Liveness analysis over automatic transition systems
(with applications to LTL model checking)

Anthony Widjaja To

LFCS, School of Informatics, University of Edinburgh

(Joint work with Leonid Libkin)

Introduction: verification

$$\begin{array}{ccc}
\text{Real system} & & \text{property} \\
\downarrow & & \downarrow \\
\text{Abstract Model} & \models & \varphi
\end{array}$$

- Abstract model: states + evolution rules
- Properties: safety, liveness, ...
- Classical approach:
  - Finite-state models
  - state-space exploration

Infinite-state models

Why consider these models?

Infinite-state models

Why consider these models?

- More convenient abstraction. Sources of infinity:
  - unbounded number of finite processes
  - unbounded stacks or FIFO queues
  - unbounded integer/real variables
  - unbounded discrete/continuous clocks
- State explosion problem: might help

Problem: undecidability

Solutions to undecidability

- Find *decidable subclasses*

Solutions to undecidability

- Find *decidable subclasses*
  - pushdown systems, prefix-recognizable systems
  - Petri nets
  - Timed systems
- Find good *semantic restriction*, e.g., well-structuredness.

Solutions to undecidability

- Find *decidable subclasses*
  - pushdown systems, prefix-recognizable systems
  - Petri nets
  - Timed systems
- Find good *semantic restriction*, e.g., well-structuredness.
- *Semi-algorithms* for general setting

Solutions to undecidability

- Find *decidable subclasses*
  - pushdown systems, prefix-recognizable systems
  - Petri nets
  - Timed systems
- Find good *semantic restriction*, e.g., well-structuredness.
- *Semi-algorithms* for general setting

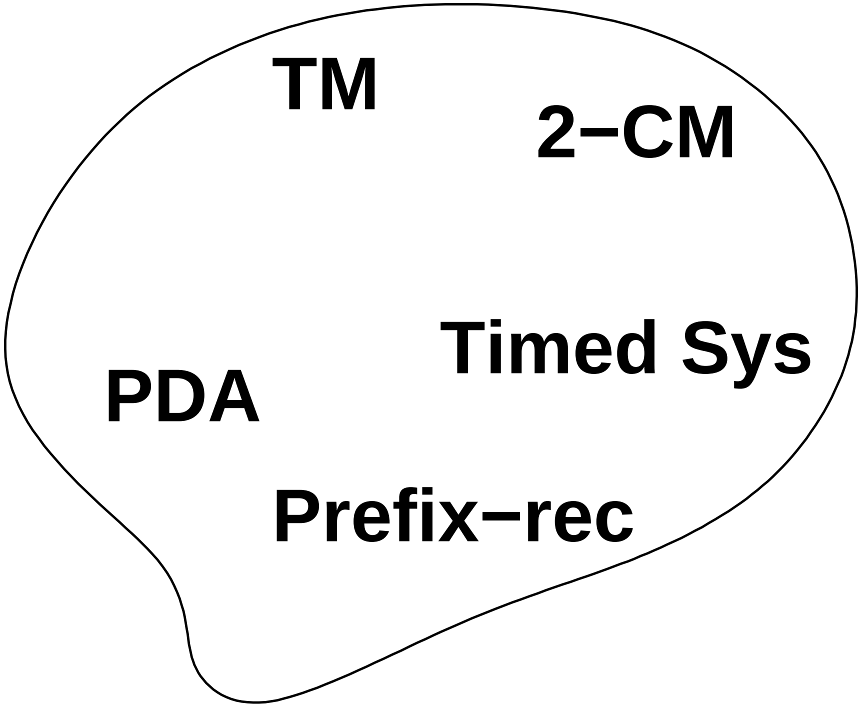
Warning: these directions are complementary and should not be competing against each other

Aim of our work

Verification: model $\models$ property?

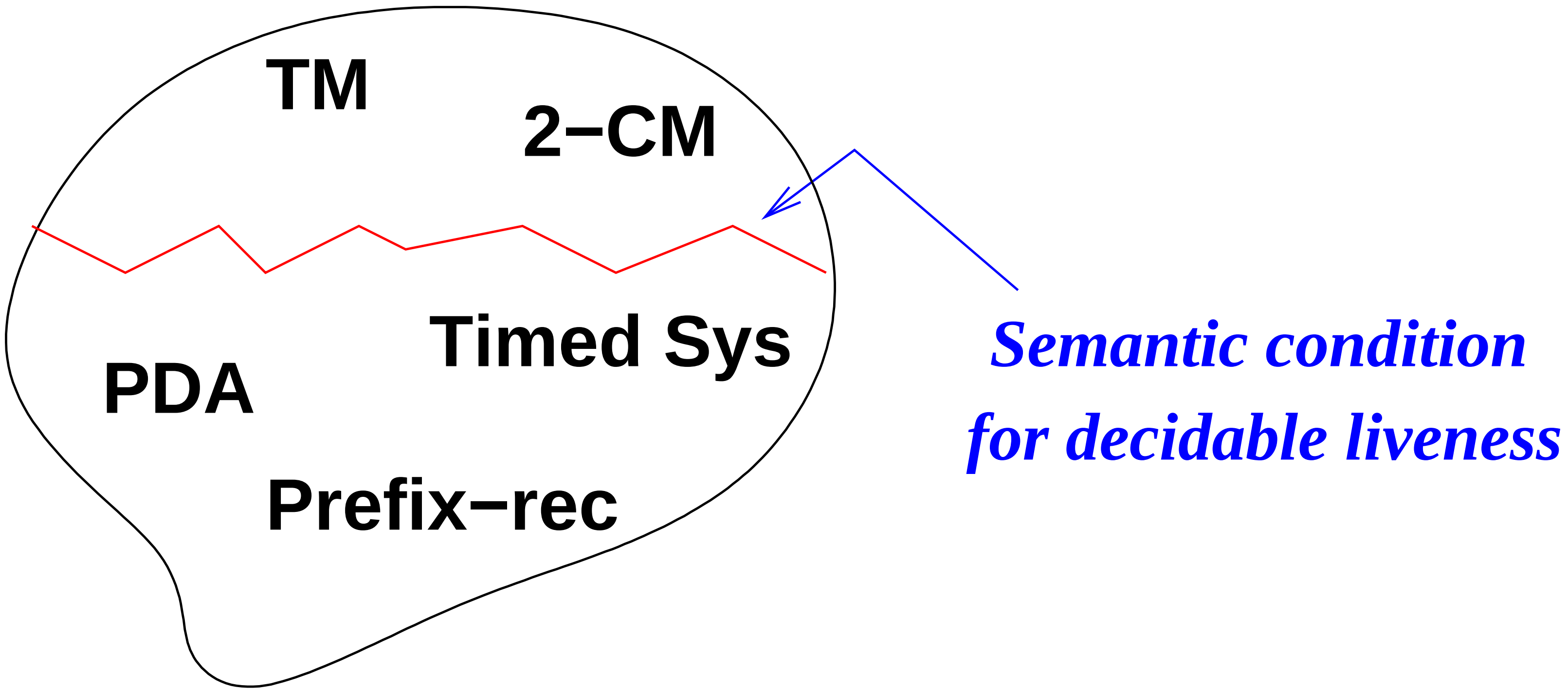
- Our model: general framework + semantic condition
- property: liveness, LTL-expressible

