Tutorial

Graph Decompositions and the Verification of Concurrent Recursive Programs I

K Narayan Kumar

Chennai Mathematical Institute, India.

IMS, Singapore, August 2016
Outline

- Models
- Behaviours
- Specifications
- Verification via graph decompositions
- Conclusion
Concurrent Recursive Programs
Concurrent Recursive Programs

Variables range over finite domains
Concurrent Recursive Programs

Variables range over finite domains

Functions can be recursive
Concurrent Recursive Programs

Multi-threaded or Distributed

Functions can be recursive

Variables range over finite domains
func f1
{while <true>
{call f1 OR
a OR
exit;}
return;}

Recursive Programs are Pushdown Systems
Multi-threaded Programs are Multi-Pushdown Systems
Concurrent Communicating Programs

Queues to model communication channels
Unordered Channels

Bags to model unordered communication channels
System: Concurrent Processes with Data-Structures
System: Concurrent Processes with Data-Structures

- Processes

\[ p \quad r \quad q \]
System: Concurrent Processes with Data-Structures

- Processes
- Data structures
  - Stacks: recursive programs
System: Concurrent Processes with Data-Structures

- Processes
- Data structures
  - Stacks: recursive programs, multithreaded
System: Concurrent Processes with Data-Structures

- Processes
- Data structures
  - Stacks: recursive programs, multithreaded
  - Queues: communication (FIFO)
System: Concurrent Processes with Data-Structures

- Processes
- Data structures
  - Stacks: recursive programs, multithreaded
  - Queues: communication (FIFO)
  - Bags: communication (unordered)
System: Concurrent Processes with Data-Structures

- Processes
- Data structures
  - Stacks: recursive programs, multithreaded
  - Queues: communication (FIFO)
  - Bags: communication (unordered)
System: Concurrent Processes with Data-Structures

- Processes
- Data structures
  - Stacks: recursive programs, multithreaded
  - Queues: communication (FIFO)
  - Bags: communication (unordered)
System: Concurrent Processes with Data-Structures

- Processes
- Data structures
  - Stacks: recursive programs, multithreaded
  - Queues: communication (FIFO)
  - Bags: communication (unordered)
Special cases

- PDA: Pushdown automata
- Recursive programs
Special cases

- PDA: Pushdown automata
  Recursive programs
- MPDA: Multi-pushdown automata
  Multi-threaded recursive programs
Special cases

- PDA: Pushdown automata
  Recursive programs
- MPDA: Multi-pushdown automata
  Multi-threaded recursive programs
- MPA: Message passing automata
  Communicating finite state machines
Special cases

- PDA: Pushdown automata
  Recursive programs
- MPDA: Multi-pushdown automata
  Multi-threaded recursive programs
- MPA: Message passing automata
  Communicating finite state machines
- PN: Petri Nets: Only bags
Remote on-off via 2 channels
System: Architecture + Boolean Programs

c_1!b → c_1!a → on

c_2!b → c_2!a

off

c_2!a → c_2!b

c_1!a → c_1!b

c_1?b → c_1?a

c_3!b → c_3!a

p → q_1 → r

c_1

c_2

c_3

c_4

WiFi

Bluetooth
Operational semantics

- Transition system TS
- Configurations (infinite)
  - local states of processes
  - contents of data structures
- Transitions
- Induced by the boolean programs
Outline

* Models
  * Behaviours
* Specifications
* Verification via graph decompositions
* Conclusion
Linear Traces

\[(p, \text{on})(p, c_2!)(p, \text{off})(q_2, c_2?)(p, c_1!)(q_1, c_1?)\]
\[(q_2, c_4!)(p, \text{on})(p, c_2!)(p, \text{off})(r, c_4?)(r, \text{on})\]
\[(q_1, c_3!)(p, c_1!)(q_1, c_1?)(q_1, c_3!)(q_2, c_2?)(q_2, c_4!)(r, c_4?)(r, \text{on})(r, c_3?)(r, \text{off}) \cdots\]
Linear Traces

