Parameterized Synthesis for Fragments of First-Order Logic over Data Words

Mathieu Lehaut[0]
with Béatrice Bérard[1], Benedikt Bollig[2], Tali Sznajder[1]

[0]University of Gothenburg, Gothenburg, Sweden
[1]Sorbonne Université, CNRS, LIP6, F-75005 Paris, France
[2]CNRS, LSV & ENS Paris-Saclay, Université Paris-Saclay, Cachan, France

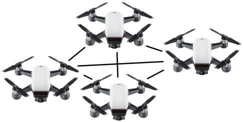
16/06/2021

The context

Distributed systems everywhere:

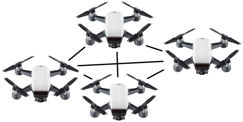
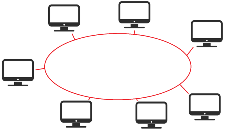
The context

Distributed systems everywhere:



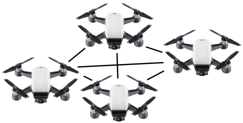
The context

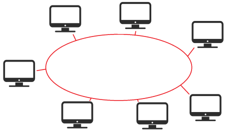
Distributed systems everywhere:

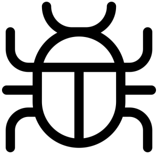
The context

Distributed systems everywhere:



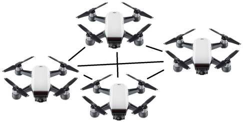
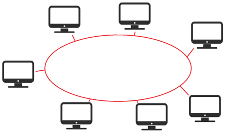


And bugs too.

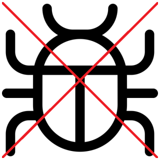


The context

Distributed systems everywhere:

And bugs too.



What kind of distributed systems?

Parameterized systems

Distributed systems with number of processes not known in advance

What kind of distributed systems?

Parameterized systems

Distributed systems with number of processes not known in advance

Open systems

Each process interacts with uncontrollable environment
(sensors, operator inputs, environmental conditions, ...)

A technique of verification

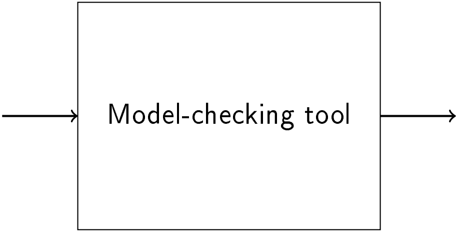
Model checking [Clarke, Emerson, Sifakis]

I: A specification S, a model $\mathcal{M}$
O: $\mathcal{M} \models S$?

A technique of verification

Model checking [Clarke, Emerson, Sifakis]

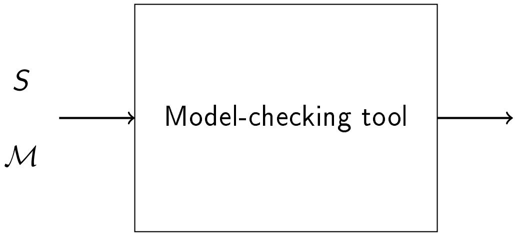
I: A specification S, a model $\mathcal{M}$
O: $\mathcal{M} \models S$?



A technique of verification

Model checking [Clarke, Emerson, Sifakis]

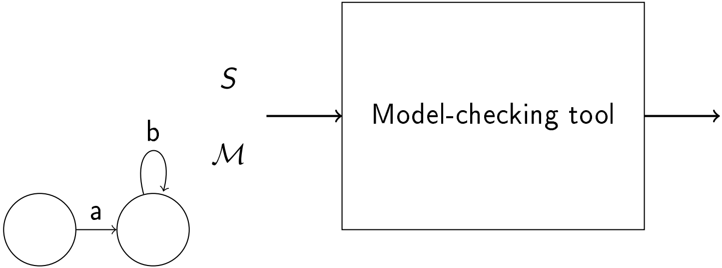
I: A specification S, a model $\mathcal{M}$
O: $\mathcal{M} \models S$?



A technique of verification

Model checking [Clarke, Emerson, Sifakis]

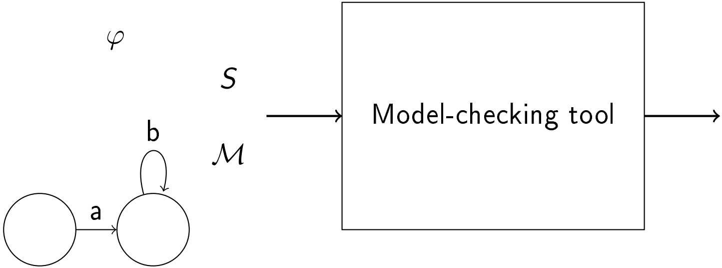
I: A specification S, a model $\mathcal{M}$
O: $\mathcal{M} \models S$?



A technique of verification

Model checking [Clarke, Emerson, Sifakis]

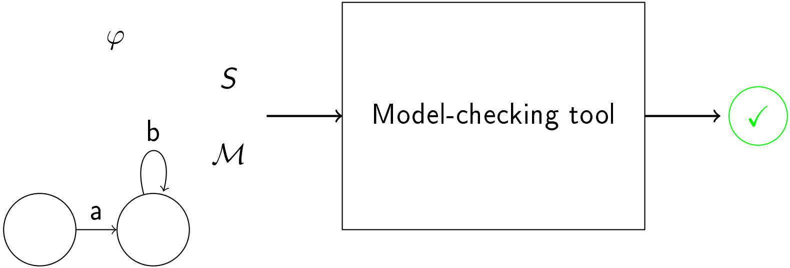
I: A specification S, a model $\mathcal{M}$
O: $\mathcal{M} \models S$?