Aim of our work



Start with a general framework

Aim of our work

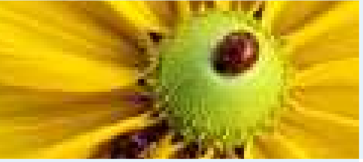


Start with a general framework

Overview of our results

Key points:

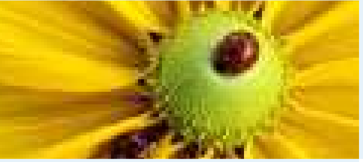
- A uniform explanation for decidable liveness for many classes of infinite systems.
- Applications to decidable LTL model checking.
- Seems to work reasonably well in practice.



Outline

- Background
  - Which properties?
  - A cursory glance of known approaches and results
- Our model
  - automatic transition systems
- Our results
  - Recurrent reachability
  - Applicaton to LTL model checking

Background



Transition systems (TSs)

Intuition: configurations + evolution rules

More formally: structures of the form

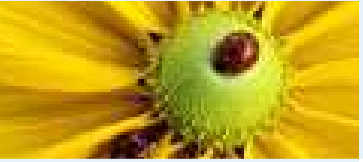
$$\mathcal{S} = \langle S, \{\longrightarrow_a\}_{a \in \Gamma} \rangle$$

where:

- S is a set of *configurations*,
- Γ is a set of *action labels*, and
- $\longrightarrow_a \subseteq S \times S$ is a transition relation labeled a.

Common properties

- Safety: no bad things will happen.
- Liveness: good things will eventually happen.
- More generally, LTL-expressible properties



Common properties

- Safety: no bad things will happen.
- Liveness: good things will eventually happen.
- More generally, LTL-expressible properties

We will instead study *recurrent reachability*

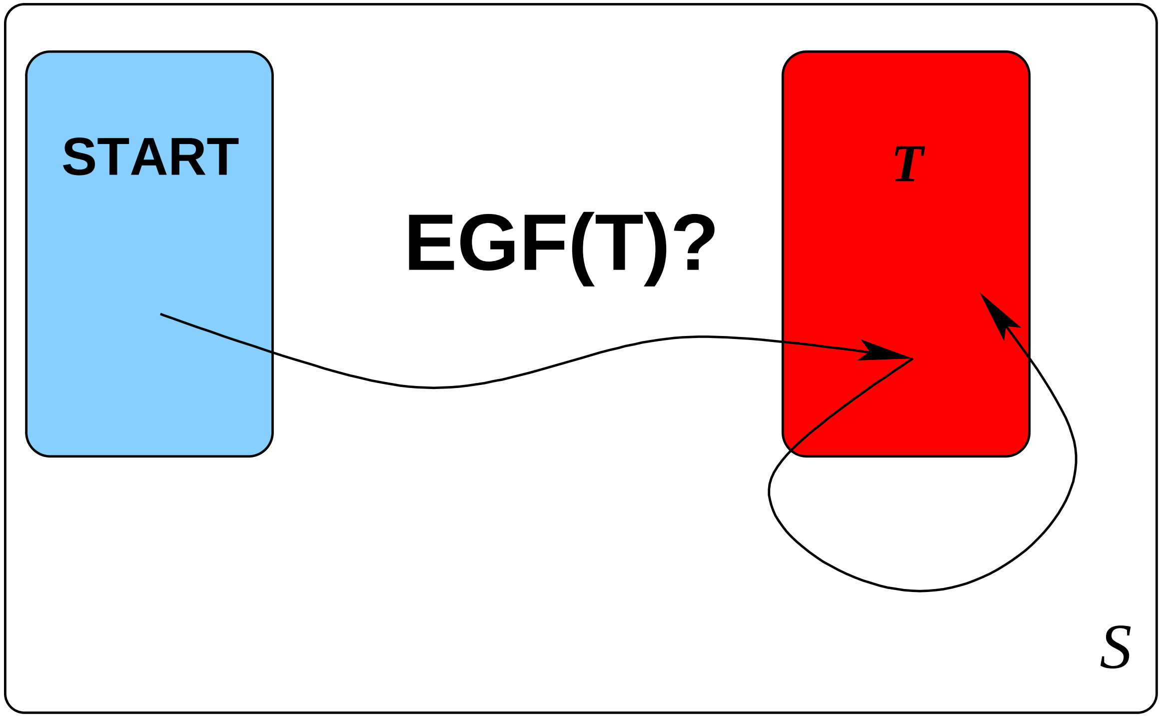
Common properties

- Safety: no bad things will happen.
- Liveness: good things will eventually happen.
- More generally, LTL-expressible properties

We will instead study *recurrent reachability*

- Intuitively: the acceptance condition of Büchi automata
- Liveness and LTL model checking can be reduced to it

Recurrent reachability



Does there exist an infinite path from START visiting T infinitely often?

Brief survey of known results

Recall two approaches:

- Decidable subclasses
- General frameworks

Brief survey of known results

Recall two approaches:

- Decidable subclasses
  - Pushdown systems
  - Prefix-recognizable systems
  - Petri nets
  - Reversal-bounded counter systems
  - Timed systems
  - ...
- General frameworks

Brief survey of known results

Recall two approaches:

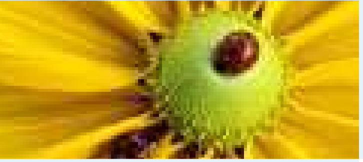
- Decidable subclasses
    - Pushdown systems
    - Prefix-recognizable systems
    - Petri nets
    - Reversal-bounded counter systems
    - Timed systems
    - ...
- General frameworks
    - Impose semantic restriction
    - Semi-algorithms

Brief survey of known results

Recall two approaches:

- Decidable subclasses
- General frameworks

I will survey a general framework called *regular model checking*



Brief survey: regular model checking (RMC)

- Use finite automata/transducers to generate TSs:
  - Configurations = words over Σ
  - Transitions = pairs of words over Σ
  - Automata represent sets of configurations
  - Transducers represent transition relations

