification problem for specification φ

Given: Program P.

Question: Does behavior of P satisfy φ, $P \models \varphi$?

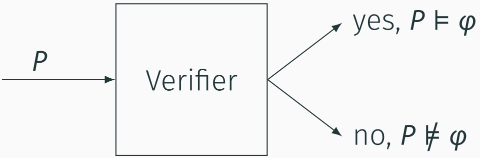
Verification problem

Verification problem for specification φ

Given: Program P.

Question: Does behavior of P satisfy φ, $P \models \varphi$?

Automated verification:



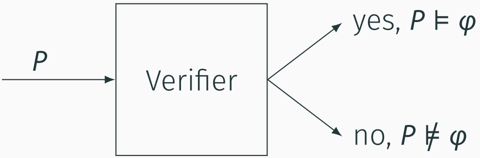
Verification problem

Verification problem for specification φ

Given: Program P.

Question: Does behavior of P satisfy φ, $P \models \varphi$?

Automated verification:



Theorem ([Church 1935/36, Turing 1936])

The verification problem is undecidable for some specification.

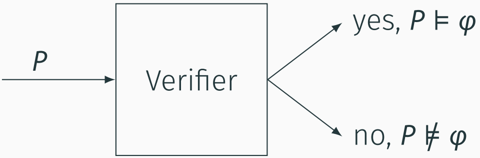
Verification problem

Verification problem for specification φ

Given: Program P.

Question: Does behavior of P satisfy φ, $P \models \varphi$?

Automated verification:



Theorem ([Church 1935/36, Turing 1936, Rice 1953])

The verification problem is undecidable for all specifications.

Theorem ([Church 1935/36, Turing 1936, Rice 1953])

The verification problem is undecidable for all specifications.

Two loopholes exist:

Theorem ([Church 1935/36, Turing 1936, Rice 1953])

The verification problem is undecidable for all specifications.

Two loopholes exist:

1. Problem is just undecidable in full generality
   - We may be able to verify some programs
     (We come back to this later)

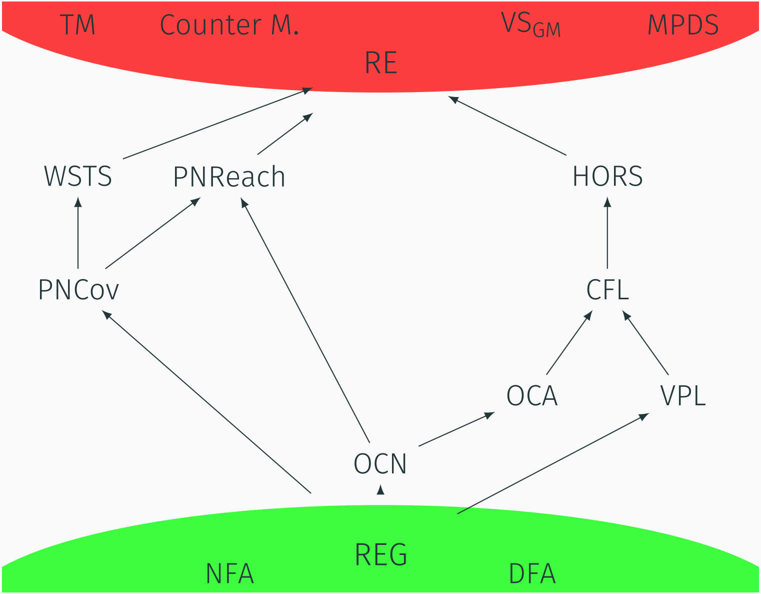
Theorem ([Church 1935/36, Turing 1936, Rice 1953])

The verification problem is undecidable for all specifications.

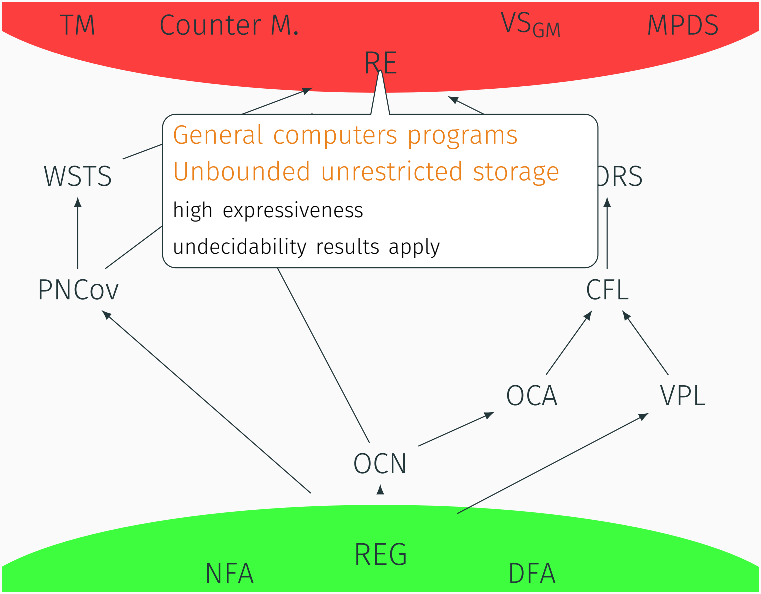
Two loopholes exist:

1. Problem is just undecidable in full generality
   - We may be able to verify some programs
     (We come back to this later)
2. Problem undecidable if input are general computer programs
   - Study restricted computer models: Automata

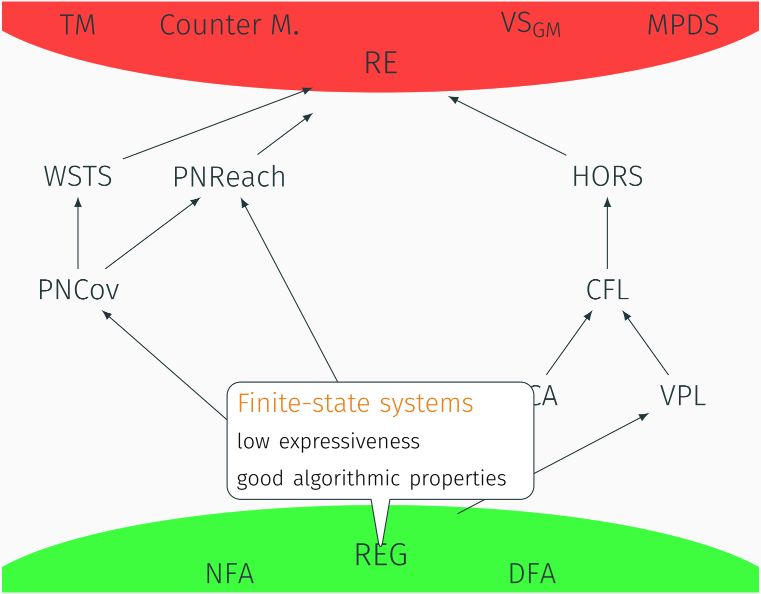
Automata theory



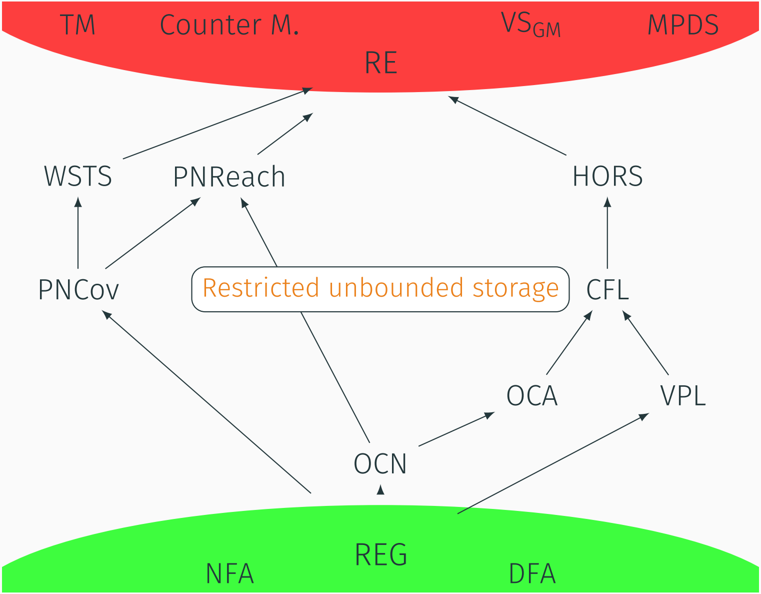
Automata theory



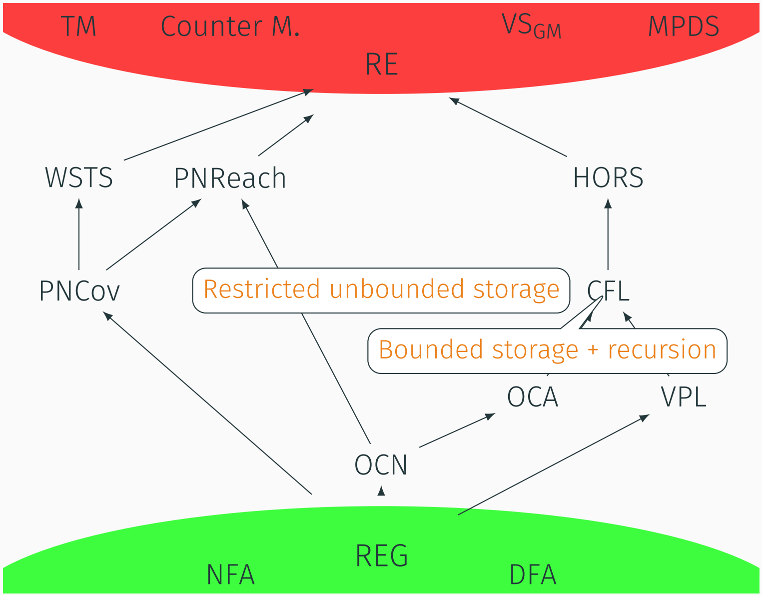
Automata theory



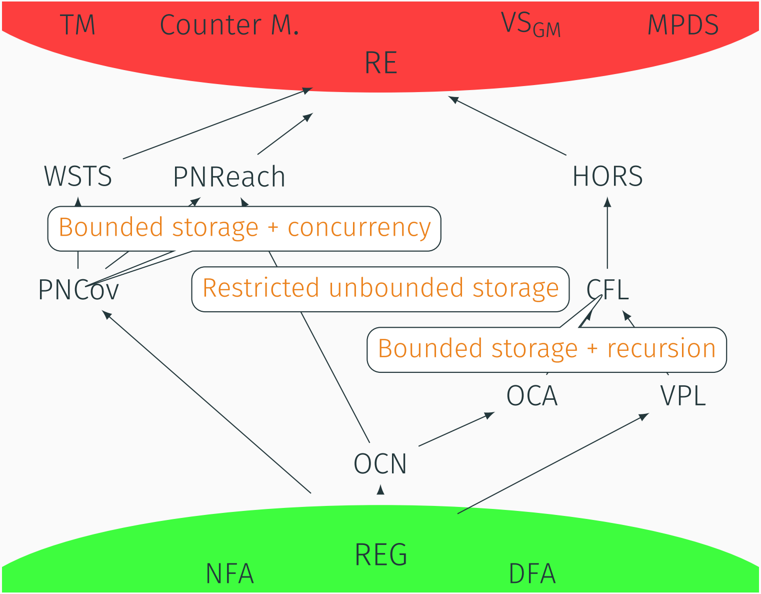
Automata theory



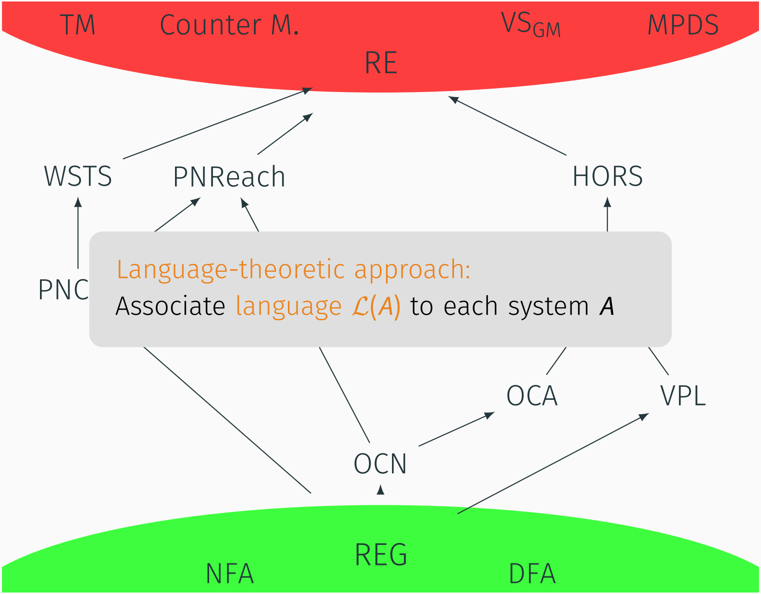
Automata theory



Automata theory



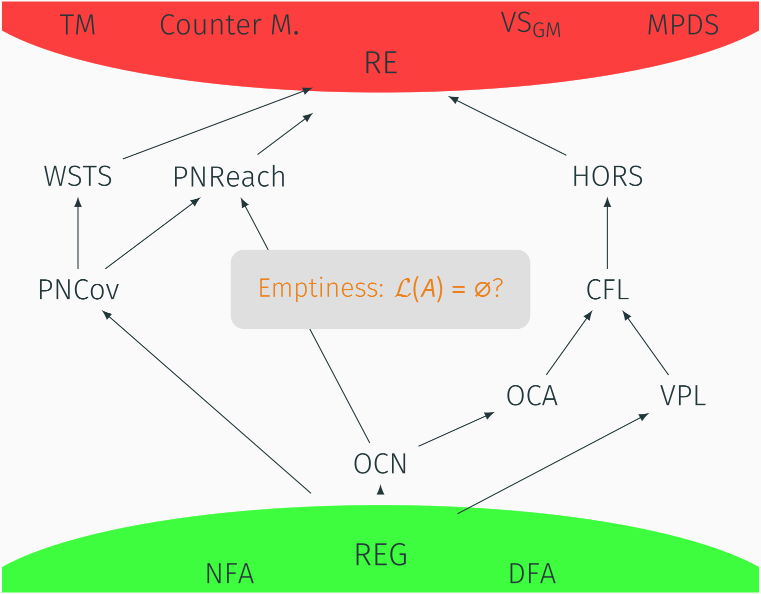
Automata theory



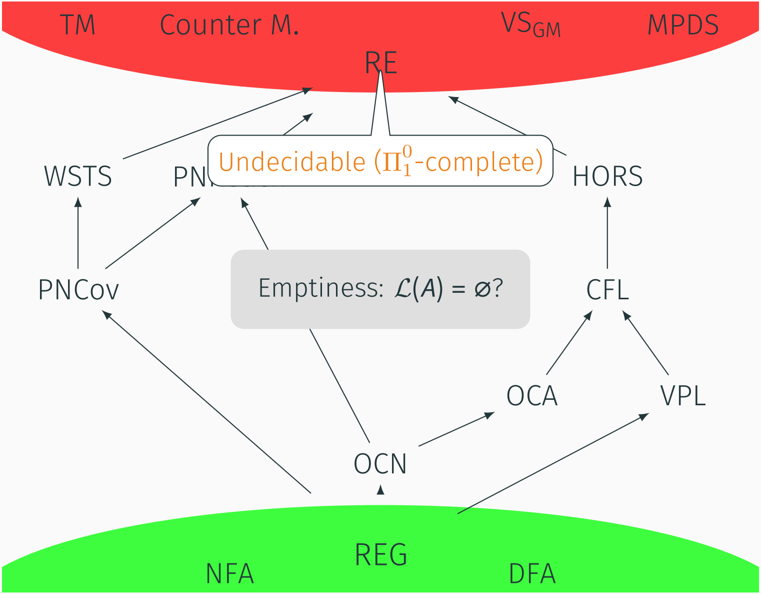
Language-theoretic approach:

Associate language $\mathcal{L}(A)$ to each system A

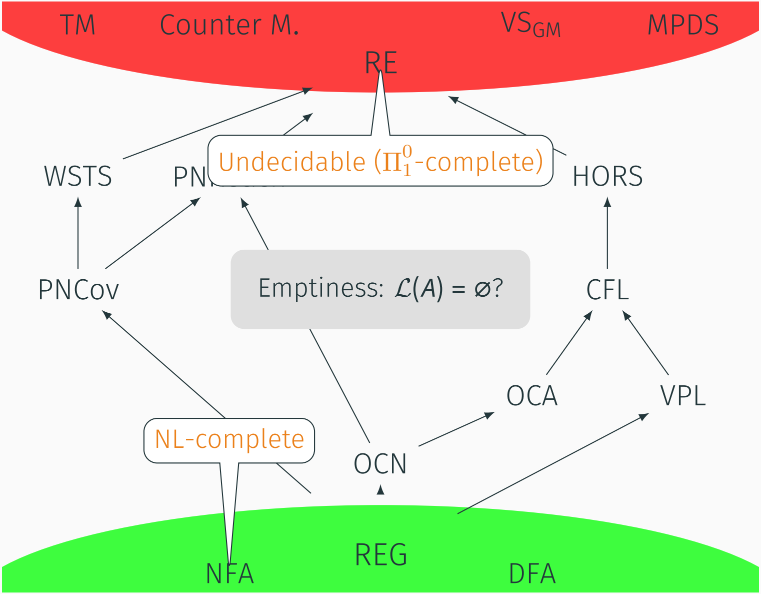
Automata theory



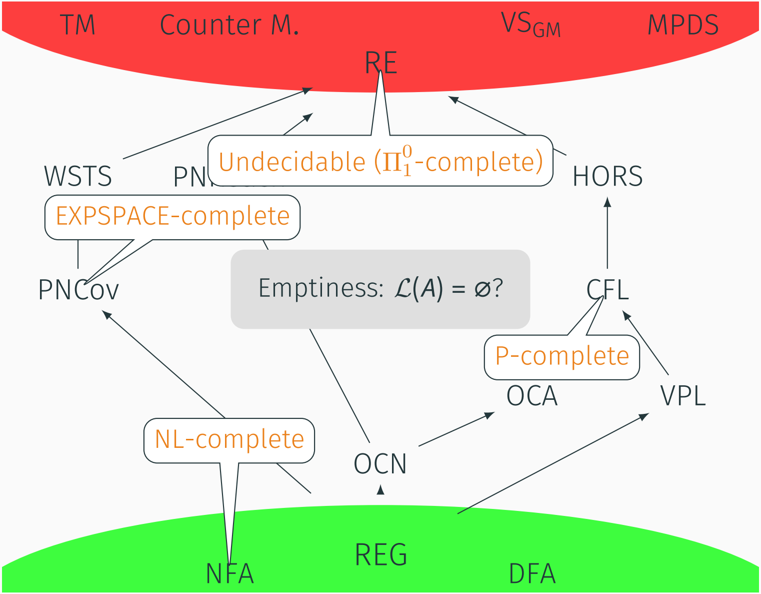
Automata theory



Automata theory



Automata theory



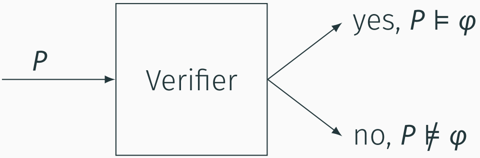
Automata theory

Verification problems may be decidable if we consider automata as input

Automata theory

Verification problems may be decidable if we consider automata as input

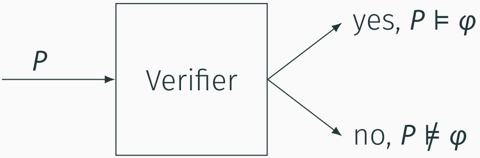
How to solve general verification problems?



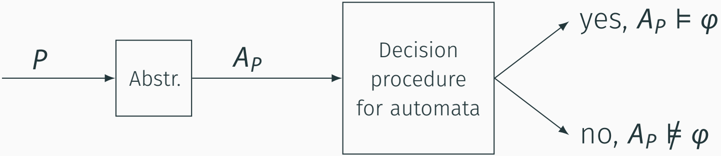
Automata theory

Verification problems may be decidable if we consider automata as input

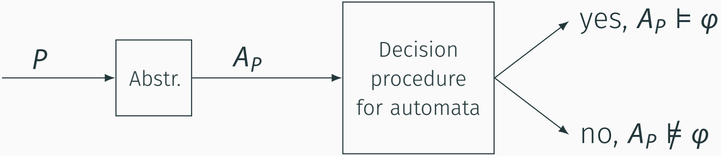
How to solve general verification problems?



Abstract to an automaton first!

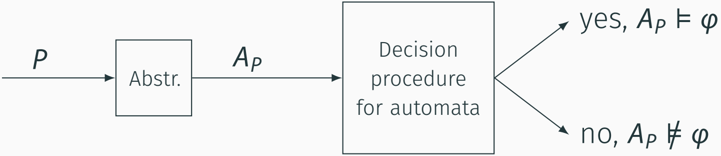


Automata theory



Does this always work?

Automata theory

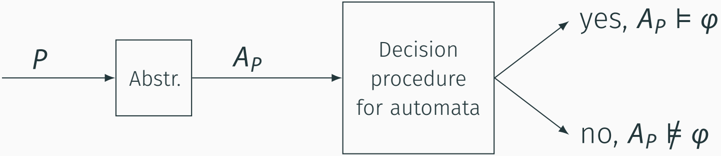


Does this always work?

NO!

Need to pick abstraction carefully

Automata theory



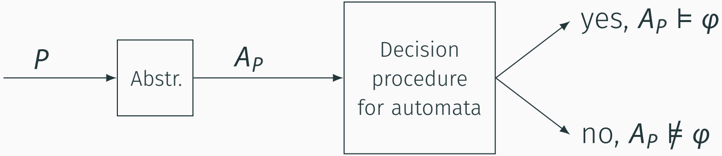
Does this always work?

NO!

Need to pick abstraction carefully

– Verification problem needs to be (efficiently) decidable

Automata theory



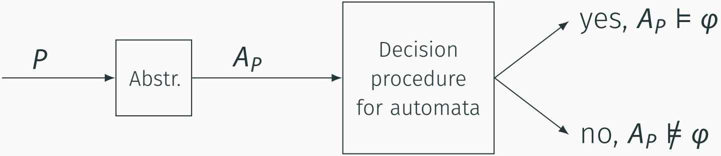
Does this always work?

NO!

Need to pick abstraction carefully

- Verification problem needs to be (efficiently) decidable
- Expressiveness needs to be high enough so that we can model the behavior relevant to the specification

Automata theory



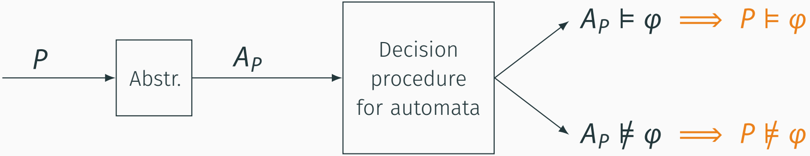
Does this always work?

NO!

Need to pick abstraction carefully

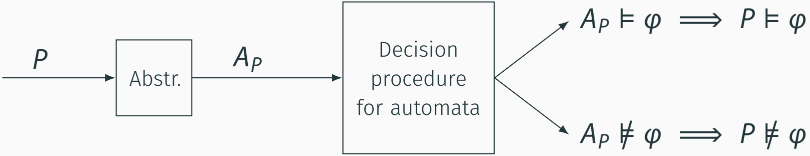
- Verification problem needs to be (efficiently) decidable
- Expressiveness needs to be high enough so that we can model the behavior relevant to the specification
- Need some relation between P and A_P, e.g. overapproximation: $\mathcal{L}(P) \subseteq \mathcal{L}(A_P)$

The automata-theoretic approach to verification



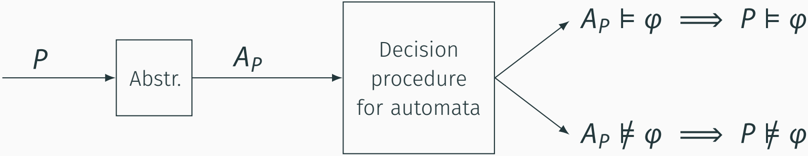
Certificates

Certificates



This is too optimistic!

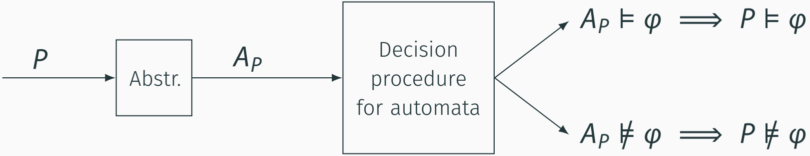
Certificates



This is too optimistic!

Problem: We assume that a boolean (yes/no) answer to the decision problem is sufficient

Certificates

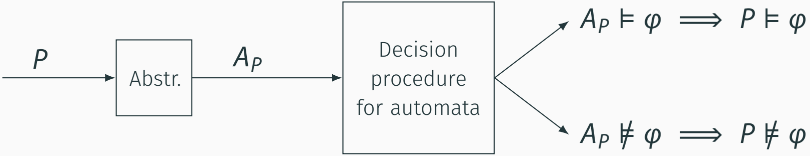


This is too optimistic!

Problem: We assume that a boolean (yes/no) answer to the decision problem is sufficient

Need more detailed output!

Certificates



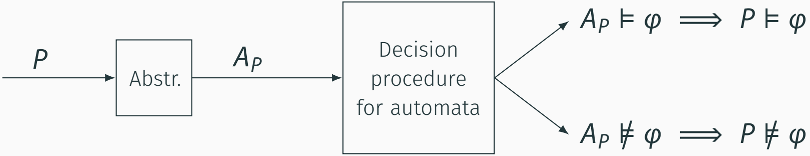
This is too optimistic!

Problem: We assume that a boolean (yes/no) answer to the decision problem is sufficient

Need more detailed output!

- Accountability: We don't want to trust the algorithm blindly

Certificates



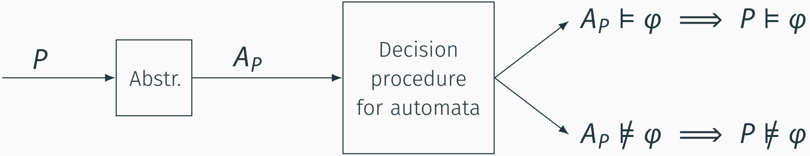
This is too optimistic!

Problem: We assume that a boolean (yes/no) answer to the decision problem is sufficient

Need more detailed output!

- Accountability: We don't want to trust the algorithm blindly
- We often need more than one call of a decision procedure

Certificates



This is too optimistic!

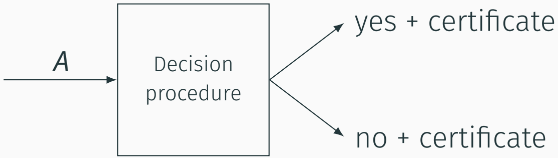
Problem: We assume that a boolean (yes/no) answer to the decision problem is sufficient

Need more detailed output!

- Accountability: We don't want to trust the algorithm blindly
- We often need more than one call of a decision procedure
- Later calls need information computed by earlier ones
  e.g. compositional verification, refinement loops (CEGAR)

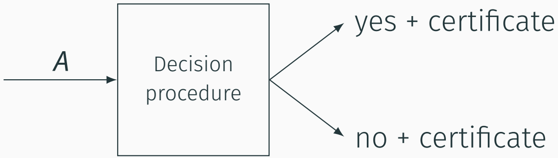
Certificates

We need algorithms that also compute certificates



Certificates

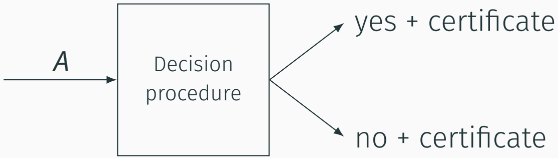
We need algorithms that also compute certificates



A certificate is additional information justifying the boolean answer

Certificates

We need algorithms that also compute certificates

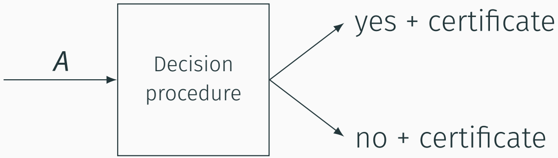


A certificate is additional information justifying the boolean answer

A certificate can be used to check the correctness of the answer

Certificates

We need algorithms that also compute certificates



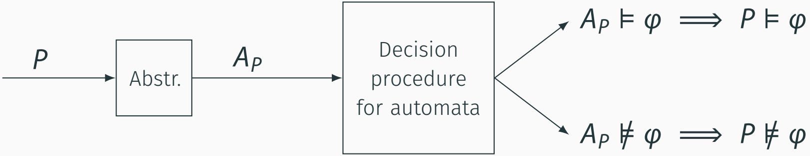
A certificate is additional information justifying the boolean answer

A certificate can be used to check the correctness of the answer

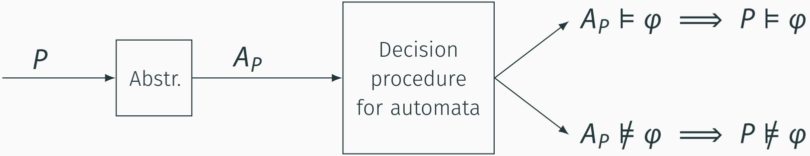
This check should be easier than the original computation

The (hostile) environment

The (hostile) environment

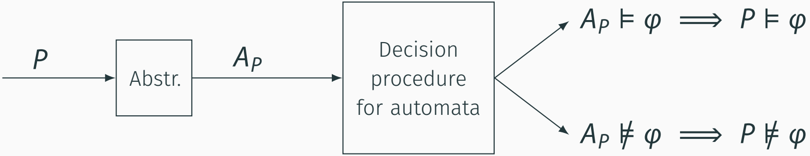


The (hostile) environment



When abstracting P into A_P, we usually forget a part of the system

The (hostile) environment

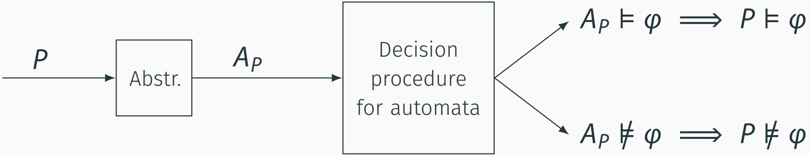


When abstracting P into A_P, we usually forget a part of the system

Example:

- P uses recursion + unbounded storage
- A_P comes from a class that only supports bounded storage
- Solution: Abstract away data
- But: This introduces non-determinism

The (hostile) environment



When abstracting P into A_P, we usually forget a part of the system

Example:

- P uses recursion + unbounded storage
- A_P comes from a class that only supports bounded storage
- Solution: Abstract away data
- But: This introduces non-determinism

This imprecision may affect verification!