A technique of verification

Model checking [Clarke, Emerson, Sifakis]

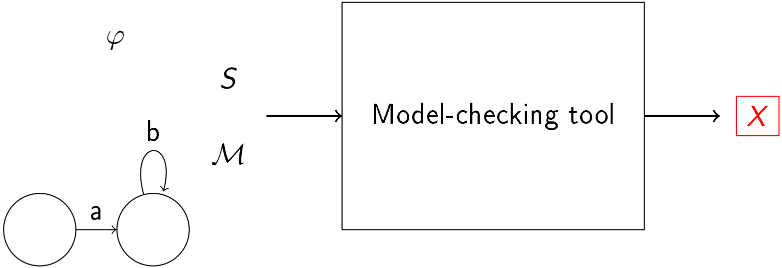
I: A specification S, a model $\mathcal{M}$
O: $\mathcal{M} \models S$?



A technique of verification

Model checking [Clarke, Emerson, Sifakis]

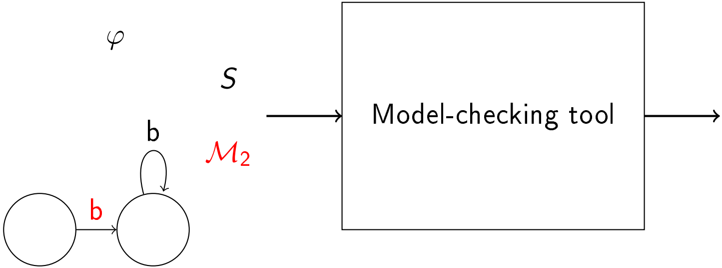
I: A specification S, a model $\mathcal{M}$
O: $\mathcal{M} \models S$?



A technique of verification

Model checking [Clarke, Emerson, Sifakis]

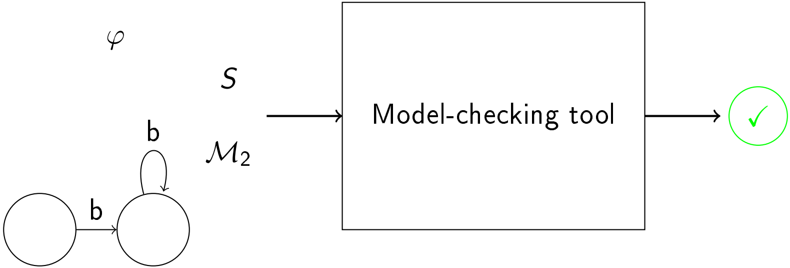
I: A specification S, a model $\mathcal{M}$
O: $\mathcal{M} \models S$?



A technique of verification

Model checking [Clarke, Emerson, Sifakis]

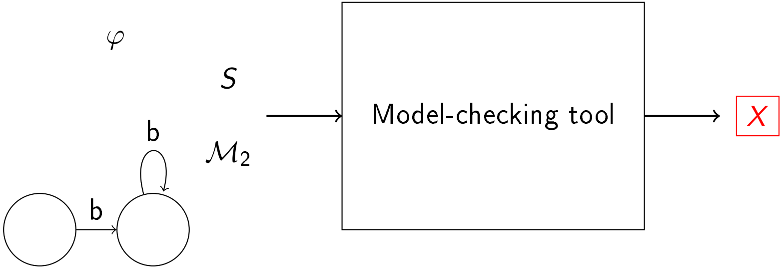
I: A specification S, a model $\mathcal{M}$
O: $\mathcal{M} \models S$?



A technique of verification

Model checking [Clarke, Emerson, Sifakis]

I: A specification S, a model $\mathcal{M}$
O: $\mathcal{M} \models S$?

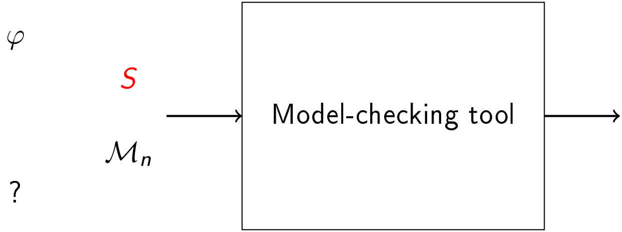


A technique of verification

Model checking [Clarke, Emerson, Sifakis]

I: A specification S, a model $\mathcal{M}$

O: $\mathcal{M} \models S$?



What about synthesis?

Synthesis

I: A specification S

What about synthesis?

Synthesis

I: A specification S

O: $\mathcal{M}$ s.t. $\mathcal{M} \models S$ if it exists

What about synthesis?

Synthesis

I: A specification S
O: $\mathcal{M}$ s.t. $\mathcal{M} \models S$ if it exists

- But first, need to define possible executions.

Executions I: Data words

Behaviors for $A = \{req, ack\}$

- 1 process: $w = req \ ack \ req \ ack$

Executions I: Data words

Behaviors for $A = \{req, ack\}$

- 1 process: $w = req\ ack\ req\ ack$
- fixed number of processes: $w = req_1 req_3 ack_1 ack_3$

Executions I: Data words

Behaviors for $A = \{req, ack\}$

- 1 process: $w = req\ ack\ req\ ack$
- fixed number of processes: $w = req_1 req_3 ack_1 ack_3$
- unknown (not bounded) number of processes:
  $w = (req, 1)(req, 3)(ack, 1)(req, 6)(ack, 6)(ack, 3)$

Executions I: Data words

Behaviors for $A = \{req, ack\}$

- 1 process: $w = req\ ack\ req\ ack$
- fixed number of processes: $w = req_1 req_3 ack_1 ack_3$
- unknown (not bounded) number of processes: $w = (req, 1)(req, 3)(ack, 1)(req, 6)(ack, 6)(ack, 3)$

Data words [Bojanczyk et al., 2006]

- A: finite alphabet (*actions*),
- $\mathcal{D}$: infinite set of data values (*process identities*)