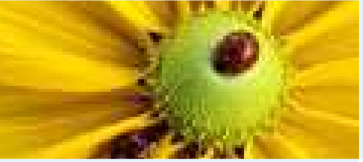
Brief survey: regular model checking (RMC)

- Use finite automata/transducers to generate TSs:
  - Configurations = words over Σ
  - Transitions = pairs of words over Σ
  - Automata represent sets of configurations
  - Transducers represent transition relations
- Many different variants depending on transducers' types

Brief survey: regular model checking (RMC)

- Use finite automata/transducers to generate TSs:
  - Configurations = words over Σ
  - Transitions = pairs of words over Σ
  - Automata represent sets of configurations
  - Transducers represent transition relations
- Many different variants depending on transducers' types

"Transducers" must determine if a given pair $(v, w) \in \Sigma^* \times \Sigma^*$ is a transition



Brief survey: regular model checking (RMC)

- Use finite automata/transducers to generate TSs:
  - Configurations = words over Σ
  - Transitions = pairs of words over Σ
  - Automata represent sets of configurations
  - Transducers represent transition relations

- Many different variants depending on transducers' types

"Transducers" must determine if a given pair $(v, w) \in \Sigma^* \times \Sigma^*$ is a transition

Next: popular transducers' type

Length-preserving transducers

- NFAs over $\Sigma \times \Sigma$

Length-preserving transducers

- NFAs over $\Sigma \times \Sigma$
- Given a pair $(v, w) \in \Sigma^* \times \Sigma^*$, how to check whether (v, w) is a transition?

Length-preserving transducers

- NFAs over $\Sigma \times \Sigma$
- Given a pair $(v, w) \in \Sigma^* \times \Sigma^*$, how to check whether (v, w) is a transition?
- Example: $v = aaabab$, $w = babbba$

Length-preserving transducers

- NFAs over $\Sigma \times \Sigma$
- Given a pair $(v, w) \in \Sigma^* \times \Sigma^*$, how to check whether (v, w) is a transition?
- Example: $v = aaabab$, $w = babbba$

$$v$$
$$w$$

Length-preserving transducers

- NFAs over $\Sigma \times \Sigma$
- Given a pair $(v, w) \in \Sigma^* \times \Sigma^*$, how to check whether (v, w) is a transition?
- Example: $v = aaabab$, $w = babbba$

$$\begin{array}{l} aaabab \\ babbba \end{array}$$

Length-preserving transducers

- NFAs over $\Sigma \times \Sigma$
- Given a pair $(v, w) \in \Sigma^* \times \Sigma^*$, how to check whether (v, w) is a transition?
- Example: $v = aaabab$, $w = babbba$

$$\begin{bmatrix} a \\ b \end{bmatrix}\begin{bmatrix} a \\ a \end{bmatrix}\begin{bmatrix} a \\ b \end{bmatrix}\begin{bmatrix} b \\ b \end{bmatrix}\begin{bmatrix} a \\ b \end{bmatrix}\begin{bmatrix} b \\ a \end{bmatrix}$$

Length-preserving transducers

- NFAs over $\Sigma \times \Sigma$
- Given a pair $(v, w) \in \Sigma^* \times \Sigma^*$, how to check whether (v, w) is a transition?
- Example: $v = aaabab$, $w = babbba$

$$\begin{bmatrix} a \\ b \end{bmatrix}\begin{bmatrix} a \\ a \end{bmatrix}\begin{bmatrix} a \\ b \end{bmatrix}\begin{bmatrix} b \\ b \end{bmatrix}\begin{bmatrix} a \\ b \end{bmatrix}\begin{bmatrix} b \\ a \end{bmatrix}$$

This is a word over $\Sigma \times \Sigma$

Length-preserving transducers: example

A *simple token-passing protocol* with 7 processes:

$$T \quad N \quad N \quad N \quad N \quad N \quad N$$

Length-preserving transducers: example

A *simple token-passing protocol* with 7 processes:

$$T \quad N \quad N \quad N \quad N \quad N \quad N$$

Length-preserving transducers: example

A *simple token-passing protocol* with 7 processes:

$$T \quad N \quad N \quad N \quad N \quad N \quad N$$

Length-preserving transducers: example

A *simple token-passing protocol* with 7 processes:

$$T \quad N \quad N \quad N \quad N \quad N \quad N$$

Length-preserving transducers: example

A *simple token-passing protocol* with 7 processes:

$$N \quad T \quad N \quad N \quad N \quad N \quad N$$

Length-preserving transducers: example

A *simple token-passing protocol* with 7 processes:

$$N \quad {\color{red}T} \quad N \quad N \quad N \quad N \quad N$$

Length-preserving transducers: example

A *simple token-passing protocol* with 7 processes:

$$N \quad T \quad N \quad N \quad N \quad N \quad N$$

Length-preserving transducers: example

A *simple token-passing protocol* with 7 processes:

$$N \quad T \quad N \quad N \quad N \quad N \quad N$$

Length-preserving transducers: example

A *simple token-passing protocol* with 7 processes:

$$N \quad N \quad T \quad N \quad N \quad N \quad N$$

Length-preserving transducers: example

A *simple token-passing protocol* with 7 processes:

$$N \quad N \quad T \quad N \quad N \quad N \quad N$$

Length-preserving transducers: example

A *simple token-passing protocol* with 7 processes:

$$N \quad N \quad N \quad T \quad N \quad N \quad N$$

A simple token-passing protocol

Description of the protocol for each $m \in \mathbb{N}$:

- m processes linearly ordered.
- Each process can either *hold a token* or *not*.
- At any given step, a process is chosen by the scheduler:
    1. If it holds a token, it can pass its token to its right process that *does not hold a token*
    2. It can remain *idle*
- Initially: only one token in the system

A simple token-passing protocol

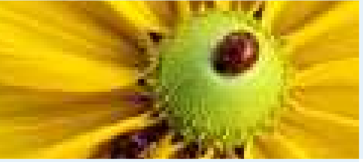
Description of the protocol for each $m \in \mathbb{N}$:

- m processes linearly ordered.
- Each process can either *hold a token* or *not*.
- At any given step, a process is chosen by the scheduler:
  1. If it holds a token, it can pass its token to its right process that *does not hold a token*
  2. It can remain *idle*
- Initially: only one token in the system

Verify for each $m \in \mathbb{N}$: system *cannot have more than one token*.

A simple token-passing protocol

How to model this in RMC framework?



A simple token-passing protocol

How to model this in RMC framework?