The (hostile) environment

The automaton lives A_P in an environment

The (hostile) environment

The automaton lives A_P in an environment

- Parts of system P abstracted away in A_P

The (hostile) environment

The automaton lives A_P in an environment

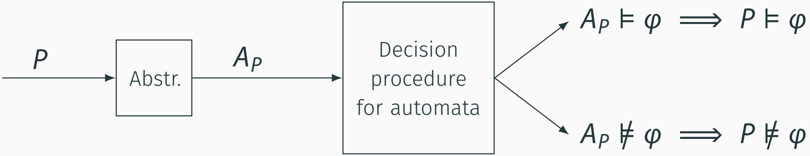
- Parts of system P abstracted away in A_P
- Parts of the system that were never modeled to begin with:
  - User input
  - External components

The (hostile) environment

The automaton lives A_P in an environment

- Parts of system P abstracted away in A_P
- Parts of the system that were never modeled to begin with:
  - User input
  - External components
- Compositional verification
  - Focus on one component
  - Rest of the components becomes the environment

The (hostile) environment



The environment is hostile because when we apply a decision procedure to A_P, it may break the correspondence between

- correctness of A_P $(A_P \models \varphi \;/\; A_P \not\models \varphi)$
- correctness of P $(P \models \varphi \;/\; P \not\models \varphi)$

Certificates for automata in a hostile environment

In order to enable the automata-theoretic approach to verification, we need decision procedures for automata that produce certificates and are equipped to take the (hostile) environment into account.

Certificates for automata in a hostile environment

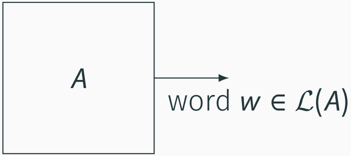
In order to enable the automata-theoretic approach to verification, we need decision procedures for automata that produce certificates and are equipped to take the (hostile) environment into account.

This thesis aims to provide such decision procedures

1st example: Unreliable communication & Language closures

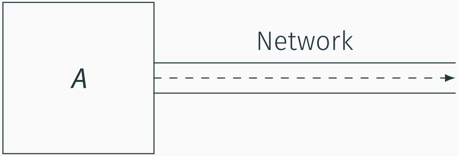
Unreliable communication

Program sending messages



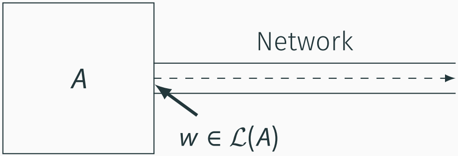
Unreliable communication

Program sending messages over a lossy network connection



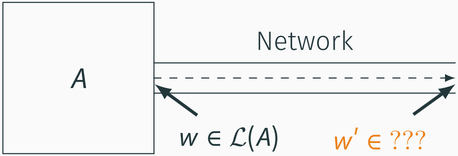
Unreliable communication

Program sending messages over a lossy network connection



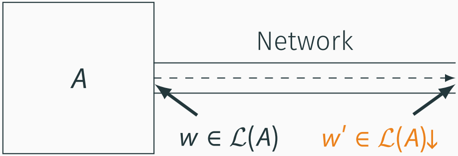
Unreliable communication

Program sending messages over a lossy network connection



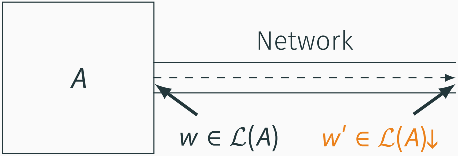
Unreliable communication

Program sending messages over a lossy network connection



Unreliable communication

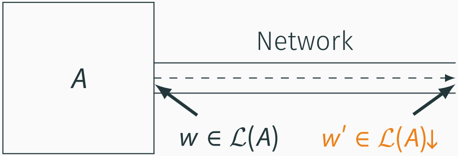
Program sending messages over a lossy network connection



We are typically given a description of A

Unreliable communication

Program sending messages over a lossy network connection

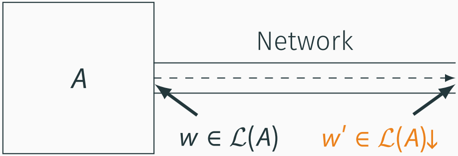


We are typically given a description of A

Specification talks about $\mathcal{L}(A)\downarrow$, the visible behavior of A

Unreliable communication

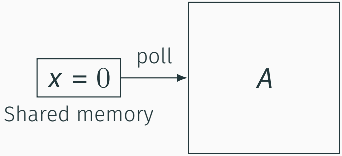
Program sending messages over a lossy network connection



- We are typically given a description of A
- Specification talks about $\mathcal{L}(A)\downarrow$, the visible behavior of A
- Unreliable communication forms an environment that has to be taken into account

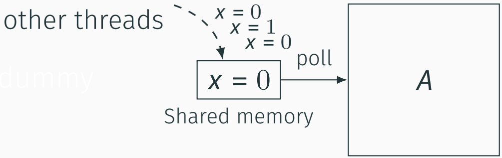
Unreliable communication

Same problem can happen even when communication is reliable



Unreliable communication

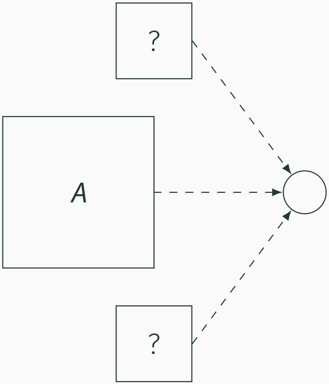
Same problem can happen even when communication is reliable



Thread A sees $\mathcal{L}$(other threads)↓

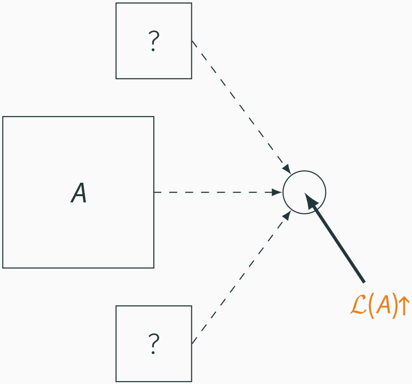
Unreliable communication

Opposite problem: Gaininess



Unreliable communication

Opposite problem: Gaininess



Closures

Environment turns $\mathcal{L}(A)$ into $\mathcal{L}(A)\downarrow$ resp. $\mathcal{L}(A)\uparrow$

How to design a theoretical model?

Closures

Environment turns $\mathcal{L}(A)$ into $\mathcal{L}(A)\downarrow$ resp. $\mathcal{L}(A)\uparrow$

How to design a theoretical model?

Subword ordering: $v \preceq w$ iff v obtained from w by deleting letters

Closures

Environment turns $\mathcal{L}(A)$ into $\mathcal{L}(A)\downarrow$ resp. $\mathcal{L}(A)\uparrow$

How to design a theoretical model?

Subword ordering: $v \preceq w$ iff v obtained from w by deleting letters

RADAR ≼ ABRACADABRA

Closures

Environment turns $\mathcal{L}(A)$ into $\mathcal{L}(A)\!\downarrow$ resp. $\mathcal{L}(A)\!\uparrow$

How to design a theoretical model?

Subword ordering: $v \preceq w$ iff v obtained from w by deleting letters

$$\text{RADAR} \preceq \text{ABRACADABRA}$$

Downward closure: $\mathcal{L}(A)\!\downarrow = \{v \mid \exists w \in \mathcal{L}(A) \colon v \preceq w\}$ (Lossiness)

Upward closure: $\mathcal{L}(A)\!\uparrow = \{v \mid \exists w \in \mathcal{L}(A) \colon w \preceq v\}$ (Gaininess)

Closures

Environment turns $\mathcal{L}(A)$ into $\mathcal{L}(A)\downarrow$ resp. $\mathcal{L}(A)\uparrow$

How to design a theoretical model?

Subword ordering: $v \preceq w$ iff v obtained from w by deleting letters

RADAR $\preceq$ ABRACADABRA

Downward closure: $\mathcal{L}(A)\downarrow = \{v \mid \exists w \in \mathcal{L}(A) \colon v \preceq w\}$ (Lossiness)

Upward closure: $\mathcal{L}(A)\uparrow = \{v \mid \exists w \in \mathcal{L}(A) \colon w \preceq v\}$ (Gaininess)

Theorem ([Haines 1969],[Abdulla et al. 2004])

$\mathcal{L}(A)\downarrow$, $\mathcal{L}(A)\uparrow$ *always simply regular.*

Closures

Environment turns $\mathcal{L}(A)$ into $\mathcal{L}(A)\downarrow$ resp. $\mathcal{L}(A)\uparrow$

How to design a theoretical model?

Subword ordering: $v \preceq w$ iff v obtained from w by deleting letters

RADAR $\preceq$ ABRACADABRA

Downward closure: $\mathcal{L}(A)\downarrow = \{v \mid \exists w \in \mathcal{L}(A) \colon v \preceq w\}$ (Lossiness)
Upward closure: $\mathcal{L}(A)\uparrow = \{v \mid \exists w \in \mathcal{L}(A) \colon w \preceq v\}$ (Gaininess)

Theorem ([Haines 1969],[Abdulla et al. 2004])

$\mathcal{L}(A)\downarrow$, $\mathcal{L}(A)\uparrow$ always simply regular.

Regular languages can be represented by finite automata

Closures

Environment turns $\mathcal{L}(A)$ into $\mathcal{L}(A)\downarrow$ resp. $\mathcal{L}(A)\uparrow$

How to design a theoretical model?

Subword ordering: $v \preceq w$ iff v obtained from w by deleting letters

RADAR $\preceq$ ABRACADABRA

Downward closure: $\mathcal{L}(A)\downarrow = \{v \mid \exists w \in \mathcal{L}(A) \colon v \preceq w\}$ (Lossiness)
Upward closure: $\mathcal{L}(A)\uparrow = \{v \mid \exists w \in \mathcal{L}(A) \colon w \preceq v\}$ (Gaininess)

Theorem ([Haines 1969],[Abdulla et al. 2004])

$\mathcal{L}(A)\downarrow, \mathcal{L}(A)\uparrow$ *always simply regular.*

Regular languages can be represented by finite automata

But: Closures are not necessarily effectively regular

Closures

Computing the downward closure

Given: Automaton A.

Compute: Finite automaton B with $\mathcal{L}(B) = \mathcal{L}(A)\!\downarrow$

Computing the upward closure

Given: Automaton A.

Compute: Finite automaton B with $\mathcal{L}(B) = \mathcal{L}(A)\!\uparrow$

Closures

Computing the downward closure

Given: Automaton A.

Compute: Finite automaton B with $\mathcal{L}(B) = \mathcal{L}(A)\downarrow$

Computing the upward closure

Given: Automaton A.

Compute: Finite automaton B with $\mathcal{L}(B) = \mathcal{L}(A)\uparrow$

Computing closure is taking the environment into account

Computing the downward closure

Given: Automaton A.

Compute: Finite automaton B with $\mathcal{L}(B) = \mathcal{L}(A)\downarrow$

Computing the upward closure

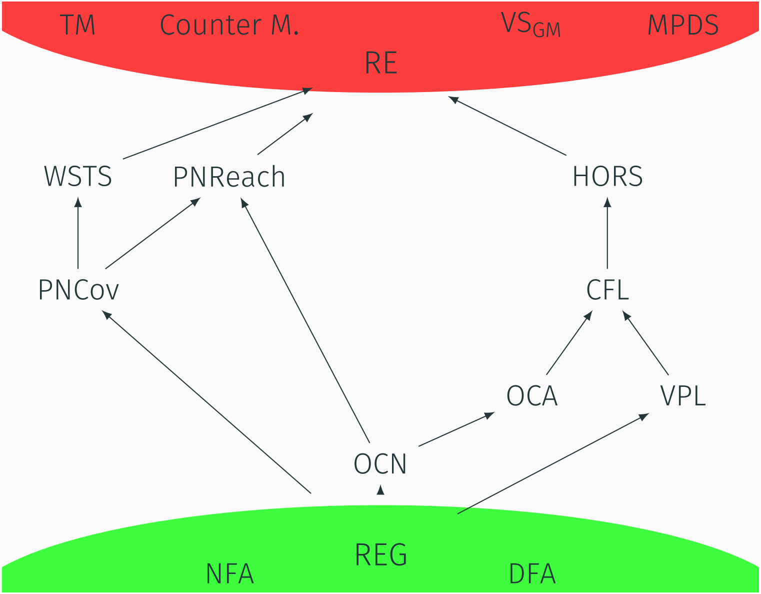
Given: Automaton A.

Compute: Finite automaton B with $\mathcal{L}(B) = \mathcal{L}(A)\uparrow$

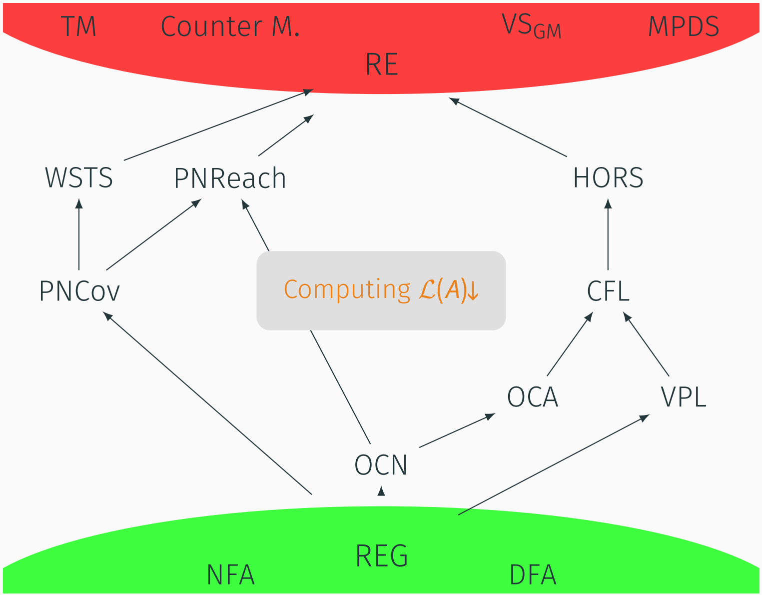
Computing closure is taking the environment into account