Data word: (in)finite word over $A \times \mathcal{D}$

Executions II: System vs Environment

System actions and Environment actions

- $A = A_{sys} \uplus A_{env}$

Executions II: System vs Environment

System actions and Environment actions

- $A = A_{sys} \uplus A_{env}$

System and Environment processes

- $\mathbb{P} = (\mathbb{P}_{sys}, \mathbb{P}_{env}, \mathbb{P}_{se})$

with $\mathbb{P}_\theta$ finite set of processes

Executions II: System vs Environment

System actions and Environment actions

- $A = A_{sys} \uplus A_{env}$

System and Environment processes

- $\mathbb{P} = (\mathbb{P}_{sys}, \mathbb{P}_{env}, \mathbb{P}_{se})$

with $\mathbb{P}_\theta$ finite set of processes

$$\Sigma_{sys} = A_{sys} \times (\mathbb{P}_{sys} \cup \mathbb{P}_{se})$$

Executions II: System vs Environment

System actions and Environment actions

- $A = A_{sys} \uplus A_{env}$

System and Environment processes

- $\mathbb{P} = (\mathbb{P}_{sys}, \mathbb{P}_{env}, \mathbb{P}_{se})$

with $\mathbb{P}_\theta$ finite set of processes

$$\Sigma_{sys} = A_{sys} \times (\mathbb{P}_{sys} \cup \mathbb{P}_{se})$$
$$\Sigma_{env} = A_{env} \times (\mathbb{P}_{env} \cup \mathbb{P}_{se})$$

Executions II: System vs Environment

System actions and Environment actions

- $A = A_{sys} \uplus A_{env}$

System and Environment processes

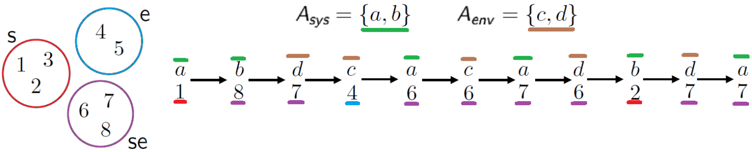
- $\mathbb{P} = (\mathbb{P}_{sys}, \mathbb{P}_{env}, \mathbb{P}_{se})$

with $\mathbb{P}_\theta$ finite set of processes

$$\Sigma_{sys} = A_{sys} \times (\mathbb{P}_{sys} \cup \mathbb{P}_{se})$$
$$\Sigma_{env} = A_{env} \times (\mathbb{P}_{env} \cup \mathbb{P}_{se})$$

$\mathbb{P}$-Execution

Execution = word over $\Sigma_{sys} \cup \Sigma_{env}$



Executions III: Strategies

- *Asynchronous* synthesis problem

Strategy for System

$f : \Sigma^* \to \Sigma_{sys} \cup \{\varepsilon\}$

Executions III: Strategies

- *Asynchronous* synthesis problem

Strategy for System

$f : \Sigma^* \to \Sigma_{sys} \cup \{\varepsilon\}$

An execution is

- f-compatible if System actions follow f
- f-fair if Environment does not always block System

Executions III: Strategies

- *Asynchronous* synthesis problem

Strategy for System

$f : \Sigma^* \to \Sigma_{sys} \cup \{\varepsilon\}$

An execution is

- f-compatible if System actions follow f
- f-fair if Environment does not always block System

Winning strategy

f is winning for a set S of executions if all f-compatible, f-fair executions are in S

Executions III: Strategies

- *Asynchronous* synthesis problem

Strategy for System

$f : \Sigma^* \to \Sigma_{sys} \cup \{\varepsilon\}$

An execution is

- f-compatible if System actions follow f
- f-fair if Environment does not always block System

Winning strategy

f is winning for a set S of executions if all f-compatible, f-fair executions are in S?

First order logic I: Definition

Example on words

$\varphi = \forall x. (req(x) \Rightarrow \exists y.(y > x \wedge ack(y)))$

First order logic I: Definition

Example on words

$\varphi = \forall x. (req(x) \Rightarrow \exists y.(y > x \wedge ack(y)))$
"every *req* is eventually followed by an *ack*"

First order logic I: Definition

Example on words

$\varphi = \forall x. (req(x) \Rightarrow \exists y.(y > x \wedge ack(y)))$
"every *req* is eventually followed by an *ack*"

- $req\ ack\ req\ req\ ack \models \varphi$
- $req\ req\ ack\ req \not\models \varphi$

First order logic I: Definition

Example on words

$\varphi = \forall x. (req(x) \Rightarrow \exists y.(y > x \wedge ack(y)))$
"every *req* is eventually followed by an *ack*"

- *req ack req req ack* $\models \varphi$
- *req req ack req* $\not\models \varphi$

Syntax for FO on words

Basic formulas: $a(x) \mid x = y \mid x < y \mid \mathrm{succ}(x, y)$
$a \in A$
Connectors and quantifiers: $\neg, \vee, \wedge, \Rightarrow, \exists, \forall$

First order logic I: Definition

Examples on data words

$\varphi = \forall x.(req(x) \Rightarrow \exists y.(y \sim x \wedge y > x \wedge ack(y)))$
"every *req* is eventually followed by an *ack* on the same process"

First order logic I: Definition

Examples on data words

$\varphi = \forall x.(req(x) \Rightarrow \exists y.(y \sim x \wedge y > x \wedge ack(y)))$
"every *req* is eventually followed by an *ack* on the same process"

- $(req, 1)(req, 3)(ack, 1)(req, 6)(ack, 6)(ack, 3) \models \varphi$
- $(req, 1)(ack, 2)(req, 1)(ack, 2) \cdots \not\models \varphi$

First order logic I: Definition

Examples on data words

$\varphi = \forall x.(req(x) \Rightarrow \exists y.(y \sim x \wedge y > x \wedge ack(y)))$
"every *req* is eventually followed by an *ack* on the same process"

- $(req, 1)(req, 3)(ack, 1)(req, 6)(ack, 6)(ack, 3) \models \varphi$
- $(req, 1)(ack, 2)(req, 1)(ack, 2) \cdots \not\models \varphi$

Syntax for FO on datawords

Basic formulas: $a(x) \mid x = y \mid x < y \mid \mathrm{succ}(x, y) \mid \theta(x) \mid x \sim y$
$a \in A$, $\theta \in \{sys, env, se\}$
Connectors and quantifiers: $\neg, \vee, \wedge, \Rightarrow, \exists, \forall$

First order logic II: Satisfiability

- Specification $S_\varphi = \{w \mid w \models \varphi\}$

First order logic II: Satisfiability

- Specification $S_\varphi = \{w \mid w \models \varphi\}$

Satisfiability

I: A first order formula φ
O: $S_\varphi \neq \emptyset$?