\[(p, \text{on})(p, \text{c}_2!)(p, \text{off})(q_2, \text{c}_2?)(p, \text{c}_1!)(q_1, \text{c}_1?)\]
\[(q_2, \text{c}_4!)(p, \text{on})(p, \text{c}_2!)(p, \text{off})(r, \text{c}_4?)(r, \text{on})\]
\[(q_1, \text{c}_3!)(p, \text{c}_1!)(q_1, \text{c}_1?)(q_1, \text{c}_3!)(q_2, \text{c}_2?)(q_2, \text{c}_4!)\]
\[(r, \text{c}_4?)(r, \text{on})(r, \text{c}_3?)(r, \text{off}) \ldots\]
Linear Traces

\[(p, \text{on})(p, c_2!)(p, \text{off})(q_2, c_2?)(p, c_1!)(q_1, c_1?)
(q_2, c_4!)(p, \text{on})(p, c_2!)(p, \text{off})(r, c_4?)(r, \text{on})
(q_1, c_3!)(p, c_1!)(q_1, c_1?)\]
\[(q_1, c_3!)(q_1, c_3!)(q_2, c_2?)(q_2, c_4!)
(r, c_4?)(r, \text{on})(r, c_3?)(r, \text{off}) \cdots \]

Is every message transmitted to the bulb?
Causal Response

\[(p, \text{on})(p, c_2!)(p, \text{off})(q_2, c_2?) (p, c_1!)(q_1, c_1?)
(q_2, c_4!)(p, \text{on})(p, c_2!)(p, \text{off})(r, c_4?) (r, \text{on})
(q_1, c_3!)(p, c_1!)(q_1, c_1?)(q_1, c_3!)(q_2, c_2?)(q_2, c_4!)
(r, c_4?)(r, \text{on})(r, c_3?)(r, \text{off}) \cdots \]
Causal Response

(p, on)(p, c₂!)(p, off)(q₂, c₂?)(p, c₁!)(q₁, c₁?)
(q₂, c₄!)(p, on)(p, c₂!)(p, off)(r, c₄?)(r, on)
(q₁, c₃!)(p, c₁!)(q₁, c₁?)(q₁, c₃!)(q₂, c₂?)(q₂, c₄!)
(r, c₄?)(r, on)(r, c₃?)(r, off)…

Does it obey the latest order?
Behaviours as Graphs

\[(p, \text{on})(p, c_2!)(p, \text{off})(q_2, c_2?)\]
\[(p, c_1!)(q_1, c_1?)\]
\[(q_2, c_4!)(p, \text{on})(p, c_2!)(p, \text{off})(r, c_4?)\]
\[(r, \text{on})\]
\[(q_1, c_3!)(p, c_1!)(q_1, c_1?)\]
\[(q_1, c_3!)(q_2, c_2?)(q_2, c_4!)(r, c_4?)\]
\[(r, \text{on})(r, c_3?)(r, \text{off})\]
Behaviours as Graphs

\[(p, \text{on})(p, c_2!)(p, \text{off})(q_2, c_2?)(p, c_1!)(q_1, c_1?)\]
\[(q_2, c_4!)(p, \text{on})(p, c_2!)(p, \text{off})(r, c_4?)(r, \text{on})\]
\[(q_1, c_3!)(p, c_1!)(q_1, c_1?)(q_1, c_3!)(q_2, c_2?)(q_2, c_4!)(r, c_4?)(r, \text{on})(r, c_3?)(r, \text{off}) \ldots\]
Behaviours as Graphs...

Does it obey the latest order?

\[(p, \text{on})(p, c_2!)(p, \text{off})(q_2, c_2?)\](q_1, c_1?)
\[(q_2, c_4!)(p, \text{on})(p, c_2!)(p, \text{off})(r, c_4?)(r, \text{on})\]
\[(q_1, c_3!)(p, c_1!)(q_1, c_1?)(q_1, c_3!)(q_2, c_2?)(q_2, c_4!)(r, c_4?)(r, \text{on})(r, c_3?)(r, \text{off}) \ldots\]
(p, on)(p, c_2!)(p, off)(q_2, c_2?)(p, c_1!)(q_1, c_1?)
(q_2, c_4!)(p, on)(p, c_2!)(p, off)(r, c_4?)(r, on)
(q_1, c_3!)(p, c_1!)(q_1, c_1?)(q_1, c_3!)(q_2, c_2?)(q_2, c_4!)
(r, c_4?)(r, on)(r, c_3?)(r, off) ...
Behaviours as Graphs...