Finite automaton can serve as certificate

Closures

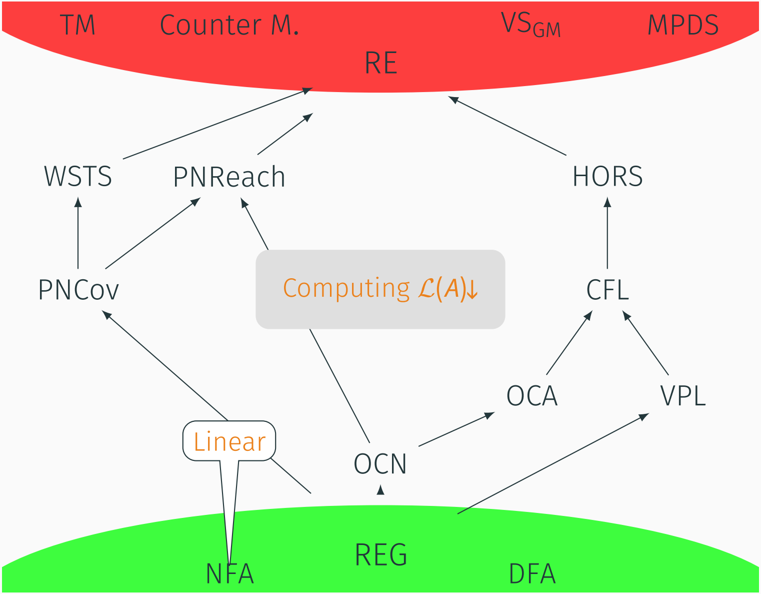

TM
Counter M.
VS_{GM}
MPDS
RE
WSTS
PNReach
HORS
PNCov
CFL
OCA
VPL
OCN
REG
NFA
DFA

Closures

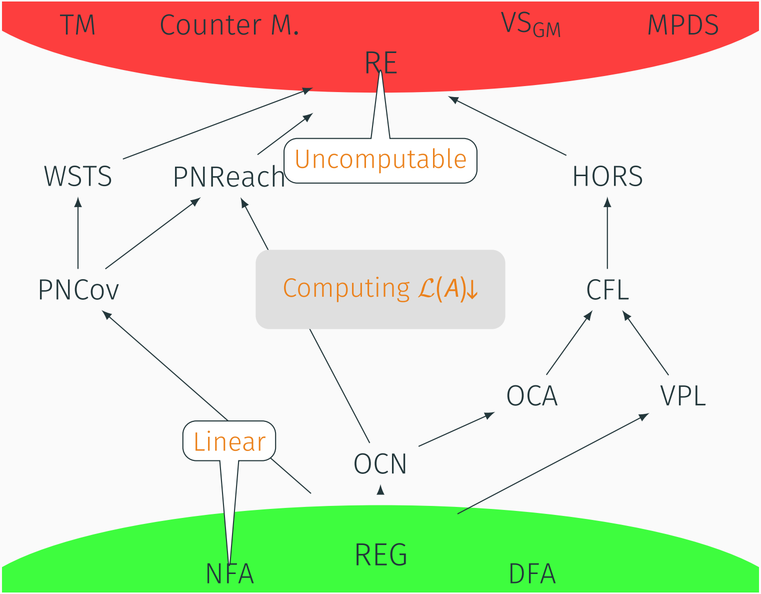


TM
Counter M.
VS_{GM}
MPDS
RE
WSTS
PNReach
HORS
PNCov
Computing $\mathcal{L}(A)\downarrow$
CFL
OCA
VPL
OCN
REG
NFA
DFA

Closures

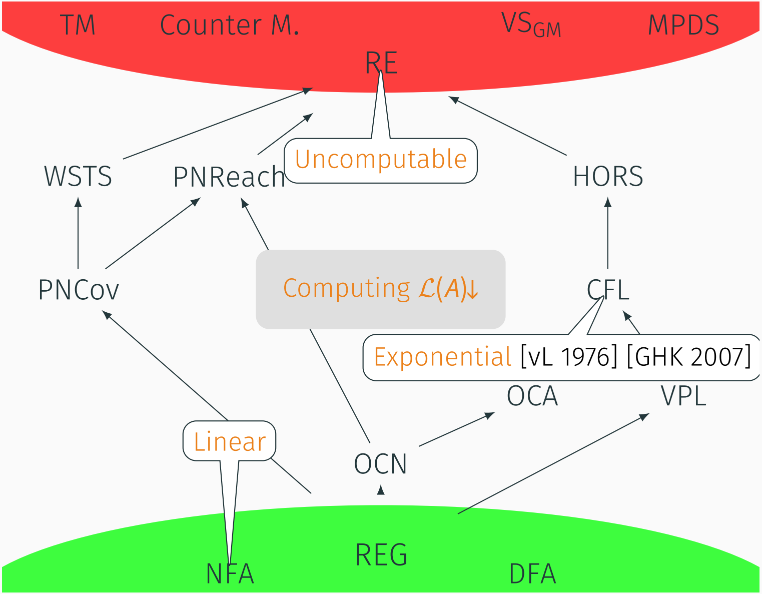

TM
Counter M.
VS_{GM}
MPDS
RE
WSTS
PNReach
HORS
PNCov
Computing $\mathcal{L}(A)\downarrow$
CFL
OCA
VPL
Linear
OCN
REG
NFA
DFA

Closures



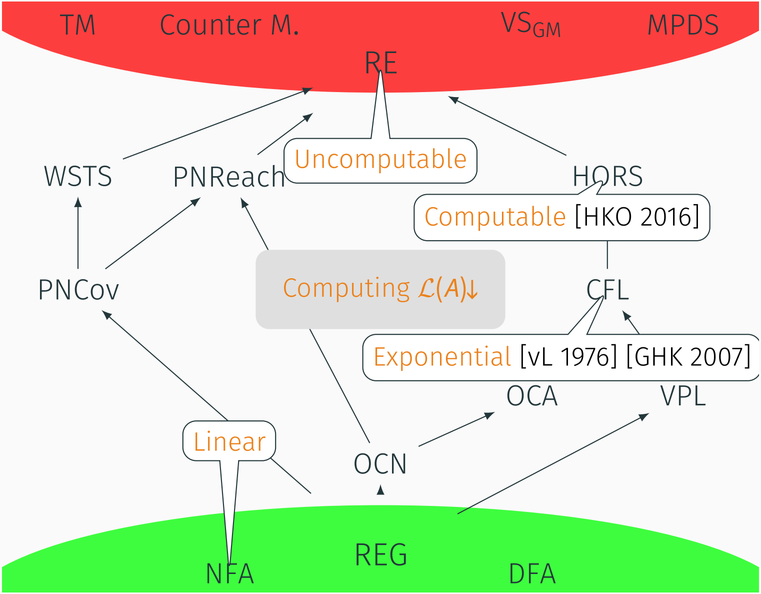
TM
Counter M.
VS_{GM}
MPDS
RE
Uncomputable
WSTS
PNReach
HORS
PNCov
Computing $\mathcal{L}(A)\downarrow$
CFL
OCA
VPL
Linear
OCN
REG
NFA
DFA

Closures



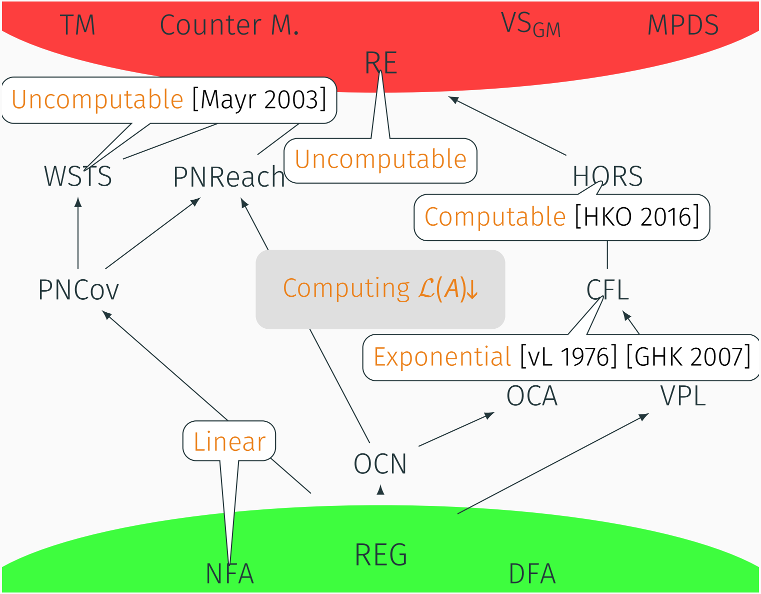
TM
Counter M.
VS_{GM}
MPDS
RE
Uncomputable
WSTS
PNReach
HORS
PNCov
Computing $\mathcal{L}(A)\downarrow$
CFL
Exponential [vL 1976] [GHK 2007]
OCA
VPL
Linear
OCN
REG
NFA
DFA

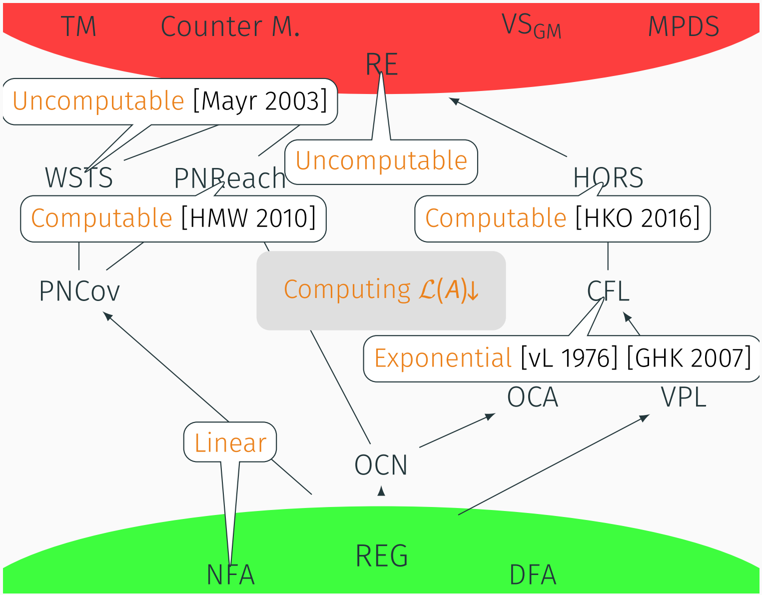
Closures



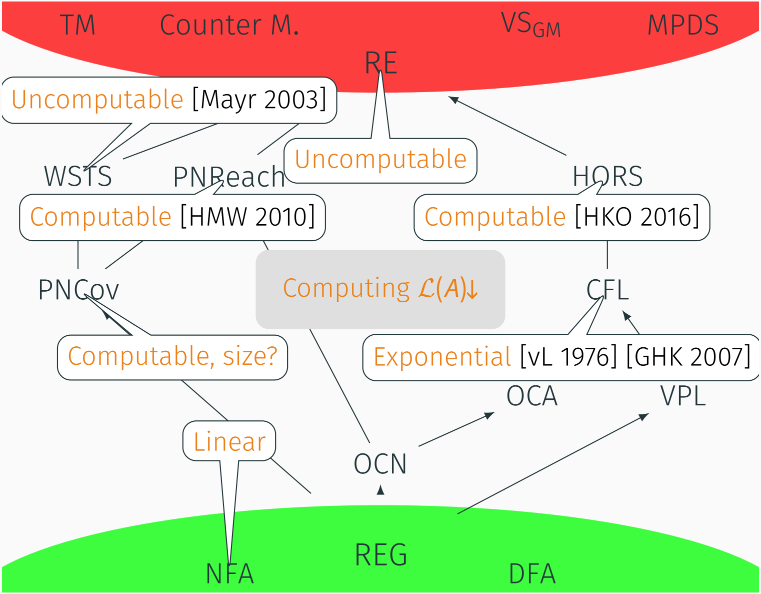
TM
Counter M.
VS$_{GM}$
MPDS
RE
Uncomputable
WSTS
PNReach
HORS
Computable [HKO 2016]
PNCov
Computing $\mathcal{L}(A)\downarrow$
CFL
Exponential [vL 1976] [GHK 2007]
OCA
VPL
Linear
OCN
REG
NFA
DFA

Closures


TM
Counter M.
VS$_{GM}$
MPDS
RE
Uncomputable [Mayr 2003]
Uncomputable
WSTS
PNReach
HORS
Computable [HKO 2016]
PNCov
Computing $\mathcal{L}(A)\downarrow$
CFL
Exponential [vL 1976] [GHK 2007]
OCA
VPL
Linear
OCN
REG
NFA
DFA


TM
Counter M.
VS_{GM}
MPDS
RE
Uncomputable [Mayr 2003]
Uncomputable
WSTS
PNReach
HORS
Computable [HMW 2010]
Computable [HKO 2016]
PNCov
Computing $\mathcal{L}(A)\downarrow$
CFL
Exponential [vL 1976] [GHK 2007]
OCA
VPL
Linear
OCN
REG
NFA
DFA

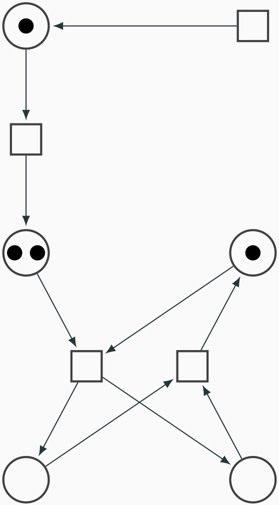
Closures



TM
Counter M.
VS_{GM}
MPDS
RE
Uncomputable [Mayr 2003]
Uncomputable
WSTS
PNReach
HORS
Computable [HMW 2010]
Computable [HKO 2016]
PNCov
Computing $\mathcal{L}(A)\downarrow$
CFL
Computable, size?
Exponential [vL 1976] [GHK 2007]
OCA
VPL
Linear
OCN
REG
NFA
DFA

Closures

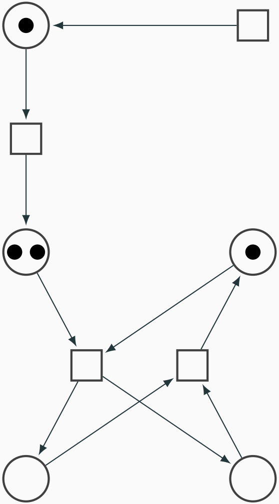
Petri nets



Closures

Petri nets

- a finite automaton run by multiple threads
- number of threads is unbounded
- threads can spawn, die, synchronize at runtime

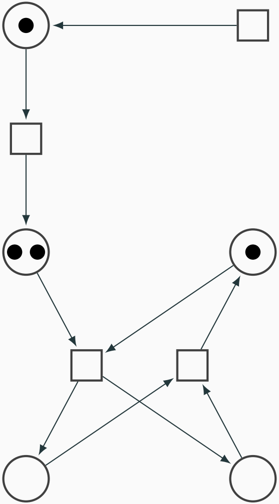


Closures

Petri nets

- a finite automaton run by multiple threads
- number of threads is unbounded
- threads can spawn, die, synchronize at runtime

Limitation: Cannot check non-existence of threads



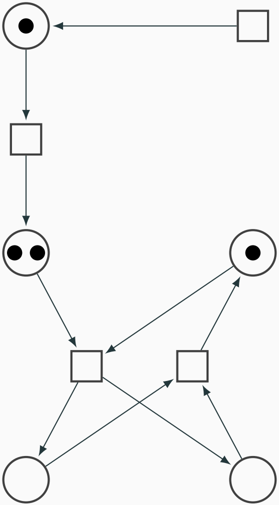
Closures

Petri nets

- a finite automaton run by multiple threads
- number of threads is unbounded
- threads can spawn, die, synchronize at runtime

Limitation: Cannot check non-existence of threads

Good for modelling concurrent systems



Closures

	Petri nets
Compute $\mathcal{L}(A)\downarrow$	non-prim. rec. [HMW 2010]
Compute $\mathcal{L}(A)\uparrow$	???

Closures

	Petri nets
Compute $\mathcal{L}(A)\downarrow$	non-prim. rec. [HMW 2010]
Compute $\mathcal{L}(A)\uparrow$	doubly exponential

Closures

	Petri nets	BPP nets
Compute $\mathcal{L}(A)\downarrow$	non-prim. rec. [HMW 2010]	exponential
Compute $\mathcal{L}(A)\uparrow$	doubly exponential	exponential

Closures

	Petri nets	BPP nets
Compute $\mathcal{L}(A)\downarrow$	non-prim. rec. [HMW 2010]	exponential
Compute $\mathcal{L}(A)\uparrow$	doubly exponential	exponential
SRE $\subseteq \mathcal{L}(A)\downarrow$	EXPSPACE-compl.	NP-compl.
SRE $\subseteq \mathcal{L}(A)\uparrow$	EXPSPACE-compl.	NP-compl.

Closures

	Petri nets	BPP nets
Compute $\mathcal{L}(A)\downarrow$	non-prim. rec. [HMW 2010]	exponential
Compute $\mathcal{L}(A)\uparrow$	doubly exponential	exponential
SRE $\subseteq \mathcal{L}(A)\downarrow$	EXPSPACE-compl.	NP-compl.
SRE $\subseteq \mathcal{L}(A)\uparrow$	EXPSPACE-compl.	NP-compl.