First order logic II: Satisfiability

- Specification $S_\varphi = \{w \mid w \models \varphi\}$

Satisfiability

I: A first order formula φ
O: $S_\varphi \neq \emptyset$?

- Decidable for words (but non-elementary) [Büchi, 60]

First order logic II: Satisfiability

- Specification $S_\varphi = \{w \mid w \models \varphi\}$

Satisfiability

I: A first order formula φ
O: $S_\varphi \neq \emptyset$?

- Decidable for words (but non-elementary) [Büchi, 60]
- Undecidable for data words [Neven et al., 04]

Winning triples

- Only important point for synthesis is number of processes, not concrete identities!

Winning triples

- Only important point for synthesis is number of processes, not concrete identities!

Winning triples for φ

$(n_{sys}, n_{env}, n_{se}) \in \mathbb{N}^3$ is a winning triple if there is a winning strategy for data words limited to $(n_{sys}, n_{env}, n_{se})$ processes

Winning triples

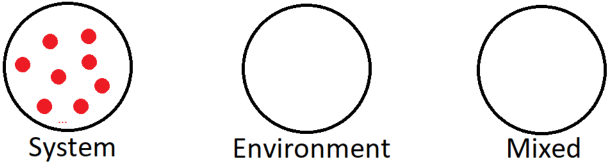
- Only important point for synthesis is number of processes, not concrete identities!

Winning triples for φ

$(n_{sys}, n_{env}, n_{se}) \in \mathbb{N}^3$ is a winning triple if there is a winning strategy for data words limited to $(n_{sys}, n_{env}, n_{se})$ processes

Intersection of set of winning triples $Win(\varphi)$ with:

$\mathbb{N} \times \{0\} \times \{0\}$: only System processes (satisfiability)



Winning triples

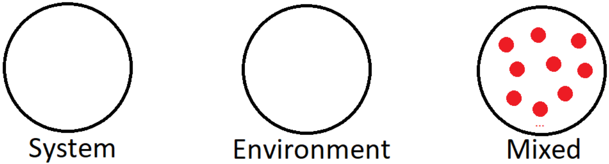
- Only important point for synthesis is number of processes, not concrete identities!

Winning triples for φ

$(n_{sys}, n_{env}, n_{se}) \in \mathbb{N}^3$ is a winning triple if there is a winning strategy for data words limited to $(n_{sys}, n_{env}, n_{se})$ processes

Intersection of set of winning triples $Win(\varphi)$ with:

$\{0\} \times \{0\} \times \mathbb{N}$: each process controlled by both System and Environment



Winning triples

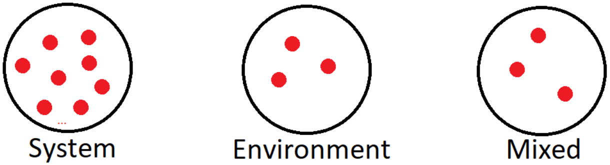
- Only important point for synthesis is number of processes, not concrete identities!

Winning triples for φ

$(n_{sys}, n_{env}, n_{se}) \in \mathbb{N}^3$ is a winning triple if there is a winning strategy for data words limited to $(n_{sys}, n_{env}, n_{se})$ processes

Intersection of set of winning triples $Win(\varphi)$ with:

$\mathbb{N} \times \{k_{env}\} \times \{k_{se}\}$: constant number of Environment and mixed processes, but unboundedly many System processes



Parameterized synthesis problem

SYNTH($\mathcal{F}, (\mathcal{N}_{sys}, \mathcal{N}_{env}, \mathcal{N}_{se})$)

I: Alphabet $A = A_{sys} \uplus A_{env}$, formula $\varphi \in \mathcal{F}$ over A
O: $Win(\varphi) \cap (\mathcal{N}_{sys} \times \mathcal{N}_{env} \times \mathcal{N}_{se}) \neq \emptyset$?

Parameterized synthesis problem

Example 1

$\varphi_1 = \forall x.(req(x) \Rightarrow \exists y.(y \sim x \wedge y > x \wedge ack(y)))$

- $A_{sys} = \{ack\}$,
- $A_{env} = \{req\}$,
- $(\mathcal{N}_{sys}, \mathcal{N}_{env}, \mathcal{N}_{se}) = (\{0\}, \{0\}, \mathbb{N})$

Parameterized synthesis problem

Example 1

$\varphi_1 = \forall x.(req(x) \Rightarrow \exists y.(y \sim x \wedge y > x \wedge ack(y)))$

- $A_{sys} = \{ack\}$,
- $A_{env} = \{req\}$,
- $(\mathcal{N}_{sys}, \mathcal{N}_{env}, \mathcal{N}_{se}) = (\{0\}, \{0\}, \mathbb{N})$

▶ $(0, 0, k)$ is a winning triple for φ_1 for all $k \in \mathbb{N}$:

Winning strategy

$f(w) = (ack, i)$ s.t. $\sigma = (req, i)$ is the first pending *req* of w

Parameterized synthesis problem

Example 2

$\varphi_2 = (\neg\exists x.a(x)) \Leftrightarrow (\forall y.sys(y) \Rightarrow \exists z.z \sim y \wedge b(z))$

- $A_{sys} = \{b\}$,
- $A_{env} = \{a\}$,
- $(\mathcal{N}_{sys}, \mathcal{N}_{env}, \mathcal{N}_{se}) = (\mathbb{N}, \{k_{env}\}, \{k_{se}\})$

Parameterized synthesis problem

Example 2

$\varphi_2 = (\neg\exists x.a(x)) \Leftrightarrow (\forall y.sys(y) \Rightarrow \exists z.z \sim y \wedge b(z))$

- $A_{sys} = \{b\}$,
- $A_{env} = \{a\}$,
- $(\mathcal{N}_{sys}, \mathcal{N}_{env}, \mathcal{N}_{se}) = (\mathbb{N}, \{k_{env}\}, \{k_{se}\})$

- No winning triple unless $k_{env} = k_{se} = 0$!