- Configurations: $\{N, T\}^*$
- Starting configurations: N^*TN^*
- Bad configurations: two or more tokens
  $(N+T)^*\textcolor{red}{T}(N+T)^*\textcolor{red}{T}(N+T)^*$

A simple token-passing protocol

How to model this in RMC framework?

- Configurations: $\{N, T\}^*$
- Starting configurations: N^*TN^*
- Bad configurations: two or more tokens
  $(N+T)^*T(N+T)^*T(N+T)^*$
- `Idle` transitions: $\left(\begin{bmatrix} N \\ N \end{bmatrix} + \begin{bmatrix} T \\ T \end{bmatrix}\right)^*$
- `Pass_token` transitions:
  $\left(\begin{bmatrix} N \\ N \end{bmatrix} + \begin{bmatrix} T \\ T \end{bmatrix}\right)^* \begin{bmatrix} T \\ N \end{bmatrix}\begin{bmatrix} N \\ T \end{bmatrix} \left(\begin{bmatrix} N \\ N \end{bmatrix} + \begin{bmatrix} T \\ T \end{bmatrix}\right)^*$

A simple token-passing protocol

How to model this in RMC framework?

- Configurations: $\{N, T\}^*$
- Starting configurations: N^*TN^*
- Bad configurations: two or more tokens
  $(N+T)^*T(N+T)^*T(N+T)^*$
- `Idle` transitions: $\left(\begin{bmatrix} N \\ N \end{bmatrix} + \begin{bmatrix} T \\ T \end{bmatrix}\right)^*$
- `Pass_token` transitions:
  $\left(\begin{bmatrix} N \\ N \end{bmatrix} + \begin{bmatrix} T \\ T \end{bmatrix}\right)^* \begin{bmatrix} T \\ N \end{bmatrix}\begin{bmatrix} N \\ T \end{bmatrix} \left(\begin{bmatrix} N \\ N \end{bmatrix} + \begin{bmatrix} T \\ T \end{bmatrix}\right)^*$
- Verify: START confs. cannot reach BAD confs.

Summary of this RMC variant

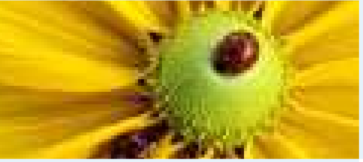
- Configurations = words over Σ
- Automata represent sets of configurations
- Transducers represent transition relations

Summary of this RMC variant

- Configurations = words over Σ
- Automata represent sets of configurations
- Transducers represent transition relations

Some facts:

- Safety, liveness, recurrent reachability are undecidable.
- Successful semi-algorithms for safety are available.



Summary of this RMC variant

- Configurations = words over Σ
- Automata represent sets of configurations
- Transducers represent transition relations

Some facts:

- Safety, liveness, recurrent reachability are undecidable.
- Successful semi-algorithms for safety are available.

Observe: each connected component is finite, i.e., *infinite paths must visit one state infinitely often*.

Our model

Automatic transition systems

- Use more general transducers
- Well-studied in automata community, but *not* in verification community.
- More suitable for modeling infinite systems, especially when liveness needs to be verified.

Note: Liveness for general infinite systems might have non-looping infinite witnessing paths

Synchronous transducers

- an NFA over $\Sigma_\perp \times \Sigma_\perp$, where $\Sigma_\perp = \Sigma \cup \{\perp\}$.
- Input: any pair of words $(v, w) \in \Sigma^* \times \Sigma^*$
- Example: $(aaabab, bab)$

Synchronous transducers

- an NFA over $\Sigma_\perp \times \Sigma_\perp$, where $\Sigma_\perp = \Sigma \cup \{\perp\}$.
- Input: any pair of words $(v, w) \in \Sigma^* \times \Sigma^*$
- Example: $(aaabab, bab)$

$$\begin{array}{cccccc} a & a & a & b & a & b \\ b & a & b & & & \end{array}$$

Synchronous transducers

- an NFA over $\Sigma_\perp \times \Sigma_\perp$, where $\Sigma_\perp = \Sigma \cup \{\perp\}$.
- Input: any pair of words $(v, w) \in \Sigma^* \times \Sigma^*$
- Example: $(aaabab, bab)$

$$\begin{array}{cccccc} a & a & a & b & a & b \\ b & a & b & \perp & \perp & \perp \end{array}$$

Synchronous transducers

- an NFA over $\Sigma_\perp \times \Sigma_\perp$, where $\Sigma_\perp = \Sigma \cup \{\perp\}$.
- Input: any pair of words $(v, w) \in \Sigma^* \times \Sigma^*$
- Example: $(aaabab, bab)$

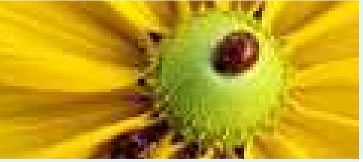
$$\begin{bmatrix} a \\ b \end{bmatrix}\begin{bmatrix} a \\ a \end{bmatrix}\begin{bmatrix} a \\ b \end{bmatrix}\begin{bmatrix} b \\ \perp \end{bmatrix}\begin{bmatrix} a \\ \perp \end{bmatrix}\begin{bmatrix} b \\ \perp \end{bmatrix}$$

Synchronous transducers

- an NFA over $\Sigma_\perp \times \Sigma_\perp$, where $\Sigma_\perp = \Sigma \cup \{\perp\}$.
- Input: any pair of words $(v, w) \in \Sigma^* \times \Sigma^*$
- Example: $(aaabab, bab)$

$$\begin{bmatrix} a \\ b \end{bmatrix}\begin{bmatrix} a \\ a \end{bmatrix}\begin{bmatrix} a \\ b \end{bmatrix}\begin{bmatrix} b \\ \perp \end{bmatrix}\begin{bmatrix} a \\ \perp \end{bmatrix}\begin{bmatrix} b \\ \perp \end{bmatrix}$$

This is a word over $\Sigma_\perp \times \Sigma_\perp$



Synchronous transducers

Intro
Outline
Background
Our model
Our results
Future work

- an NFA over $\Sigma_\perp \times \Sigma_\perp$, where $\Sigma_\perp = \Sigma \cup \{\perp\}$.
- Input: any pair of words $(v, w) \in \Sigma^* \times \Sigma^*$
- Example: $(aaabab, bab)$

$$\begin{bmatrix} a \\ b \end{bmatrix}\begin{bmatrix} a \\ a \end{bmatrix}\begin{bmatrix} a \\ b \end{bmatrix}\begin{bmatrix} b \\ \perp \end{bmatrix}\begin{bmatrix} a \\ \perp \end{bmatrix}\begin{bmatrix} b \\ \perp \end{bmatrix}$$