Theorem

It is decidable whether $\mathcal{L}$(finite automaton) $\subseteq \mathcal{L}$(Petri net).

Closures

	Petri nets	BPP nets
Compute $\mathcal{L}(A)\!\downarrow$	non-prim. rec. [HMW 2010]	exponential
Compute $\mathcal{L}(A)\!\uparrow$	doubly exponential	exponential
SRE $\subseteq \mathcal{L}(A)\!\downarrow$	EXPSPACE-compl.	NP-compl.
SRE $\subseteq \mathcal{L}(A)\!\uparrow$	EXPSPACE-compl.	NP-compl.

Theorem

It is decidable whether $\mathcal{L}$(finite automaton) $\subseteq \mathcal{L}$(Petri net).

Theorem

It is decidable whether $\mathcal{L}(PN) = \mathcal{L}(PN)\!\downarrow$ resp. $\mathcal{L}(PN) = \mathcal{L}(PN)\!\uparrow$.

Closures

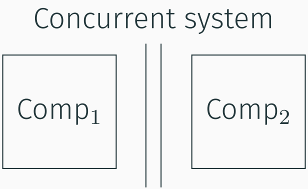
Part III. of the thesis

Publication:

M. F. Atig, R. Meyer, *S. M.*, and P. Saivasan
On the upward/downward closure of Petri nets
In: MFCS 2017, volume 83 of LIPIcs, pages 49:1–49:14

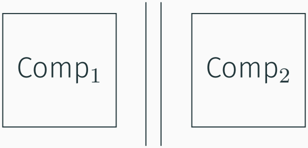
2^{nd} example: Compositional verification & Regular separability

Compositional verification



Compositional verification

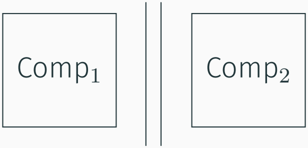
Concurrent system



State explosion problem

Compositional verification

Concurrent system



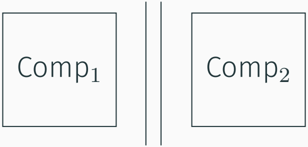
State explosion problem

$$\#\text{LoC}(\text{Comp}_1 \,||\, \text{Comp}_2) = \#\text{LoC}(\text{Comp}_1) + \#\text{LoC}(\text{Comp}_2)$$

$$\#\text{States}(\text{Comp}_1 \,||\, \text{Comp}_2) = \#\text{States}(\text{Comp}_1) * \#\text{States}(\text{Comp}_2)$$

Compositional verification

Concurrent system



State explosion problem

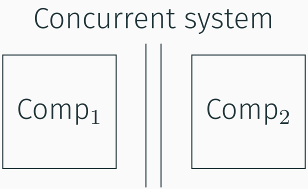
$$\#\text{LoC}(\text{Comp}_1 \,||\, \text{Comp}_2) = \#\text{LoC}(\text{Comp}_1) + \#\text{LoC}(\text{Comp}_2)$$

$$\#\text{States}(\text{Comp}_1 \,||\, \text{Comp}_2) = \#\text{States}(\text{Comp}_1) * \#\text{States}(\text{Comp}_2)$$

Solution: Compositional verification

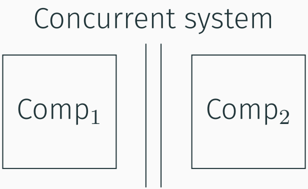
verify each component separately

Compositional verification



Assume-guarantee reasoning [Jones 1983]

Compositional verification

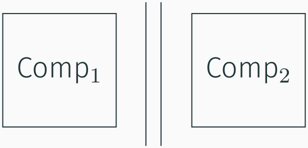


Assume-guarantee reasoning [Jones 1983]

⟨Assume⟩ Comp ⟨Guarantee⟩

Compositional verification

Concurrent system



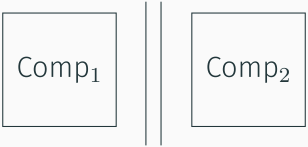
Assume-guarantee reasoning [Jones 1983]

$$\langle \text{Assume} \rangle \ \text{Comp} \ \langle \text{Guarantee} \rangle \text{ satisfied if}$$

$$\forall \text{Env}\colon \text{Comp} \,||\, \text{Env} \models \text{Assume} \implies \text{Comp} \,||\, \text{Env} \models \text{Guarantee}$$

Compositional verification

Concurrent system



Assume-guarantee reasoning [Jones 1983]

⟨Assume⟩ Comp ⟨Guarantee⟩ satisfied if

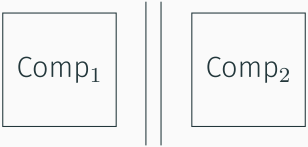
$$\forall \text{Env}\colon \text{Comp} \,||\, \text{Env} \models \text{Assume} \implies \text{Comp} \,||\, \text{Env} \models \text{Guarantee}$$

Asymmetric proof rule for two components:

$$\frac{\langle \text{true} \rangle\ \text{Comp}_1\ \langle \text{Assume} \rangle \qquad \langle \text{Assume} \rangle\ \text{Comp}_2\ \langle \text{Guarantee} \rangle}{\langle \text{true} \rangle\ \text{Comp}_1 \,||\, \text{Comp}_2\ \langle \text{Guarantee} \rangle}$$

Compositional verification

Concurrent system



Assume-guarantee reasoning [Jones 1983]

$$\langle \text{Assume} \rangle\ \text{Comp}\ \langle \text{Guarantee} \rangle \text{ satisfied if}$$

$$\forall \text{Env}\colon \text{Comp} \,||\, \text{Env} \vDash \text{Assume} \implies \text{Comp} \,||\, \text{Env} \vDash \text{Guarantee}$$

Asymmetric proof rule for two components:

$$\frac{\langle \text{true} \rangle\ \text{Comp}_1\ \langle \text{Assume} \rangle \qquad \langle \text{Assume} \rangle\ \text{Comp}_2\ \langle \text{Guarantee} \rangle}{\langle \text{true} \rangle\ \text{Comp}_1 \,||\, \text{Comp}_2\ \langle \text{Guarantee} \rangle}$$

When checking Comp$_2$:

Environment: Comp$_1$

Certificate: Assume

Compositional verification

$$\langle \text{true} \rangle \ \text{Comp}_1 \,||\, \text{Comp}_2 \ \langle \text{Guarantee} \rangle$$

How to express this using languages?

Compositional verification

$$\langle \text{true} \rangle\ \text{Comp}_1 \,||\, \text{Comp}_2\ \langle \text{Guarantee} \rangle$$

How to express this using languages?

First simplification:

$$\langle \text{Assume} \rangle\ \text{Comp}\ \langle \text{Guarantee} \rangle \text{ satisfied if}$$

$$\forall \text{Env}\colon \text{Comp} \,||\, \text{Env} \models \text{Assume} \implies \text{Comp} \,||\, \text{Env} \models \text{Guarantee}$$

Compositional verification

$$\langle \text{true} \rangle\ \text{Comp}_1 \,||\, \text{Comp}_2\ \langle \text{Guarantee} \rangle$$

How to express this using languages?

First simplification:

$$\langle \text{Assume} \rangle\ \text{Comp}\ \langle \text{Guarantee} \rangle \text{ satisfied if}$$

$$\text{Comp} \models \text{Assume} \implies \text{Comp} \models \text{Guarantee}$$

In particular:

$\langle \text{true} \rangle\ \text{Comp}\ \langle \text{Guarantee} \rangle \quad \text{if} \quad \text{Comp} \models \text{Guarantee}.$

Compositional verification

$$\mathrm{Comp}_1 \,||\, \mathrm{Comp}_2 \models \mathrm{Guarantee}$$

How to express this using languages?

Compositional verification

$$\text{Comp}_1 \,||\, \text{Comp}_2 \models \text{Guarantee}$$

How to express this using languages?

Second simplification:

$\text{Comp} \models \text{Spec} \iff \mathcal{L}(\text{Comp}) \subseteq \mathcal{L}(\text{Spec}) \iff \mathcal{L}(\text{Comp}) \cap \mathcal{L}(\overline{\text{Spec}}) = \emptyset$

Compositional verification

$$\mathcal{L}(\mathrm{Comp}_1 \,||\, \mathrm{Comp}_2) \cap \mathcal{L}\left(\overline{\mathrm{Guarantee}}\right) = \varnothing$$

How to express this using languages?

Second simplification:

$\mathrm{Comp} \vDash \mathrm{Spec} \iff \mathcal{L}(\mathrm{Comp}) \subseteq \mathcal{L}(\mathrm{Spec}) \iff \mathcal{L}(\mathrm{Comp}) \cap \mathcal{L}(\overline{\mathrm{Spec}}) = \varnothing$

Compositional verification

$$\mathcal{L}(\text{Comp}_1 \,||\, \text{Comp}_2) \cap \mathcal{L}\left(\overline{\text{Guarantee}}\right) = \varnothing$$

How to express this using languages?

Second simplification:
$\text{Comp} \models \text{Spec} \iff \mathcal{L}(\text{Comp}) \subseteq \mathcal{L}(\text{Spec}) \iff \mathcal{L}(\text{Comp}) \cap \mathcal{L}(\overline{\text{Spec}}) = \varnothing$

Third simplification:
$\mathcal{L}(\text{Comp}_1 \,||\, \text{Comp}_2) = \mathcal{L}(\text{Comp}_1) \,||\, \mathcal{L}(\text{Comp}_2) = \mathcal{L}(\text{Comp}'_1) \cap \mathcal{L}(\text{Comp}'_2)$

Compositional verification

$$\mathcal{L}(\mathrm{Comp}_1') \cap \mathcal{L}(\mathrm{Comp}_2') \cap \mathcal{L}(\overline{\mathrm{Guarantee}}) = \varnothing$$

How to express this using languages?

Second simplification:
$\mathrm{Comp} \models \mathrm{Spec} \iff \mathcal{L}(\mathrm{Comp}) \subseteq \mathcal{L}(\mathrm{Spec}) \iff \mathcal{L}(\mathrm{Comp}) \cap \mathcal{L}(\overline{\mathrm{Spec}}) = \varnothing$

Third simplification:
$\mathcal{L}(\mathrm{Comp}_1 \,||\, \mathrm{Comp}_2) = \mathcal{L}(\mathrm{Comp}_1) \,||\, \mathcal{L}(\mathrm{Comp}_2) = \mathcal{L}(\mathrm{Comp}_1') \cap \mathcal{L}(\mathrm{Comp}_2')$

Compositional verification

$$\mathcal{L}(\text{Comp}_1') \cap \mathcal{L}(\text{Comp}_2') \cap \mathcal{L}\left(\overline{\text{Guarantee}}\right) = \varnothing$$

How to express this using languages?

Second simplification:

$\text{Comp} \models \text{Spec} \iff \mathcal{L}(\text{Comp}) \subseteq \mathcal{L}(\text{Spec}) \iff \mathcal{L}(\text{Comp}) \cap \mathcal{L}(\overline{\text{Spec}}) = \varnothing$

Third simplification:

$\mathcal{L}(\text{Comp}_1 \,||\, \text{Comp}_2) = \mathcal{L}(\text{Comp}_1) \,||\, \mathcal{L}(\text{Comp}_2) = \mathcal{L}(\text{Comp}_1') \cap \mathcal{L}(\text{Comp}_2')$

Final simplification:

$\mathcal{L}(\text{Comp}_2'') := \text{Comp}_2' \cap \mathcal{L}(\overline{\text{Guarantee}})$

Compositional verification

$$\mathcal{L}(\mathrm{Comp}_1') \cap \mathcal{L}(\mathrm{Comp}_2'') = \varnothing$$

How to express this using languages?

Second simplification:

$\mathrm{Comp} \models \mathrm{Spec} \iff \mathcal{L}(\mathrm{Comp}) \subseteq \mathcal{L}(\mathrm{Spec}) \iff \mathcal{L}(\mathrm{Comp}) \cap \mathcal{L}(\overline{\mathrm{Spec}}) = \varnothing$

Third simplification:

$\mathcal{L}(\mathrm{Comp}_1 \,||\, \mathrm{Comp}_2) = \mathcal{L}(\mathrm{Comp}_1) \,||\, \mathcal{L}(\mathrm{Comp}_2) = \mathcal{L}(\mathrm{Comp}_1') \cap \mathcal{L}(\mathrm{Comp}_2')$

Final simplification:

$\mathcal{L}(\mathrm{Comp}_2'') := \mathrm{Comp}_2' \cap \mathcal{L}(\overline{\mathrm{Guarantee}})$

Compositional verification

$\langle \text{true} \rangle \ \text{Comp}_1 \,||\, \text{Comp}_2 \ \langle \text{Guarantee} \rangle$ $\qquad$ $\mathcal{L}(\text{Comp}_1') \cap \mathcal{L}(\text{Comp}_2'') = \varnothing$

Compositional verification

$\langle \text{true} \rangle \text{ Comp}_1 \,||\, \text{Comp}_2 \ \langle \text{Guarantee} \rangle \qquad \mathcal{L} \cap \mathcal{K} = \emptyset$

Compositional verification

$\langle \text{true} \rangle\ \text{Comp}_1 \,||\, \text{Comp}_2\ \langle \text{Guarantee} \rangle$ $\mathcal{L} \cap \mathcal{K} = \emptyset$

Proof rule:

$$\frac{\begin{array}{l}\langle \text{true} \rangle\ \text{Comp}_1\ \langle \text{Assume} \rangle \\ \langle \text{Assume} \rangle\ \text{Comp}_2\ \langle \text{Guarantee} \rangle\end{array}}{\langle \text{true} \rangle\ \text{Comp}_1 \,||\, \text{Comp}_2\ \langle \text{Guarantee} \rangle}$$

Compositional verification

$\langle \text{true} \rangle\ \text{Comp}_1 \,||\, \text{Comp}_2\ \langle \text{Guarantee} \rangle$

$\mathcal{L} \cap \mathcal{K} = \varnothing$

Proof rule:

$$\frac{\langle \text{true} \rangle\ \text{Comp}_1\ \langle \text{Assume} \rangle \qquad \langle \text{Assume} \rangle\ \text{Comp}_2\ \langle \text{Guarantee} \rangle}{\langle \text{true} \rangle\ \text{Comp}_1 \,||\, \text{Comp}_2\ \langle \text{Guarantee} \rangle}$$

Separability proof rule:

$$\frac{\mathcal{L} \subseteq \mathcal{R} \qquad \mathcal{K} \subseteq \overline{\mathcal{R}}}{\mathcal{L} \cap \mathcal{K} = \varnothing}$$

Compositional verification

$\langle \text{true} \rangle\ \text{Comp}_1 \,||\, \text{Comp}_2\ \langle \text{Guarantee} \rangle$

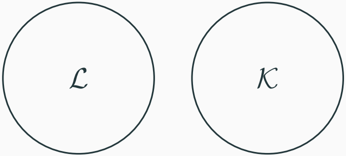
Proof rule:

$$\frac{\begin{array}{l}\langle \text{true} \rangle\ \text{Comp}_1\ \langle \text{Assume} \rangle \\ \langle \text{Assume} \rangle\ \text{Comp}_2\ \langle \text{Guarantee} \rangle\end{array}}{\langle \text{true} \rangle\ \text{Comp}_1 \,||\, \text{Comp}_2\ \langle \text{Guarantee} \rangle}$$

$\mathcal{L} \cap \mathcal{K} = \varnothing$

Separability proof rule:

$$\frac{\begin{array}{l}\mathcal{L} \subseteq \mathcal{R} \\ \mathcal{K} \subseteq \overline{\mathcal{R}}\end{array}}{\mathcal{L} \cap \mathcal{K} = \varnothing}$$