Two-variable first-order logic: FO^2

- FO^2: restrict to two variable names

Two-variable first-order logic: FO^2

- FO^2: restrict to two variable names

Examples

- $\exists x, y, z. \neg(x \sim y) \wedge \neg(y \sim z) \wedge \neg(x \sim z) \notin \mathrm{FO}^2$

Two-variable first-order logic: FO^2

- FO^2: restrict to two variable names

Examples

- $\exists x, y, z. \neg(x \sim y) \wedge \neg(y \sim z) \wedge \neg(x \sim z) \notin \mathrm{FO}^2$
- $\exists x. a(x) \wedge (\exists y. x < y \wedge a(y) \wedge (\exists x. y < x \wedge a(x))) \in \mathrm{FO}^2$

Two-variable first-order logic: FO^2

- FO^2: restrict to two variable names

Examples

- $\exists x, y, z. \neg(x \sim y) \wedge \neg(y \sim z) \wedge \neg(x \sim z) \notin \mathrm{FO}^2$
- $\exists x. a(x) \wedge (\exists y. x < y \wedge a(y) \wedge (\exists x. y < x \wedge a(x))) \in \mathrm{FO}^2$

- Satisfiability is decidable! [Bojanczyk et al., 06]

Results for FO^2

Theorem [FoSSaCS 20]

SYNTH($\mathrm{FO}^2, (\{0\}, \{0\}, \mathbb{N})$) is undecidable

Results for FO^2

Theorem [FoSSaCS 20]

SYNTH($\mathrm{FO}^2, (\{0\}, \{0\}, \mathbb{N})$) is undecidable

Proof

Adapt proof of [Figueira and Praveen, 18] to reduce halting problem for D2CM:

- Counters value encoded by number of processes with an action from System but not Environment (and vice versa)
- FO^2 formula to enforce simulation of a run

FO[$\sim$]

- FO[$\sim$] = FO without $<$ and succ

$$\exists x.\mathit{bcast}(x) \wedge \forall y.(y \not\sim x \Rightarrow \exists z.(z \sim y \wedge \mathit{rcv}(z)))$$

FO[$\sim$]

- $\mathrm{FO}[\sim] = \mathrm{FO}$ without $<$ and succ

$$\exists x. bcast(x) \wedge \forall y.(y \not\sim x \Rightarrow \exists z.(z \sim y \wedge rcv(z)))$$

Some remarks

- No way to specify an order

FO[∼]

- $\mathrm{FO}[\sim] = \mathrm{FO}$ without $<$ and succ

$$\exists x. bcast(x) \wedge \forall y.(y \not\sim x \Rightarrow \exists z.(z \sim y \wedge rcv(z)))$$

Some remarks

- No way to specify an order
- Can count letters on a given class up to some bound B

FO[$\sim$]

- FO[$\sim$] = FO without $<$ and succ

$$\exists x.bcast(x) \wedge \forall y.(y \not\sim x \Rightarrow \exists z.(z \sim y \wedge rcv(z)))$$

Some remarks

- No way to specify an order
- Can count letters on a given class up to some bound B
- Can count such classes up to some number

FO[$\sim$]

- FO[$\sim$] = FO without $<$ and succ

$$\exists x.bcast(x) \wedge \forall y.(y \not\sim x \Rightarrow \exists z.(z \sim y \wedge rcv(z)))$$

Some remarks

- No way to specify an order
- Can count letters on a given class up to some bound B
- Can count such classes up to some number

Roadmap

1. Establish normal form for FO[$\sim$]

$\mathrm{FO}[\sim]$

- $\mathrm{FO}[\sim] = \mathrm{FO}$ without $<$ and succ

$$\exists x.bcast(x) \wedge \forall y.(y \not\sim x \Rightarrow \exists z.(z \sim y \wedge rcv(z)))$$

Some remarks

- No way to specify an order
- Can count letters on a given class up to some bound B
- Can count such classes up to some number

Roadmap

1. Establish normal form for $\mathrm{FO}[\sim]$
2. Translate to game formalism

FO[$\sim$]

- FO[$\sim$] = FO without $<$ and succ

$$\exists x.bcast(x) \wedge \forall y.(y \not\sim x \Rightarrow \exists z.(z \sim y \wedge rcv(z)))$$

Some remarks

- No way to specify an order
- Can count letters on a given class up to some bound B
- Can count such classes up to some number

Roadmap

1. Establish normal form for FO[$\sim$]
2. Translate to game formalism
3. Use games to prove results

Normal form

Normal form [FoSSaCS 20]

There is a bound $\mathrm{B} \in \mathbb{N}$ s.t. φ is equivalent to a disjunction of conjunctions of formulas of the form

$$\exists^{\bowtie m} y.(\theta(y) \wedge \psi_{\mathrm{B},\ell}(y))$$

Normal form

Normal form [FoSSaCS 20]

There is a bound $\mathrm{B} \in \mathbb{N}$ s.t. φ is equivalent to a disjunction of conjunctions of formulas of the form

$$\exists^{\bowtie m} y.(\theta(y) \wedge \psi_{\mathrm{B},\ell}(y))$$

$\equiv$ "There are $\bowtie$ m processes of type θ with local state ℓ."