This is a word over $\Sigma_\perp \times \Sigma_\perp$

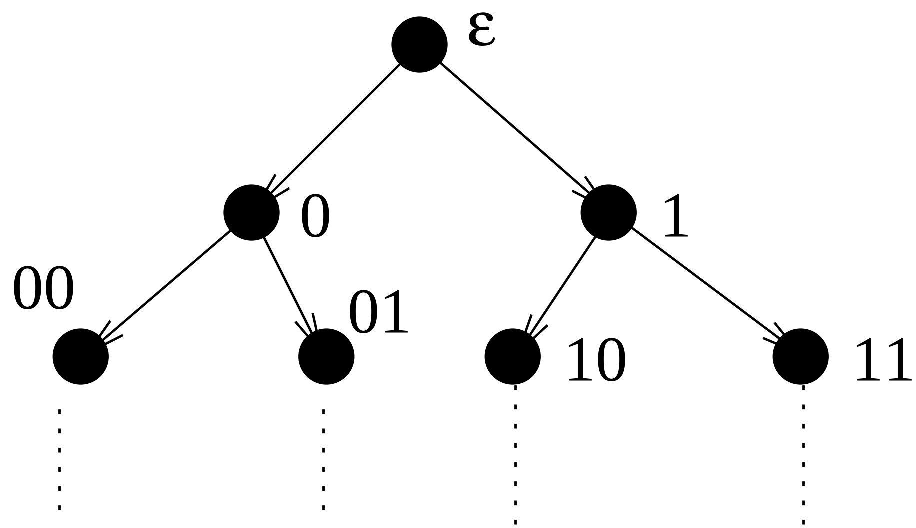
Conclusion: can recognize non-length preserving relations

Automatic transition systems: definition

$$\mathcal{S} = (S, \{\rightarrow_a\}_{a \in \Gamma})$$

- $S = \Sigma^*$ for some finite Σ
- $\rightarrow_a \subseteq \Sigma^* \times \Sigma^*$ is recognized by a synchronous transducer over Σ — called (*regular relation*)

A concrete example: infinite binary tree



$\mathfrak{T} = \langle \{0,1\}^*; \mathsf{succ}_0, \mathsf{succ}_1 \rangle$:

- $\mathsf{succ}_0 = \left(\begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right)^* \cdot \begin{bmatrix} \perp \\ 0 \end{bmatrix}$,
- $\mathsf{succ}_1 = \left(\begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right)^* \cdot \begin{bmatrix} \perp \\ 1 \end{bmatrix}$.

Note: $(\mathsf{succ}_0 \cup \mathsf{succ}_1)^*$ is also a regular relation.

More examples

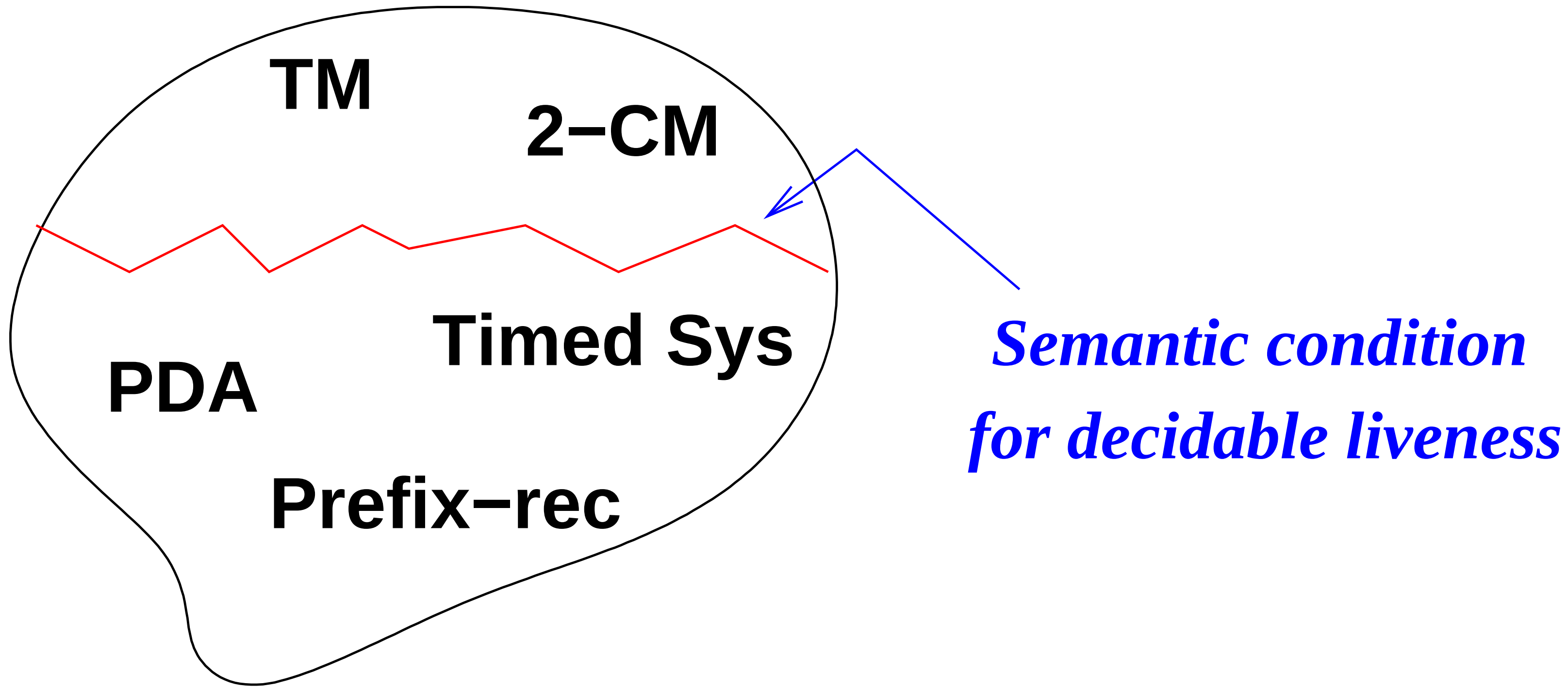
- Pushdown systems
- Prefix-recognizable systems
- Petri nets
- Turing machines
- Lossy channel systems
- Counter systems
- Discrete-time systems

Length-preserving vs. general transducers

- Safety checking: general case reducible to length-preserving case
- Not possible for liveness!!!
- Non-looping infinite paths exist in general
  - Sometimes uncountably many of them exist