Compositional verification

$\langle \text{true} \rangle\ \text{Comp}_1 \,||\, \text{Comp}_2\ \langle \text{Guarantee} \rangle$

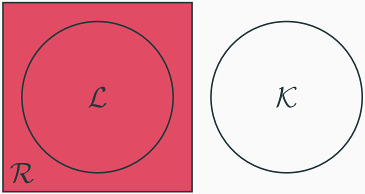
$\mathcal{L} \cap \mathcal{K} = \emptyset$

Proof rule:

$$\frac{\langle \text{true} \rangle\ \text{Comp}_1\ \langle \text{Assume} \rangle \qquad \langle \text{Assume} \rangle\ \text{Comp}_2\ \langle \text{Guarantee} \rangle}{\langle \text{true} \rangle\ \text{Comp}_1 \,||\, \text{Comp}_2\ \langle \text{Guarantee} \rangle}$$

Separability proof rule:

$$\frac{\mathcal{L} \subseteq \mathcal{R} \qquad \mathcal{K} \subseteq \overline{\mathcal{R}}}{\mathcal{L} \cap \mathcal{K} = \emptyset}$$



Compositional verification

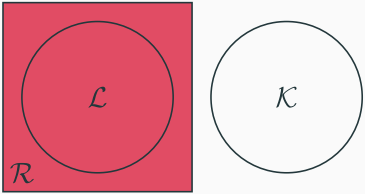
⟨true⟩ $\text{Comp}_1 \,||\, \text{Comp}_2$ ⟨Guarantee⟩ $\mathcal{L} \cap \mathcal{K} = \varnothing$

Proof rule:

$$\frac{\langle\text{true}\rangle\ \text{Comp}_1\ \langle\text{Assume}\rangle \qquad \langle\text{Assume}\rangle\ \text{Comp}_2\ \langle\text{Guarantee}\rangle}{\langle\text{true}\rangle\ \text{Comp}_1 \,||\, \text{Comp}_2\ \langle\text{Guarantee}\rangle}$$

Separability proof rule:

$$\frac{\mathcal{L} \subseteq \mathcal{R} \qquad \mathcal{K} \subseteq \overline{\mathcal{R}}}{\mathcal{L} \cap \mathcal{K} = \varnothing}$$



Comp. verification $\triangleq$ finding certificate for intersection emptiness

Regular separability

Separability

Given: Languages $\mathcal{L}, \mathcal{K}$.

Question: Is there $\mathcal{R}$ with $\mathcal{L} \subseteq \mathcal{R}$ and $\mathcal{K} \cap \mathcal{R} = \emptyset$?

Regular separability

Separability

Given: Languages $\mathcal{L}, \mathcal{K}$.

Question: Is there $\mathcal{R}$ with $\mathcal{L} \subseteq \mathcal{R}$ and $\mathcal{K} \cap \mathcal{R} = \emptyset$?

$\mathcal{R}$ is an abstraction of $\mathcal{L}$ that is a certificate for $\mathcal{L} \cap \mathcal{K} = \emptyset$.

Regular separability

Separability

Given: Languages $\mathcal{L}, \mathcal{K}$.

Question: Is there $\mathcal{R}$ with $\mathcal{L} \subseteq \mathcal{R}$ and $\mathcal{K} \cap \mathcal{R} = \emptyset$?

$\mathcal{R}$ is an abstraction of $\mathcal{L}$ that is a certificate for $\mathcal{L} \cap \mathcal{K} = \emptyset$.

Only makes sense if $\mathcal{R}$ is from a simpler class!

Regular separability

Regular separability for class $\mathcal{F}$

Given: Languages $\mathcal{L}, \mathcal{K}$ from class $\mathcal{F}$.

Question: Is there $\mathcal{R}$ regular with $\mathcal{L} \subseteq \mathcal{R}$ and $\mathcal{K} \cap \mathcal{R} = \emptyset$?

$\mathcal{R}$ is an abstraction of $\mathcal{L}$ that is a certificate for $\mathcal{L} \cap \mathcal{K} = \emptyset$.

Only makes sense if $\mathcal{R}$ is from a simpler class!

Regular separability

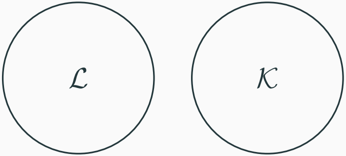
Regular separability for class $\mathcal{F}$

Given: Languages $\mathcal{L}, \mathcal{K}$ from class $\mathcal{F}$.

Question: Is there $\mathcal{R}$ regular with $\mathcal{L} \subseteq \mathcal{R}$ and $\mathcal{K} \cap \mathcal{R} = \emptyset$?

$\mathcal{R}$ is an abstraction of $\mathcal{L}$ that is a certificate for $\mathcal{L} \cap \mathcal{K} = \emptyset$.

Only makes sense if $\mathcal{R}$ is from a simpler class!



Regular separability

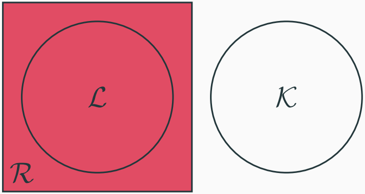
Regular separability for class $\mathcal{F}$

Given: Languages $\mathcal{L}, \mathcal{K}$ from class $\mathcal{F}$.

Question: Is there $\mathcal{R}$ regular with $\mathcal{L} \subseteq \mathcal{R}$ and $\mathcal{K} \cap \mathcal{R} = \emptyset$?

$\mathcal{R}$ is an abstraction of $\mathcal{L}$ that is a certificate for $\mathcal{L} \cap \mathcal{K} = \emptyset$.

Only makes sense if $\mathcal{R}$ is from a simpler class!



Regular separability

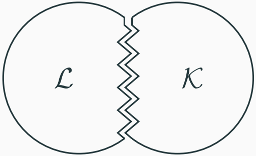
Regular separability for class $\mathcal{F}$

Given: Languages $\mathcal{L}, \mathcal{K}$ from class $\mathcal{F}$.

Question: Is there $\mathcal{R}$ regular with $\mathcal{L} \subseteq \mathcal{R}$ and $\mathcal{K} \cap \mathcal{R} = \emptyset$?

$\mathcal{R}$ is an abstraction of $\mathcal{L}$ that is a certificate for $\mathcal{L} \cap \mathcal{K} = \emptyset$.

Only makes sense if $\mathcal{R}$ is from a simpler class!



No separator exists!

Regular separability

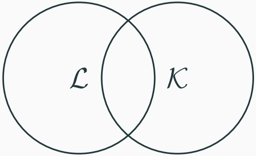
Regular separability for class $\mathcal{F}$

Given: Languages $\mathcal{L}, \mathcal{K}$ from class $\mathcal{F}$.

Question: Is there $\mathcal{R}$ regular with $\mathcal{L} \subseteq \mathcal{R}$ and $\mathcal{K} \cap \mathcal{R} = \emptyset$?

$\mathcal{R}$ is an abstraction of $\mathcal{L}$ that is a certificate for $\mathcal{L} \cap \mathcal{K} = \emptyset$.

Only makes sense if $\mathcal{R}$ is from a simpler class!



No separator exists!

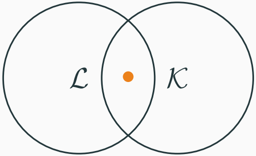
Regular separability for class $\mathcal{F}$

Given: Languages $\mathcal{L}, \mathcal{K}$ from class $\mathcal{F}$.

Question: Is there $\mathcal{R}$ regular with $\mathcal{L} \subseteq \mathcal{R}$ and $\mathcal{K} \cap \mathcal{R} = \emptyset$?

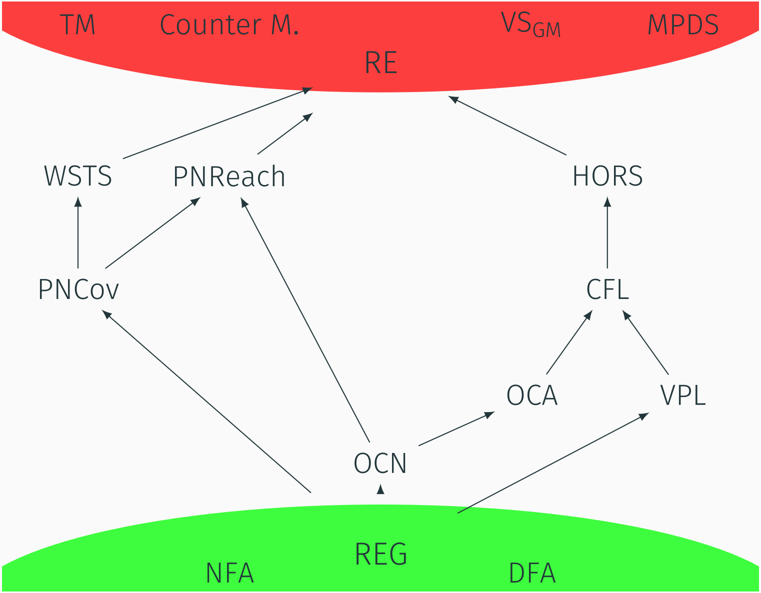
$\mathcal{R}$ is an abstraction of $\mathcal{L}$ that is a certificate for $\mathcal{L} \cap \mathcal{K} = \emptyset$.

Only makes sense if $\mathcal{R}$ is from a simpler class!

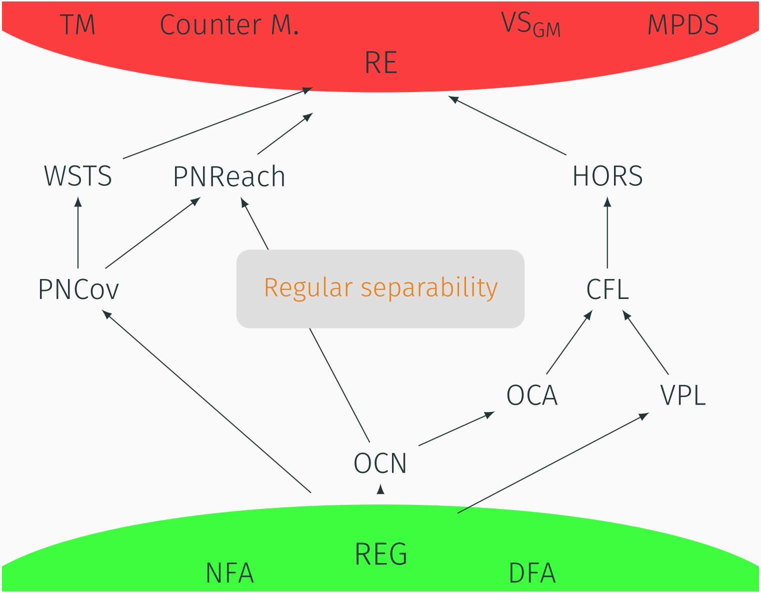


No separator exists!