Normal form

Normal form [FoSSaCS 20]

There is a bound $\mathrm{B} \in \mathbb{N}$ s.t. φ is equivalent to a disjunction of conjunctions of formulas of the form

$$\exists^{\bowtie m} y.(\theta(y) \wedge \psi_{\mathrm{B},\ell}(y))$$

$\equiv$ "There are $\bowtie m$ processes of type θ with local state ℓ."

Normal form

Normal form [FoSSaCS 20]

There is a bound $\mathrm{B} \in \mathbb{N}$ s.t. φ is equivalent to a disjunction of conjunctions of formulas of the form

$$\exists^{\bowtie m} y.(\theta(y) \wedge \psi_{\mathrm{B},\ell}(y))$$

$\equiv$ "There are $\bowtie$ m processes of type θ with local state ℓ."

- Local state of a process $\ell : A \to \{0, \ldots, \mathrm{B}\}$

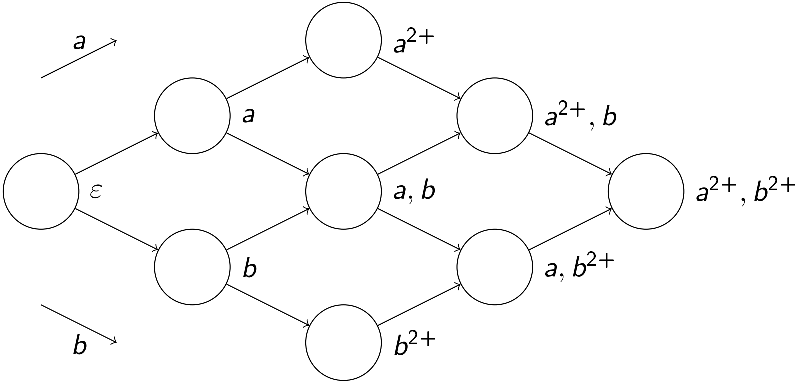
Parameterized Vector Games

Game framework for $\mathrm{FO}[\sim]$ formulas

$\mathcal{G} = (A, \mathrm{B}, \mathfrak{F})$ where $A = A_{sys} \uplus A_{env}$, $\mathrm{B} > 0$, and $\mathfrak{F}$ is the *acceptance condition*

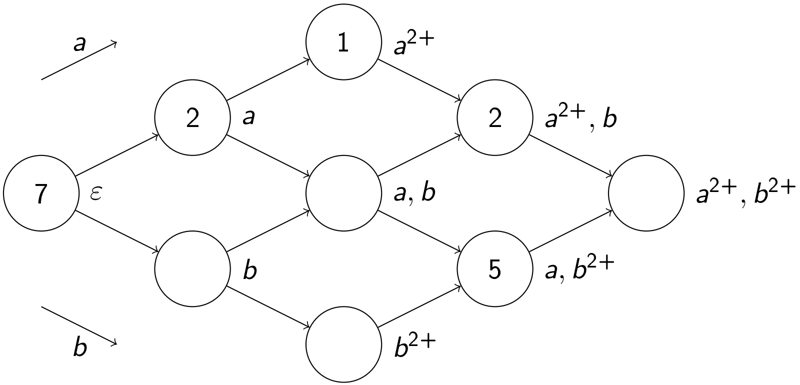
Parameterized Vector Games

Arena for $A_{sys} = \{a\}, A_{env} = \{b\}, \mathrm{B} = 2$: local states



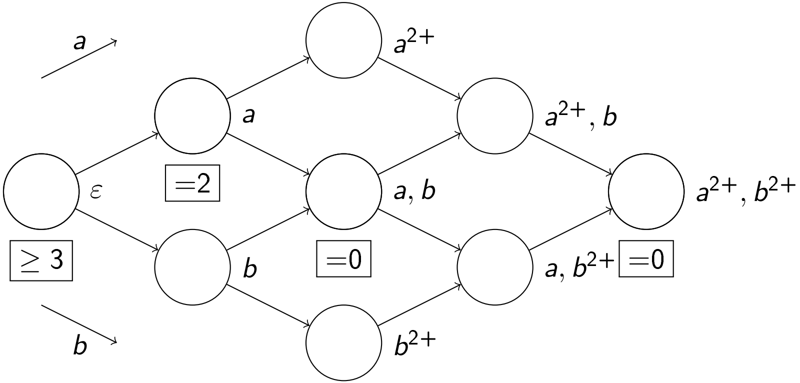
Parameterized Vector Games

Configuration c maps local states to number of tokens (default: 0)



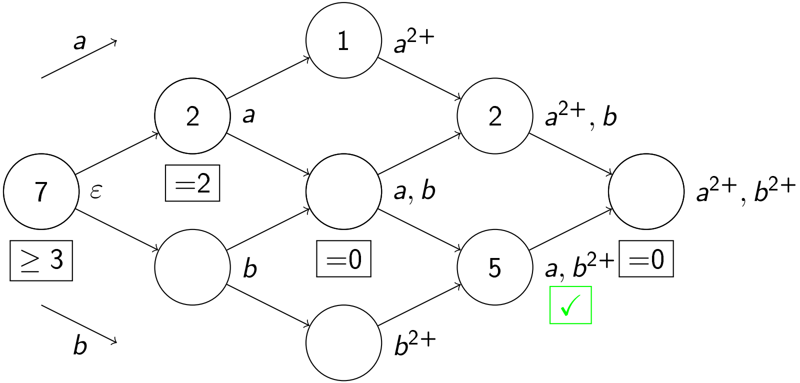
Parameterized Vector Games

Goal g = set of constraints for local states (default: ≥ 0)