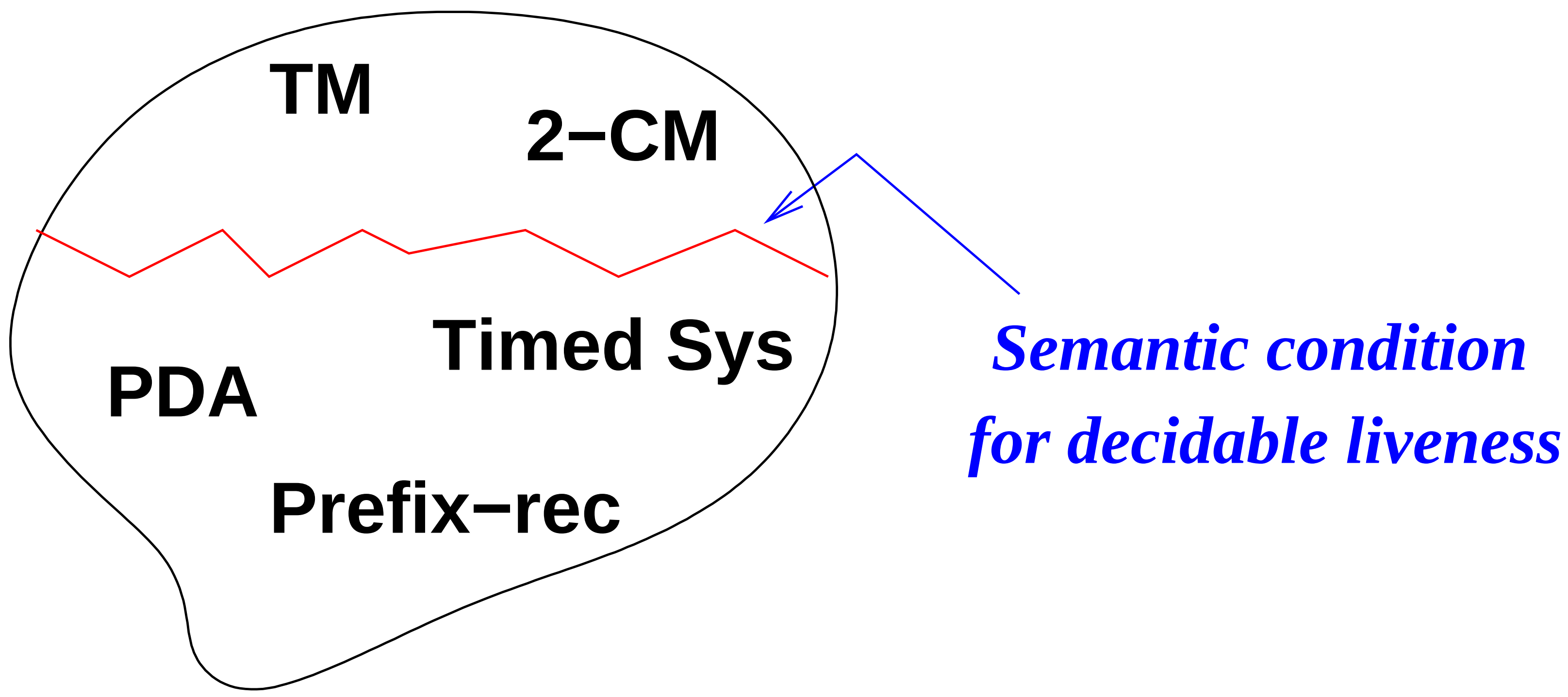
Our results

What we propose to do



The class of automatic transition systems

What we propose to do



The class of automatic transition systems

Remember: without further restrictions $\Longrightarrow$ undecidable

Our semantic condition

The transitive closure relation $\rightarrow^+ := (\bigcup_{a \in \Gamma} \rightarrow_a)^+$ is effectively regular (C1)

Convention: use $\mathcal{R}^+$ to denote the transducer for $\rightarrow^+$

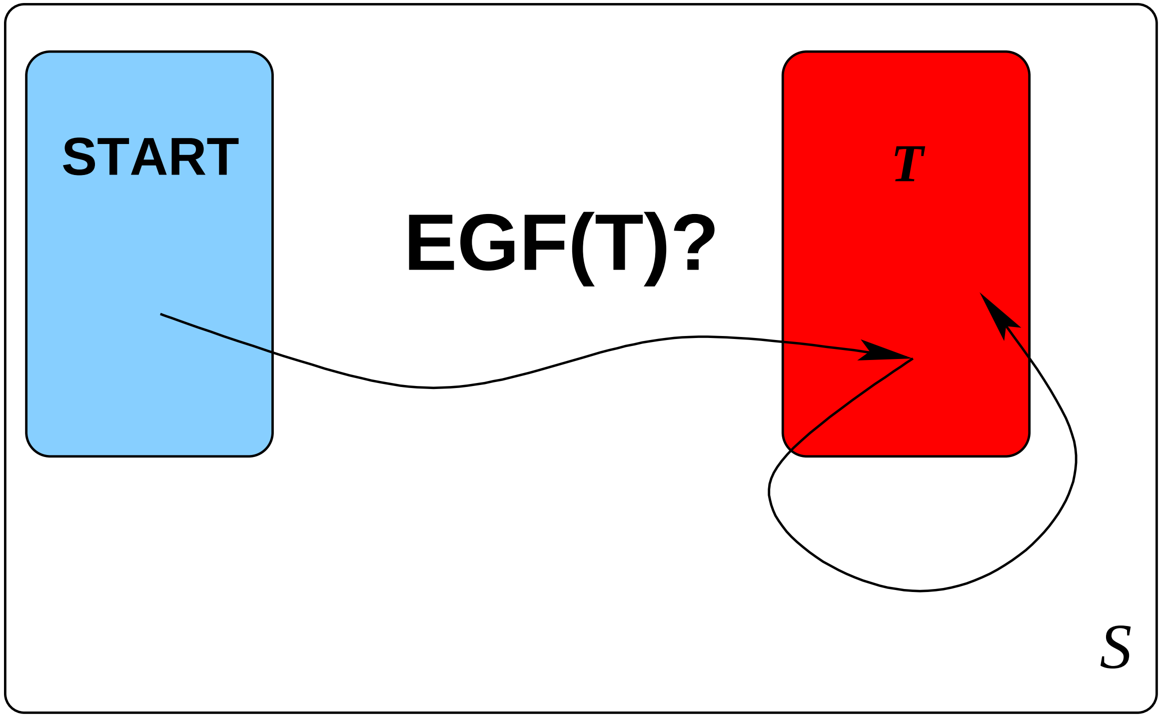
Our semantic condition

Why is this reasonable?

1. Satisfied by many subclasses of automatic TSs, e.g.,
   - pushdown systems
   - prefix-recognizable systems
   - reversal-bounded counter systems
   - discrete-time systems
   - communication-free nets (BPPs)

2. Semi-algorithms computing $\mathcal{R}^+$ exist for restricted classes of automatic transition systems, e.g., those which are Presburger-definable.