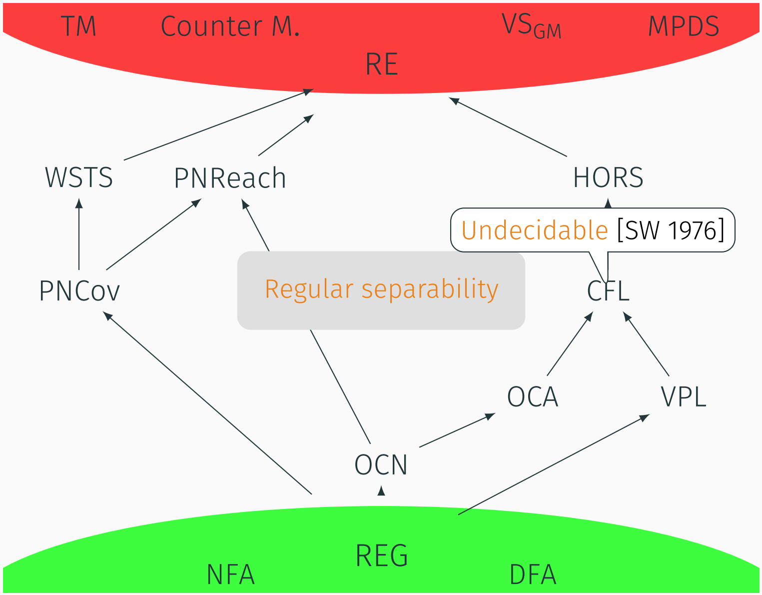
Regular separability



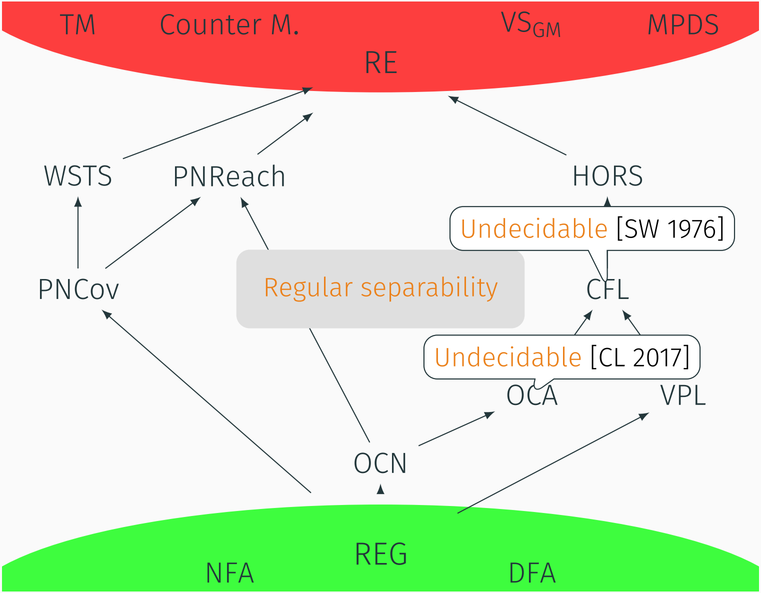
Regular separability



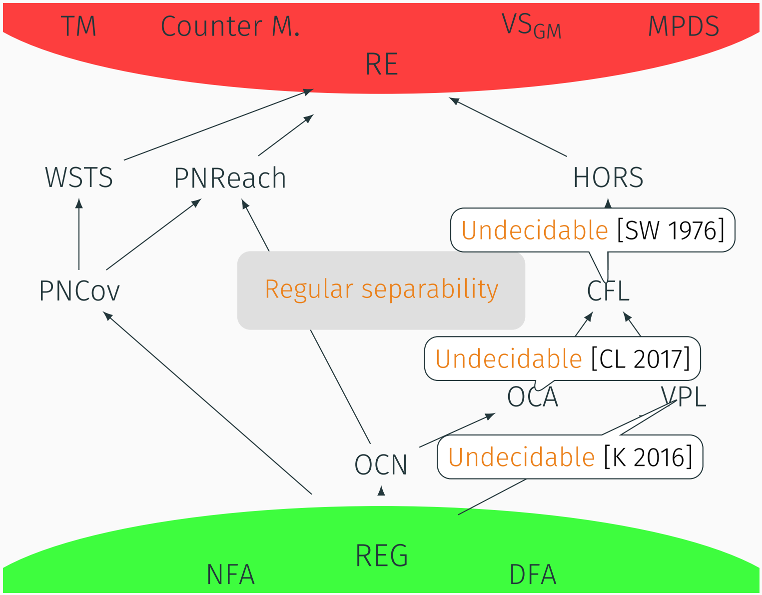
Regular separability



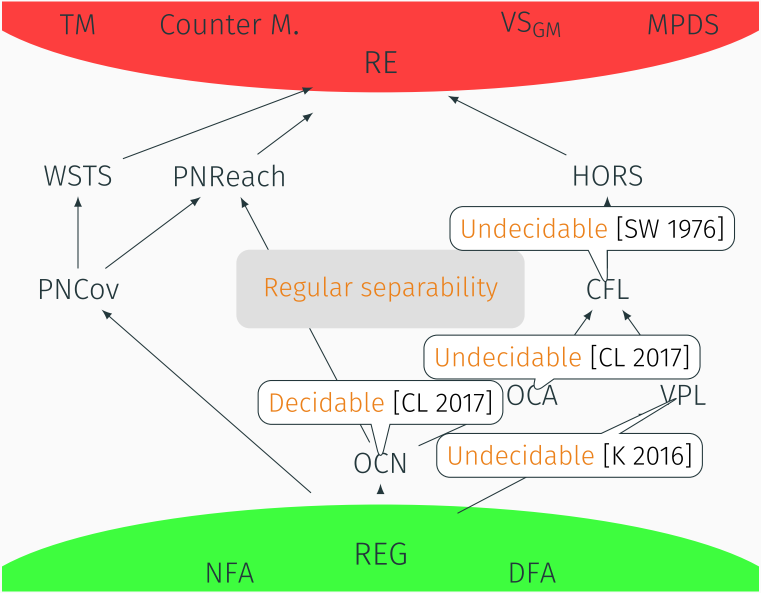
Regular separability



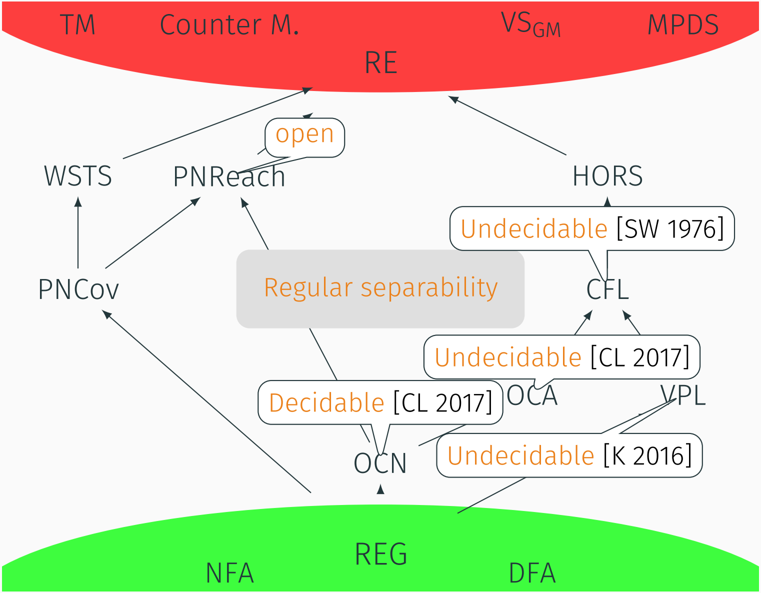
Regular separability



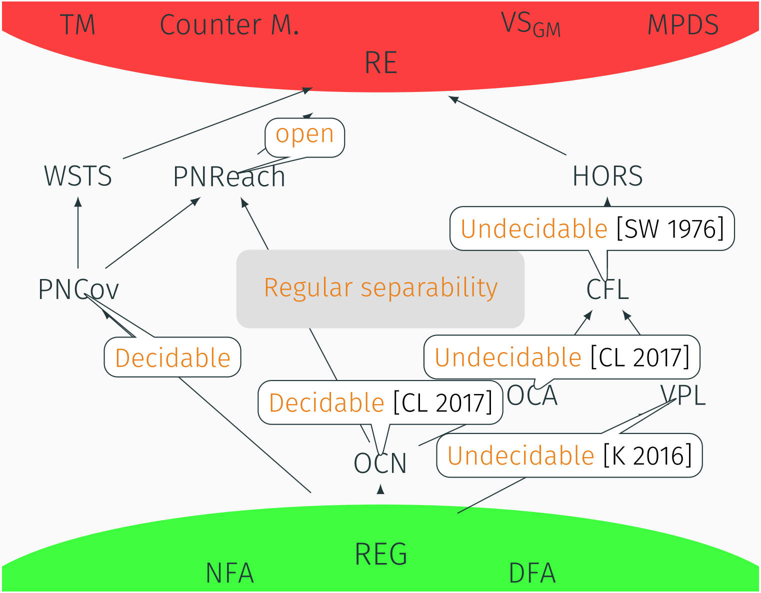
Regular separability



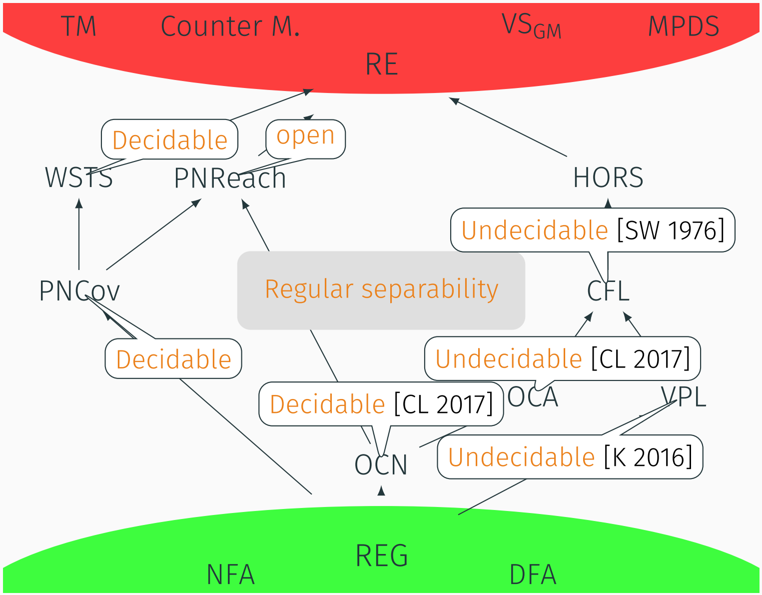
Regular separability



Regular separability



Regular separability



Regular separability

WSTS = class of languages of finitely branching well-structured transition systems

Regular separability

WSTS = class of languages of finitely branching well-structured transition systems e.g. Petri nets with coverability

Regular separability

WSTS = class of languages of finitely branching well-structured transition systems e.g. Petri nets with coverability

Theorem

If two WSTS languages are disjoint, then they are regularly separable.

Regular separability

WSTS = class of languages of finitely branching well-structured transition systems e.g. Petri nets with coverability

Theorem

If two WSTS languages are disjoint, then they are regularly separable.

Consequences I:

Regular separability

WSTS = class of languages of finitely branching well-structured transition systems e.g. Petri nets with coverability

Theorem

If two WSTS languages are disjoint, then they are regularly separable.

Consequences I:

- Separability is decidable under mild assumptions

Regular separability

WSTS = class of languages of finitely branching well-structured transition systems e.g. Petri nets with coverability

Theorem

If two WSTS languages are disjoint, then they are regularly separable.

Consequences I:

- Separability is decidable under mild assumptions
  (Just check whether the languages are disjoint)

Regular separability

WSTS = class of languages of finitely branching well-structured transition systems e.g. Petri nets with coverability

Theorem

If two WSTS languages are disjoint, then they are regularly separable.

Consequences I:

- Separability is decidable under mild assumptions (Just check whether the languages are disjoint)
- Separator can be constructed under mild assumptions

Regular separability

WSTS = class of languages of finitely branching well-structured transition systems e.g. Petri nets with coverability

Theorem

If two WSTS languages are disjoint, then they are regularly separable.

Consequences II:

Regular separability

WSTS = class of languages of finitely branching well-structured transition systems e.g. Petri nets with coverability

Theorem

If two WSTS languages are disjoint, then they are regularly separable.

Consequences II:

Corollary

If a language and its complement are WSTS languages, they are necessarily regular.

Regular separability

WSTS = class of languages of finitely branching well-structured transition systems e.g. Petri nets with coverability

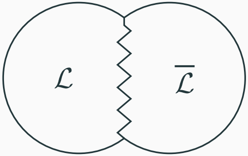
Theorem

If two WSTS languages are disjoint, then they are regularly separable.

Consequences II:

Corollary

If a language and its complement are WSTS languages, they are necessarily regular.



Regular separability

WSTS = class of languages of finitely branching well-structured transition systems e.g. Petri nets with coverability

Theorem

If two WSTS languages are disjoint, then they are regularly separable.

Consequences II:

Corollary

If a language and its complement are WSTS languages, they are necessarily regular.

Regular separability

WSTS = class of languages of finitely branching well-structured transition systems e.g. Petri nets with coverability

Theorem

If two WSTS languages are disjoint, then they are regularly separable.

Proof:

Given $\mathcal{L}(A_1), \mathcal{L}(A_2)$ disjoint WSTS languages.

Regular separability

WSTS = class of languages of finitely branching well-structured transition systems e.g. Petri nets with coverability

Theorem

If two WSTS languages are disjoint, then they are regularly separable.

Proof:

Given $\mathcal{L}(A_1), \mathcal{L}(A_2)$ disjoint WSTS languages.

1. Show that we can assume wlog. that A_2 is deterministic.

Regular separability

WSTS = class of languages of finitely branching well-structured transition systems e.g. Petri nets with coverability

Theorem

If two WSTS languages are disjoint, then they are regularly separable.

Proof:

Given $\mathcal{L}(A_1), \mathcal{L}(A_2)$ disjoint WSTS languages.

1. Show that we can assume wlog. that A_2 is deterministic.
2. Find safe inductive invariant for $A_1 \times A_2$.

Regular separability

WSTS = class of languages of finitely branching well-structured transition systems e.g. Petri nets with coverability

Theorem

If two WSTS languages are disjoint, then they are regularly separable.

Proof:

Given $\mathcal{L}(A_1), \mathcal{L}(A_2)$ disjoint WSTS languages.

1. Show that we can assume wlog. that A_2 is deterministic.
2. Find safe inductive invariant for $A_1 \times A_2$.
3. Find a finite representation of the invariant using ideals.

Regular separability

WSTS = class of languages of finitely branching well-structured transition systems e.g. Petri nets with coverability

Theorem

If two WSTS languages are disjoint, then they are regularly separable.

Proof:

Given $\mathcal{L}(A_1), \mathcal{L}(A_2)$ disjoint WSTS languages.

1. Show that we can assume wlog. that A_2 is deterministic.
2. Find safe inductive invariant for $A_1 \times A_2$.
3. Find a finite representation of the invariant using ideals.
4. Convert this representation into an NFA defining a regular separator.

Closures

Part IV. of the thesis

Publication:

W. Czerwiński, S. Lasota, R. Meyer, *S. M*, K N. Kumar, and P. Saivasan
Regular separability of well-structured transition Systems
In: CONCUR 2018, volume 118 of LIPIcs, pages 35:1–35:18

3rd example: Synthesis & Games

Synthesis

Verification: Checking whether program is correct

Synthesis

Synthesis: Constructing a correct program

Synthesis

Synthesis: Constructing a correct program from program template

Synthesis

Synthesis: Constructing a correct program from program template

Two player game:

- Environment player: Non-determinism in the program
- Synthesis player: Replacing wildcards

Synthesis

Synthesis: Constructing a correct program from program template

Two player game:

- Environment player: Non-determinism in the program
- Synthesis player: Replacing wildcards

```
if (x == 0)
  code₁
else
  code₂
```

Synthesis

Synthesis: Constructing a correct program from program template

Two player game:

- Environment player: Non-determinism in the program
- Synthesis player: Replacing wildcards

```
if (x == 0)
  code₁
else
  code₂
```

$\text{assert}(x = 0).\texttt{code}_1 \wedge$
$\text{assert}(x \neq 0).\texttt{code}_2$

Synthesis

Synthesis: Constructing a correct program from program template

Two player game:

Environment player: Non-determinism in the program

Synthesis player: Replacing wildcards

```
if (x == 0)
  code₁
else
  code₂
```

```
if (???)
  code₁
else
  code₂
```

$\text{assert}(x = 0).\texttt{code}_1 \land$
$\text{assert}(x \neq 0).\texttt{code}_2$

Synthesis

Synthesis: Constructing a correct program from program template

Two player game:

Environment player: Non-determinism in the program

Synthesis player: Replacing wildcards

```
if (x == 0)
  code₁
else
  code₂
```

```
if (???)
  code₁
else
  code₂
```

assert($x = 0$).$\mathtt{code}_1 \wedge$
assert($x \neq 0$).$\mathtt{code}_2$

$\mathtt{code}_1 \vee$
$\mathtt{code}_2$

Synthesis

Synthesis: Constructing a correct program from program template

Two player game:

Environment player: Non-determinism in the program

Synthesis player: Replacing wildcards

```
if (x == 0)
  code₁
else
  code₂
```

$\text{assert}(x = 0).\texttt{code}_1 \wedge$
$\text{assert}(x \neq 0).\texttt{code}_2$

```
if (???)
  code₁
else
  code₂
```

$\texttt{code}_1 \vee$
$\texttt{code}_2$

Certificate: Winning strategy $\triangleq$ Instantiation of the template

Solving a game

Given: Game system, specification Spec

Question: Has the synthesis player a strategy s so that Game @ $s \models$ Spec?

Solving an inclusion game

Given: Game system, specification Spec

Question: Has the synthesis player a strategy s so that $\mathcal{L}(\text{Game} @ s) \subseteq \mathcal{L}(\text{Spec})$?

Solving an inclusion game

Given: Game system, specification Spec

Question: Has the synthesis player a strategy s so that $\mathcal{L}(\text{CF-Game} @ s) \subseteq \mathcal{L}(\text{NFA})$?

Problem 1: Game is context-free
(it models the control flow of a program)

Solving an inclusion game

Given: Game system, specification Spec

Question: Has the synthesis player a strategy s so that $\mathcal{L}(\text{CF-Game} @ s) \subseteq \mathcal{L}(\text{NFA})$?

Problem 1: Game is context-free
(it models the control flow of a program)

Solution: Various algorithms for games on context-free systems

- Guess-and-check [Walukiewicz 1996]
- Alternating two-way automata [KV 2000]
- Saturation [Cachat 2002]

Solving an inclusion game

Given: Game system, specification Spec

Question: Has the synthesis player a strategy s so that $\mathcal{L}(\text{CF-Game} @ s) \subseteq \mathcal{L}(\text{NFA})$?

Problem 2: Specification is given as NFA

Solving an inclusion game

Given: Game system, specification Spec

Question: Has the synthesis player a strategy s so that $\mathcal{L}(\text{CF-Game} @ s) \subseteq \mathcal{L}(\text{NFA})$?

Problem 2: Specification is given as NFA

Three entities make decisions:

1) System player chooses (a part of) the behavior of Game
2) Environment player chooses (a part of) the behavior of Game
3) NFA chooses the behavior of the automaton for Spec
   the choices are invisible to the other players!

Solving an inclusion game

Given: Game system, specification Spec

Question: Has the synthesis player a strategy s so that $\mathcal{L}(\text{CF-Game} @ s) \subseteq \mathcal{L}(\text{NFA})$?

Succinct context-free inclusion game

Left-hand side: Context-free game grammar

Right-hand side: Non-deterministic automaton

Solving an inclusion game

Given: Game system, specification Spec

Question: Has the synthesis player a strategy s so that $\mathcal{L}(\text{CF-Game} @ s) \subseteq \mathcal{L}(\text{NFA})$?