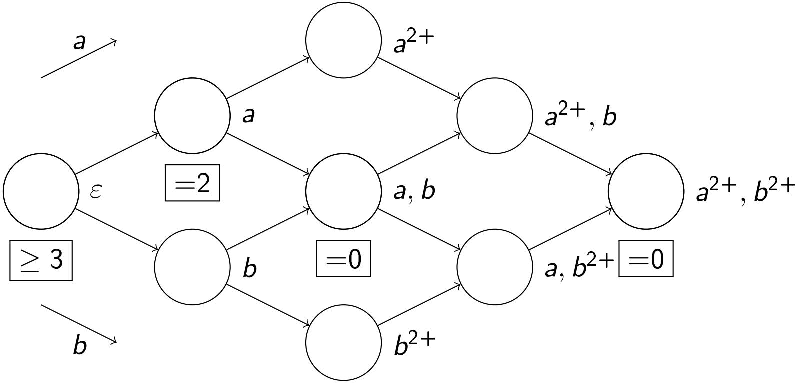
Parameterized Vector Games

Goal g = set of constraints for local states (default: ≥ 0)



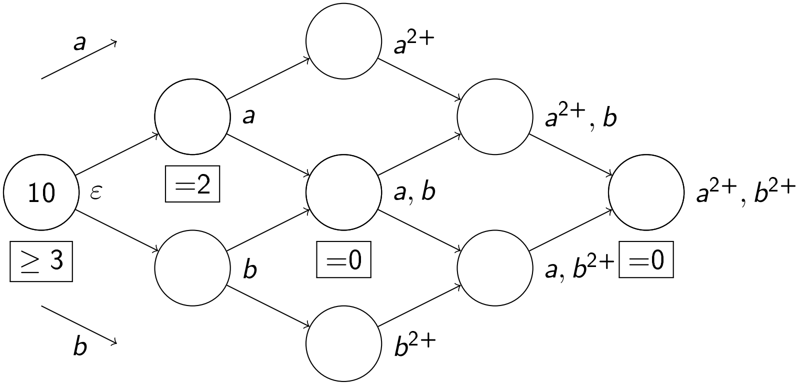
Parameterized Vector Games

Goal g = set of constraints for local states (default: ≥ 0)
Acceptance condition $\mathfrak{F}$ = disjunction of goals