Recurrent reachability: more precisely



Does there exist an infinite path from a *regular set* START visiting a *regular set* T infinitely often?

Notation: $Rec(T) := [\![\mathbf{EGF}T]\!]$

Our main result

Theorem (LPAR'08): Over automatic systems satisfying C1: recurrent reachability is decidable in time $O(|\mathsf{START}| \times |T|^2 \times |\mathcal{R}^+|^3)$.

Our main result

Theorem (LPAR'08): Over automatic systems satisfying C1: recurrent reachability is decidable in time $O(|\text{START}| \times |T|^2 \times |\mathcal{R}^+|^3)$.

Furthermore:

- A "small" NFA for $Rec(T)$ can be efficiently constructed
- A "small" symbolic representation for a witnessing infinite path can be efficiently constructed

How to apply our results

Example 1:

Systems	$=$	subclass of aut. TSs satisfying (C1)
Property	$=$	Recurrent reachability

Example 2: (semi-algorithmic)

Systems	$=$	all aut. TSs
Property	$=$	Recurrent reachability

How to apply our results: LTL

Example 3:

Systems	=	subclass of aut. TSs satisfying (C1) AND closed under product with NFAs
Property	=	LTL-expressible

Example 4: (semi-algorithmic)

Systems	=	all aut. TSs
Property	=	LTL-expressible

How to apply our results (cont.)

Classes	Automatic	Regular $\longrightarrow^{+}$	Closure
Pushdown	Yes	Yes	Yes
Prefix-rec	Yes	Yes	Yes
D-time sys.	Yes	Yes	Yes
Rev-bc. sys.	Yes	Yes	Yes
Petri nets	Yes	**No**	Yes
BPPs	Yes	Yes	**No**
Turing mc.	Yes	**No**	Yes
Count sys.	Yes	**No**	Yes

Some more corollaries

Recurrent reachability:

- Pushdown systems: PTIME
- Prefix-recognizable systems: EXPTIME
- BPP: EXPTIME
- D-time rev-b. counter systems with one free counter: EXPTIME (double exponential in the number of clocks) — NEW

Some more corollaries

Recurrent reachability:

- Pushdown systems: PTIME
- Prefix-recognizable systems: EXPTIME
- BPP: EXPTIME
- D-time rev-b. counter systems with one free counter: EXPTIME (double exponential in the number of clocks) — NEW

LTL model checking:

- Pushdown systems: $2^{O(|\varphi| \times \log(|\mathcal{S}|)}$
- Prefix-rec: $2^{O(|\varphi| \times |\mathcal{S}|)}$
- D-time rev-b counter systems with one free counter: EXPTIME (double exponential in the number of clocks and $|\varphi|$) — NEW

Proof ideas for our theorem

Theorem (LPAR'08): Over automatic systems satisfying C1: recurrent reachability is decidable in time $O(|\text{START}| \times |T|^2 \times |\mathcal{R}^+|^3)$.

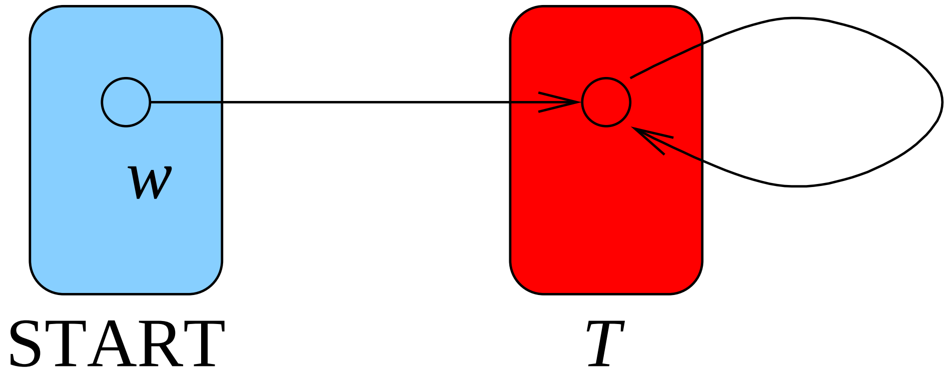
Proof Ideas: If $w \in Rec(T)$, it can have two kinds of witnessing paths:

- Looping (L): visits a configuration in T twice.
- Non-looping (NL): never visits a configuration in T twice.

Looping witnessing path (easy case)

- $Rec_1(T) := \{w \in Rec(T) :$ with (L)-witnessing path $\}$.

- Each $w \in Rec_1(T)$ has *lasso-shaped* witnessing path



- Since we have $\mathcal{R}^+$, an NFA for $Rec_1(T)$ is easy to construct.
    - Guess a word in T and check for reachability

Analyzing (NL)-infinite paths

- $Rec_2(T) := \{w \in Rec(T) :$ with (NL)-wit. path $\}$.
- Since we have $\mathcal{R}^+$, need only know initial point and the points in T

Analyzing (NL)-infinite paths

- $Rec_2(T) := \{w \in Rec(T) :$ with (NL)-wit. path $\}$.
- Since we have $\mathcal{R}^+$, need only know initial point and the points in T

$$
\begin{array}{cccccccc}
s_0 \longrightarrow & s_1 \longrightarrow & s_2 \longrightarrow \ldots \longrightarrow & s_{78} \longrightarrow & s_{79} \longrightarrow & s_{80} \longrightarrow & \ldots \\
 & \pitchfork & & \pitchfork & \pitchfork & & \ldots \\
 & T & & T & T & & \ldots
\end{array}
$$

Analyzing (NL)-infinite paths

- $Rec_2(T) := \{w \in Rec(T) :$ with (NL)-wit. path $\}$.
- Since we have $\mathcal{R}^+$, need only know initial point and the points in T

$$\begin{array}{ccccccccc} s_0 & \longrightarrow^+ & s_1 & \longrightarrow^+ & s_{78} & \longrightarrow^+ & s_{79} & \longrightarrow^+ & \ldots \\ & & \pitchfork & & \pitchfork & & \pitchfork & & \ldots \\ & & T & & T & & T & & \ldots \end{array}$$

Analyzing (NL)-infinite paths

- $Rec_2(T) := \{w \in Rec(T) :$ with (NL)-wit. path $\}$.
- Since we have $\mathcal{R}^+$, need only know initial point and the points in T

$$
\begin{array}{ccccccccc}
s_0 & \longrightarrow^+ & s_1 & \longrightarrow^+ & s_{78} & \longrightarrow^+ & s_{79} & \longrightarrow^+ & \dots \\
 & & \pitchfork & & \pitchfork & & \pitchfork & & \dots \\
 & & T & & T & & T & & \dots
\end{array}
$$