Succinct context-free inclusion game

Left-hand side: Context-free game grammar

Right-hand side: Non-deterministic automaton

Existing techniques require an upfront determinization:

Construct DFA with $\mathcal{L}(\text{DFA}) = \mathcal{L}(\text{NFA})$ and consider CF-Game $\times$ DFA

Upfront determinization, leading to an exponential blowup

Games

Given:

Context-free game grammar (representing the game),
NFA (representing the Spec)

Effective denotational semantics

Games

Given:

Context-free game grammar (representing the game),
NFA (representing the Spec)

Effective denotational semantics

1. See grammar as a system of equations
   using three operations
   - choices of the system player
   - choices of the environment player
   - concatenation

Games

Given:

Context-free game grammar (representing the game),
NFA (representing the Spec)

Effective denotational semantics

1. See grammar as a system of equations
2. Solve the system of equations
   using Boolean formulas over the transition monoid
   - represent terminals by their effect on the automaton
   - represent choices of the system by conjunction
   - represent choices of the environment by disjunction
   - represent concatenation by formula composition

Games

Given:

Context-free game grammar (representing the game),
NFA (representing the Spec)

Effective denotational semantics

1. See grammar as a system of equations
2. Solve the system of equations
3. Least solution associates to each non-terminal a formula
   - represents the effect of the game on the automaton
   - winning regions can be read-off
   - winning strategies can be read-off

Given:

Context-free game grammar (representing the game),
NFA (representing the Spec)

Effective denotational semantics

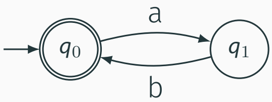
1. See grammar as a system of equations
2. Solve the system of equations
3. Least solution associates to each non-terminal a formula

Advantages:

- On-the-fly determinization
- Reduce to a well-understood subproblem
- Prototype implementation performs better than competitors (Problem is 2EXPTIME-complete!)

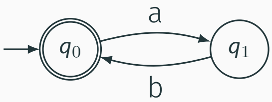
Example:

$$X_{\text{Synth}} \to a.Y \mid \varepsilon$$
$$Y_{\text{Env}} \to b.X$$



Example:

$$X_{\text{Synth}} \to a.Y \mid \varepsilon$$
$$Y_{\text{Env}} \to b.X$$



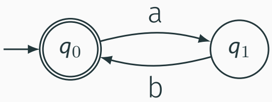
System of equations:

$$X = a.Y \mid_{\text{Synth}} \varepsilon$$
$$Y = b.X$$

Games

Example:

$$X_{\text{Synth}} \rightarrow a.Y \mid \varepsilon$$
$$Y_{\text{Env}} \rightarrow b.X$$



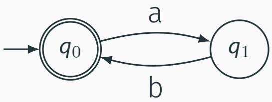
System of equations:

$$X = [a]; Y \vee [\varepsilon]$$
$$Y = [b]; X$$

Games

Example:

$$X_{\text{Synth}} \to a.Y \mid \varepsilon$$
$$Y_{\text{Env}} \to b.X$$



System of equations:

$$X = [a]; Y \vee [\varepsilon]$$
$$Y = [b]; X$$

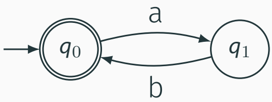
Iteration:

Nr.	X	Y

Games

Example:

$$X_{\text{Synth}} \to a.Y \mid \varepsilon$$
$$Y_{\text{Env}} \to b.X$$



System of equations:

$$X = [a]; Y \vee [\varepsilon]$$
$$Y = [b]; X$$

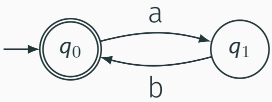
Iteration:

Nr.	X	Y
0	*false*	*false*

Games

Example:

$$X_{\text{Synth}} \to a.Y \mid \varepsilon$$
$$Y_{\text{Env}} \to b.X$$



Iteration:

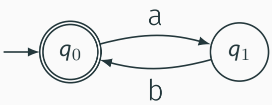
Nr.	X	Y
0	*false*	*false*
1	$[\varepsilon]$	*false*

System of equations:

$$X = [a]; Y \vee [\varepsilon]$$
$$Y = [b]; X$$

Example:

$$X_{\text{Synth}} \rightarrow a.Y \mid \varepsilon$$
$$Y_{\text{Env}} \rightarrow b.X$$



System of equations:

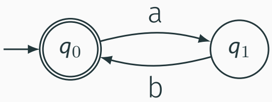
$$X = [a]; Y \vee [\varepsilon]$$
$$Y = [b]; X$$

Iteration:

Nr.	X	Y
0	*false*	*false*
1	$[\varepsilon]$	*false*
2	$[\varepsilon]$	$[b]; [\varepsilon] = [b]$

Example:

$$X_{\text{Synth}} \to a.Y \mid \varepsilon$$
$$Y_{\text{Env}} \to b.X$$



System of equations:

$$X = [a]; Y \vee [\varepsilon]$$
$$Y = [b]; X$$

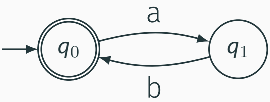
Iteration:

Nr.	X	Y
0	*false*	*false*
1	$[\varepsilon]$	*false*
2	$[\varepsilon]$	$[b]; [\varepsilon] = [b]$
3	$[ab] \vee [\varepsilon]$	$[b]$

Games

Example:

$$X_{\text{Synth}} \rightarrow a.Y \mid \varepsilon$$
$$Y_{\text{Env}} \rightarrow b.X$$



System of equations:

$$X = [a]; Y \vee [\varepsilon]$$
$$Y = [b]; X$$

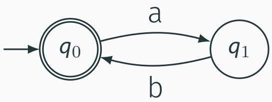
Iteration:

Nr.	X	Y
0	*false*	*false*
1	$[\varepsilon]$	*false*
2	$[\varepsilon]$	$[b];[\varepsilon] = [b]$
3	$[ab] \vee [\varepsilon]$	$[b]$
4	$[ab] \vee [\varepsilon]$	$[b];([ab] \vee [\varepsilon])$

Games

Example:

$$X_{\text{Synth}} \to a.Y \mid \varepsilon$$
$$Y_{\text{Env}} \to b.X$$



System of equations:

$$X = [a]; Y \vee [\varepsilon]$$
$$Y = [b]; X$$

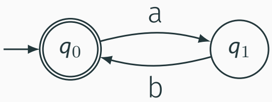
Iteration:

Nr.	X	Y
0	*false*	*false*
1	$[\varepsilon]$	*false*
2	$[\varepsilon]$	$[b]; [\varepsilon] = [b]$
3	$[ab] \vee [\varepsilon]$	$[b]$
4	$[ab] \vee [\varepsilon]$	$[b]; ([ab] \vee [\varepsilon])$ $= [bab] \vee [b]$

Games

Example:

$$X_{\text{Synth}} \to a.Y \mid \varepsilon$$
$$Y_{\text{Env}} \to b.X$$



System of equations:

$$X = [a]; Y \vee [\varepsilon]$$
$$Y = [b]; X$$

Iteration:

Nr.	X	Y
0	*false*	*false*
1	$[\varepsilon]$	*false*
2	$[\varepsilon]$	$[b];[\varepsilon] = [b]$
3	$[ab] \vee [\varepsilon]$	$[b]$
4	$[ab] \vee [\varepsilon]$	$[b];([ab] \vee [\varepsilon])$ $= [bab] \vee [b]$ $\Leftrightarrow [b]$

Games

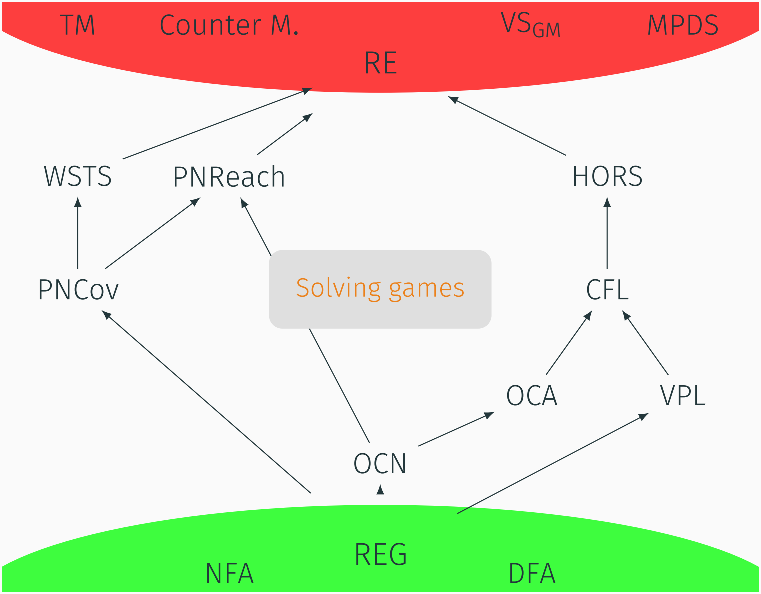
Part V. of the thesis

Effective denotational semantics for context-free games

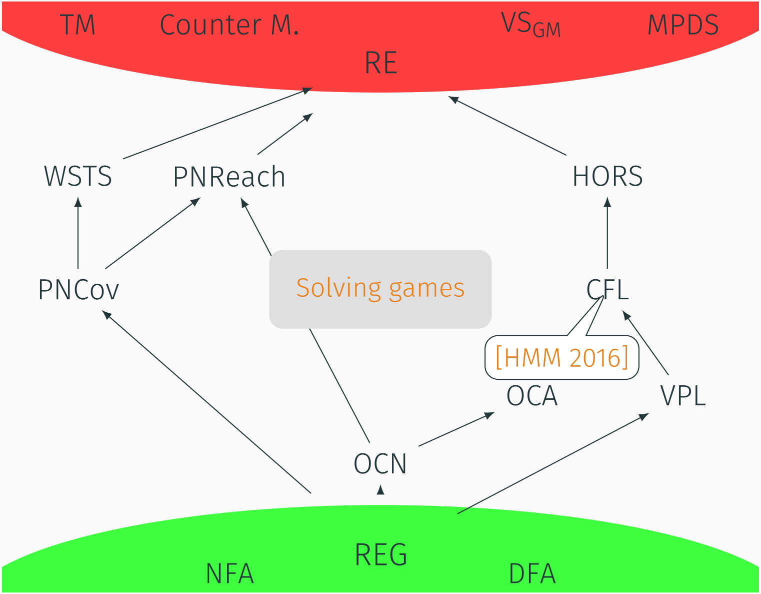
Publication:

L. Holík, R. Meyer, and *S. M.*
Summaries for context-free games
In: FSTTCS 2016, volume 65 of LIPIcs, pages 41:1–41:16

Games


TM
Counter M.
VS_{GM}
MPDS
RE
WSTS
PNReach
HORS
PNCov
Solving games
CFL
OCA
VPL
OCN
REG
NFA
DFA

Games



TM
Counter M.
VS_{GM}
MPDS
RE
WSTS
PNReach
HORS
PNCov
Solving games
CFL
[HMM 2016]
OCA
VPL
OCN
REG
NFA
DFA

Games

Part V. of the thesis

Extensions to games with infinite executions (ω-languages)

Publication:

R. Meyer, *S. M.*, and E. Neumann
Liveness verification and synthesis:
New algorithms for recursive programs
Unpublished preprint (available on arXiv)

Games

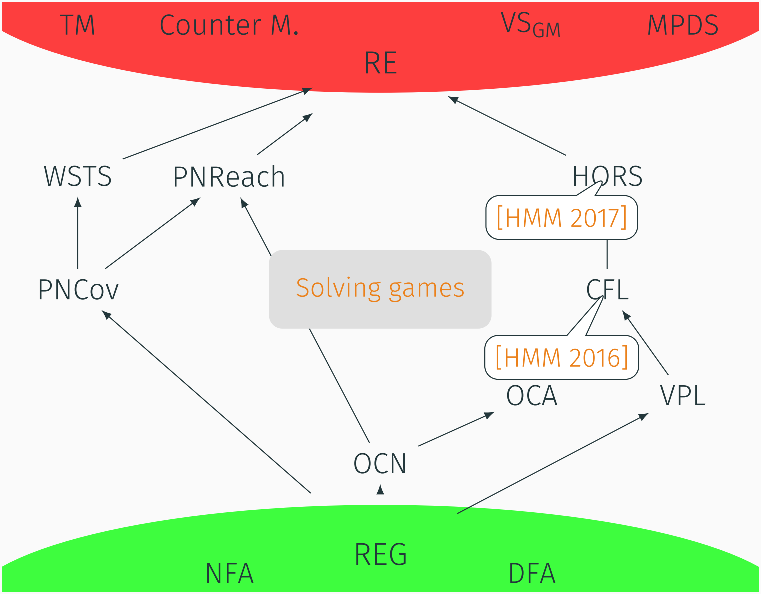
Part V. of the thesis

Extensions to higher-order recursion schemes (HORSes)

Publication:

M. Hague, R. Meyer, and *S. M.*
Domains for higher-order games
In: MFCS 2017, volume 83 of LIPIcs, pages 59:1–59:15

Games



TM
Counter M.
VS_{GM}
MPDS
RE
WSTS
PNReach
HORS
[HMM 2017]
PNCov
Solving games
CFL
[HMM 2016]
OCA
VPL
OCN
REG
NFA
DFA

Part V. of the thesis

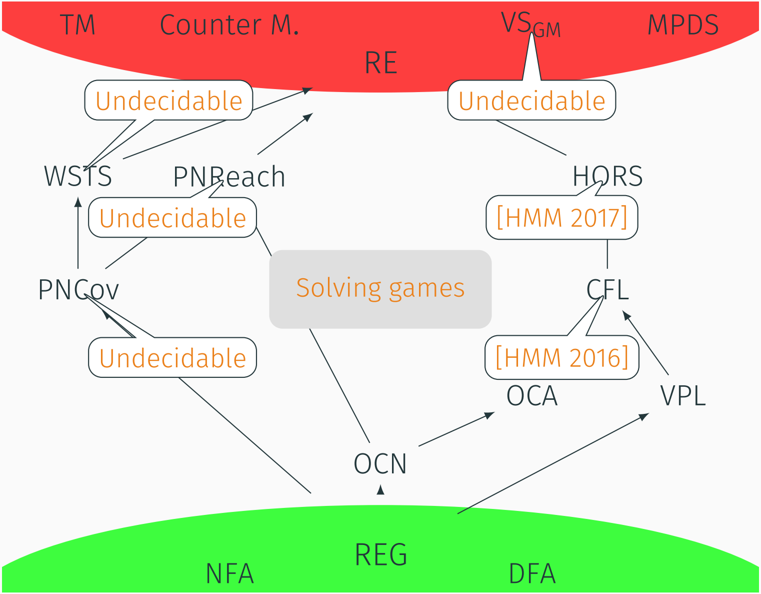
The frontier of the decidability of games

Publication:

R. Meyer, *S. M.*, and G. Zetzsche
Bounded context switching for valence systems
In: CONCUR 2018, volume 118 of LIPIcs, pages 12:1–12:18

+ unpublished work

Games


TM
Counter M.
VS_{GM}
MPDS
RE
Undecidable
Undecidable
WSTS
PNReach
HORS
Undecidable
[HMM 2017]
PNCov
Solving games
CFL
Undecidable
[HMM 2016]
OCA
VPL
OCN
REG
NFA
DFA

Conclusion

Conclusion

In the thesis

Certificates for automata in a hostile environment

we have presented certificate-producing procedures

(1) for computing the closures of Petri net languages
modeling the visible behavior under lossiness/gaininess,

(2) for the regular separability of WSTS languages
with applications in compositional verification,

(3) solving inclusion games
using effective denotational semantics
with applications in program synthesis.

Conclusion

The work constituting the thesis has resulted in

- 5 peer-reviewed conference publications,
- 1 unpublished preprints,
- ongoing work on these subjects.

Thank you!