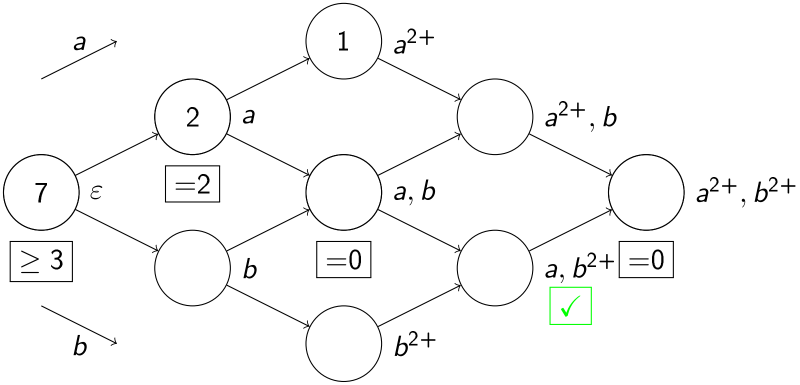
Parameterized Vector Games

Play on $\mathcal{G}$: System's turn



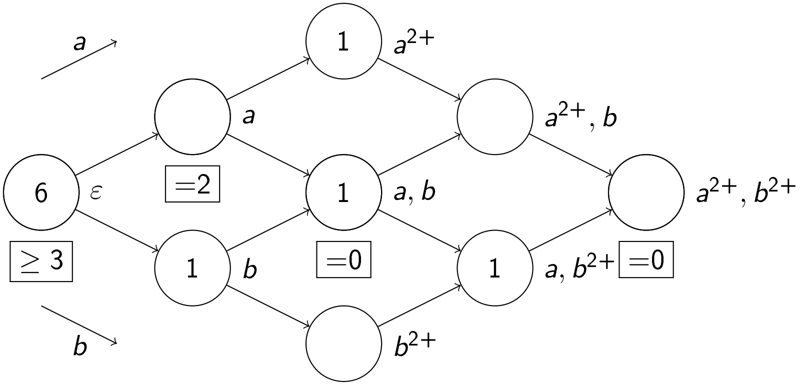
Parameterized Vector Games

Play on $\mathcal{G}$: Environment's turn



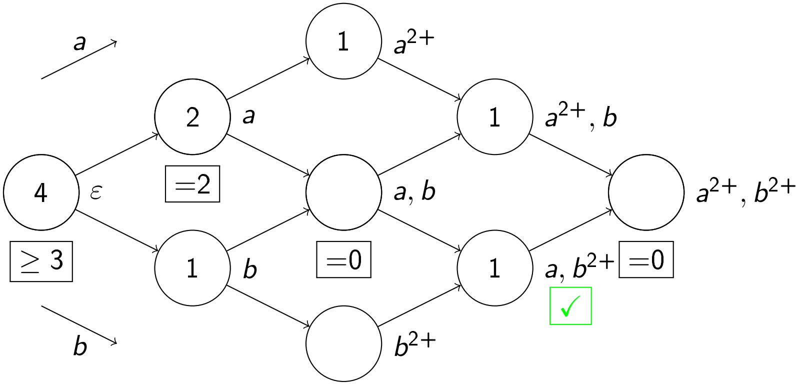
Parameterized Vector Games

Play on $\mathcal{G}$: System's turn



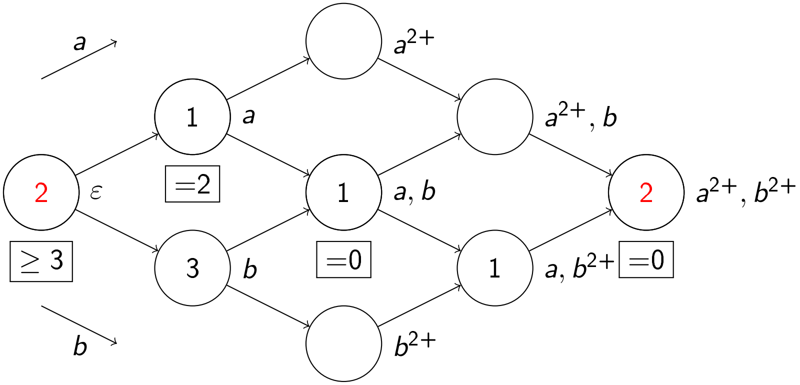
Parameterized Vector Games

Play on $\mathcal{G}$: Environment's turn



Parameterized Vector Games

Play on $\mathcal{G}$: System's turn



Equivalence with $\mathrm{FO}[\sim]$

- Asynchronous $\rightarrow$ turn-based game

No way to specify an order with $\mathrm{FO}[\sim]$

Equivalence with $\mathrm{FO}[\sim]$

- Asynchronous $\rightarrow$ turn-based game

No way to specify an order with $\mathrm{FO}[\sim]$

$(\mathit{req}, 1)(\mathit{req}, 2)(\mathit{ack}, 1)(\mathit{req}, 3)(\mathit{ack}, 2)(\mathit{ack}, 3)$

Equivalence with $\mathrm{FO}[\sim]$

- Asynchronous $\rightarrow$ turn-based game

No way to specify an order with $\mathrm{FO}[\sim]$

$$\begin{aligned} & (\mathit{req},1)(\mathit{req},2)(\mathit{ack},1)(\mathit{req},3)(\mathit{ack},2)(\mathit{ack},3) \\ \equiv\; & (\mathit{req},1)(\mathit{ack},1)(\mathit{req},2)(\mathit{ack},2)(\mathit{req},3)(\mathit{ack},3) \end{aligned}$$

Equivalence with FO[$\sim$]

- Asynchronous $\rightarrow$ turn-based game

No way to specify an order with FO[$\sim$]

$$(req,1)(req,2)(ack,1)(req,3)(ack,2)(ack,3)$$
$$\equiv (req,1)(ack,1)(req,2)(ack,2)(req,3)(ack,3)$$

- Normal form $\rightarrow$ Acceptance condition

There is a bound $\mathrm{B} \in \mathbb{N}$ s.t. φ is equivalent to a disjunction of conjunctions of formulas of the form

$$\exists^{\bowtie m} y.(\theta(y) \wedge \psi_{\mathrm{B},\ell}(y))$$

$\equiv$ "There are $\bowtie m$ processes of type θ with local state ℓ."

Results [FoSSaCS 20]

Undecidability

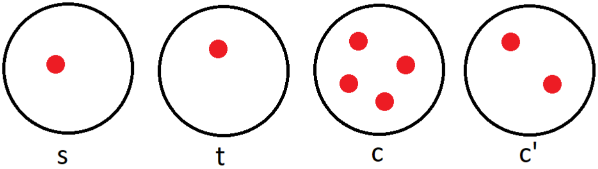
SYNTH$(\mathrm{FO}[\sim], (\{0\}, \{0\}, \mathbb{N}))$ is undecidable

Results [FoSSaCS 20]

Undecidability

SYNTH(FO[$\sim$], $(\{0\}, \{0\}, \mathbb{N})$) is undecidable

- Proof idea: encoding 2CM configuration



$$(s, c, c') \xrightarrow{t} \ldots$$

Results [FoSSaCS 20]

Positive result

SYNTH($\mathrm{FO}[\sim], (\mathbb{N}, \{k_{env}\}, \{k_{se}\})$) is decidable

Results [FoSSaCS 20]

Positive result

SYNTH($\mathrm{FO}[\sim], (\mathbb{N}, \{k_{env}\}, \{k_{se}\})$) is decidable

Cutoff

$\mathrm{k} = (k_{sys}, k_{env}, k_{se})$ is a cutoff wrt $(\mathcal{N}_{sys}, \mathcal{N}_{env}, \mathcal{N}_{se})$ for φ if either:

- for all $\mathrm{k}' \geq \mathrm{k}$, $\mathrm{k}' \in \mathit{Win}(\varphi)$
- for all $\mathrm{k}' \geq \mathrm{k}$, $\mathrm{k}' \notin \mathit{Win}(\varphi)$

Results [FoSSaCS 20]

Positive result

$\mathrm{SYNTH}(\mathrm{FO}[\sim], (\mathbb{N}, \{k_{env}\}, \{k_{se}\}))$ is decidable

Cutoff

$\mathrm{k} = (k_{sys}, k_{env}, k_{se})$ is a cutoff wrt $(\mathcal{N}_{sys}, \mathcal{N}_{env}, \mathcal{N}_{se})$ for φ if either:

- for all $\mathrm{k}' \geq \mathrm{k}$, $\mathrm{k}' \in \mathit{Win}(\varphi)$
- for all $\mathrm{k}' \geq \mathrm{k}$, $\mathrm{k}' \notin \mathit{Win}(\varphi)$

▶ Existence of cutoff ⇒ Synthesis decidable!

Conclusion

Summary

Synthesis for FO on data words is hard, but there are interesting decidable fragments.

Conclusion

Summary

Synthesis for FO on data words is hard, but there are interesting decidable fragments.

Future works

- Cases left open ($\mathrm{FO}^2[\sim]$, etc.)

Conclusion

Summary

Synthesis for FO on data words is hard, but there are interesting decidable fragments.

Future works

- Cases left open ($\mathrm{FO}^2[\sim]$, etc.)
- Synthesis without global view

Conclusion

Summary

Synthesis for FO on data words is hard, but there are interesting decidable fragments.

Future works

- Cases left open ($\mathrm{FO}^2[\sim]$, etc.)
- Synthesis without global view

▶ Thank you for your attention! ◀