Note: every infinite subsequence of a witnessing sequence, which does not omit s_0, is still a witnessing sequence

Analzing (NL)-witnessing sequence

$aaabaab$

aab

$aaaaaaaaaaaaaaaabab$

$ababaaaa$

$ababaaababa$

$aaabaaa$

$aabaaabababababaabbbbbbbbbbbbbbbbbbbbbbbbbb$

$\ldots$

(NL)-witnessing sequence

Analzing (NL)-witnessing sequence

$aaabaab$

aab

$aaaaaaaaaaaaaaaaabab$

$ababaaaa$

$ababaaababa$

$aaabaaa$

$aabaaabababababaabbbbbbbbbbbbbbbbbbbbbbbbbb$

$\ldots$

Choose a *strictly increasing* subsequence

Analzing (NL)-witnessing sequence

In summary we have:

s_0	ε	ε	ε	$\dots$
$s_{1,1}$	$s_{1,2}$	ε	ε	$\dots$
$s_{2,1}$	$s_{2,2}$	$s_{2,3}$	ε	$\dots$
$s_{3,1}$	$s_{3,2}$	$s_{3,3}$	$s_{3,4}$	ε
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Look at first column (except for s_0):

s_0	ε	ε	ε	$\dots$
$s_{1,1}$	$s_{1,2}$	ε	ε	$\dots$
$s_{2,1}$	$s_{2,2}$	$s_{2,3}$	ε	$\dots$
$s_{3,1}$	$s_{3,2}$	$s_{3,3}$	$s_{3,4}$	ε
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Observation: There exists $\beta_0 \in \Sigma^*$ with $|\beta_0| = |s_0|$ and $\beta_0 = s_{j,1}$ for infinitely many $j \in \mathbb{Z}_{>0}$

Analzing (NL)-witnessing sequence

Choose subsequence s_0 followed by all these s_j's:

s_0	ε	ε	ε	$\ldots$
β_0	$s'_{1,2}$	ε	ε	$\ldots$
β_0	$s'_{2,2}$	$s'_{2,3}$	ε	$\ldots$
β_0	$s'_{3,2}$	$s'_{3,3}$	$s'_{3,4}$	ε
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

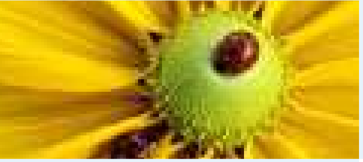
Observation: This is still an (NL)-witnessing sequence.

Analzing (NL)-witnessing sequence

Now disregard the first row and first column:

s_0	ε	ε	ε	$\dots$
β_0	$s'_{1,2}$	ε	ε	$\dots$
β_0	$s'_{2,2}$	$s'_{2,3}$	ε	$\dots$
β_0	$s'_{3,2}$	$s'_{3,3}$	$s'_{3,4}$	ε
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Observation: We can repeat the same procedure with $s'_{1,2}$ as the starting point.



Analzing (NL)-witnessing sequence

In summary, we obtain the following (NL)-seq:

s_0	ε	ε	ε	$\ldots$
β_0	α_1	ε	ε	$\ldots$
β_0	β_1	α_2	ε	$\ldots$
β_0	β_1	β_2	α_3	ε
$\vdots$	$\vdots$	$\vdots$	β_3 $\vdots$	$\ddots$

Analzing (NL)-witnessing sequence

In summary, we obtain the following (NL)-seq:

s_0	ε	ε	ε	$\dots$
β_0	α_1	ε	ε	$\dots$
β_0	β_1	α_2	ε	$\dots$
β_0	β_1	β_2	α_3	ε
$\vdots$	$\vdots$	$\vdots$	β_3 $\vdots$	$\ddots$

This (NL)-seq can be represented as:

$$\begin{bmatrix} s_0 \\ \beta_0 \end{bmatrix} \# \begin{bmatrix} \alpha_1 \\ \beta_1 \end{bmatrix} \# \begin{bmatrix} \alpha_2 \\ \beta_2 \end{bmatrix} \# \dots$$

Key: we need to construct a Büchi automaton recognizing such sequences.

Finishing the proof

Construct this Büchi automaton:

- Need to also "compress" the runs of T and $\mathcal{R}^+$
- Compressing runs of T: same as before
- Compressing runs of $\mathcal{R}^+$: use Ramsey theory

Construct the NFA for $Rec(T) := Rec_1(T) \cup Rec_2(T)$:

- Use the Büchi automaton
- Not-so-difficult automata constructions

Our main result (again)

Theorem (LPAR'08): Over automatic systems satisfying **C1**: recurrent reachability is decidable in time $O(|\mathsf{START}| \times |T|^2 \times |\mathcal{R}^+|^3)$.

Our main result (again)

Theorem (LPAR'08): Over automatic systems satisfying C1: recurrent reachability is decidable in time $O(|\mathsf{START}| \times |T|^2 \times |\mathcal{R}^+|^3)$.

Furthermore:

- "small" NFA for $Rec(T)$ can be efficiently constructed
- "small" symbolic representation for a witnessing infinite path can be efficiently constructed
- *Many applications and LTL*

Our main result (again)

Theorem (LPAR'08): Over automatic systems satisfying C1: recurrent reachability is decidable in time $O(|\text{START}| \times |T|^2 \times |\mathcal{R}^+|^3)$.

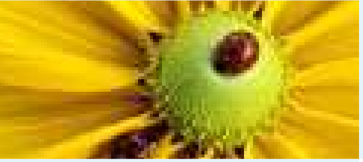
Furthermore:

- "small" NFA for $Rec(T)$ can be efficiently constructed
- "small" symbolic representation for a witnessing infinite path can be efficiently constructed
- *Many applications and LTL*

Advice for you: try our theorem first when you want to solve LTL model checking over infinite systems

Omitted results

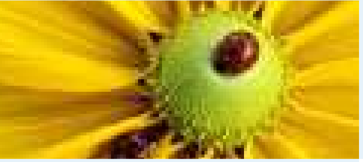
- Initial experimental results
  - fully-automatically verify freedom from (global) starvation for various cache coherence protocols
- Results for tree-automatic systems



Future work

Future work

- Experimental results
- Develop semi-algorithms for computing $\mathcal{R}^+$ for general automatic systems
- Find other subclasses of aut. TSs satisfying (C1)



Future work

- Experimental results
- Develop semi-algorithms for computing $\mathcal{R}^+$ for general automatic systems
- Find other subclasses of aut. TSs satisfying (C1)

THANK YOU!!