$(p, \text{on})(p, c_2!)(p, \text{off})(q_2, c_2?)(p, c_1!)(q_1, c_1?)$

$(q_2, c_4!)(p, \text{on})(p, c_2!)(p, \text{off})(r, c_4?)(r, \text{on})$

$(q_1, c_3!)(p, c_1!)(q_1, c_1?)(q_1, c_3!)(q_2, c_2?)(q_2, c_4!)$

$(r, c_4?)(r, \text{on})(r, c_3?)(r, \text{off}) \ldots$

Does it obey the latest order?
Graphs for Sequential Systems

\[ q \]

a↑ba↑c↑aa↓↑b↓a↓cb↑↑a↓↓a↑b↑cb↓a↓↓ab↑b↑aa↑↑↓↓↓↓↓↓↓↓↓↓b
Letter before outermost call is the same as the letter after its return
Graphs for Sequential Systems

Letter before outermost call is the same as the letter after its return
Graphs for Sequential Systems

a↑ba↑c↑aa↓↑b↓a↓cb↑↑a↓↓a↑b↑cb↓a↓ab↑b↑aa↑↑↓↑↓↓↓↓↓↓b
Graphs for Sequential Systems

Nested Words
Alur, Madhusudan, 2009
Concurrent Behaviour with Matching (CBM)
Concurrent Behaviour with Matching (CBM)

A linear order (or path) for each process
* Edges labeled with data structures
Concurrent Behaviour with Matching (CBM)

* A linear order (or path) for each process
* Edges labeled with data structures
  * Communication edges form a matching
  * Edge labelled d relates the writer and reader of d
* Edges follow the discipline of the data structure
  * LIFO/FIFO/Bag
# Graphs vs Linear Traces

## Understanding Behaviors

<table>
<thead>
<tr>
<th>Linear Traces</th>
<th>Graphs (CBMs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Interactions are obfuscated and very difficult to recover.</td>
<td>• Interactions are visible</td>
</tr>
<tr>
<td>• Successor relation not meaningful</td>
<td>• no combinatorial explosion</td>
</tr>
<tr>
<td>• Combinatorial explosion</td>
<td></td>
</tr>
<tr>
<td>single distributed behavior results in a huge number of linear traces</td>
<td></td>
</tr>
</tbody>
</table>
Semantics of CPDS on CBMs
Semantics of CPDS on CBMs
Semantics of CPDS on CBMs
Semantics of CPDS on CBMs
Semantics of CPDS on CBMs
Semantics of CPDS on CBMs
Semantics of CPDS on CBMs
Semantics of CPDS on CBMs

Accepting states to turn them into language acceptors
Outline

* Models
* Behaviours
* Specifications
* Verification via graph decompositions
* Conclusion
Reachability
Is the bad state reachable?
Semantics of CPDS on CBMs
Semantics of CPDS on CBMs

Reachability reduces to Language Emptiness
Specification over CBMs

MSO: Monadic Second Order Logic

\[ \varphi ::= \text{false} \mid a(x) \mid p(x) \mid x \leq y \mid x \succ^d y \mid x \rightarrow y \]
\[ \mid x \in X \mid \varphi \lor \varphi \mid \neg \varphi \mid \exists x \varphi \mid \exists X \varphi \]
Specification over CBMs

MSO: Monadic Second Order Logic

\( \varphi ::= \text{false} \mid a(x) \mid p(x) \mid x \leq y \mid x \triangleright^d y \mid x \to y \)

\( \mid x \in X \mid \varphi \lor \varphi \mid \neg \varphi \mid \exists x \, \varphi \mid \exists X \, \varphi \)
Specification over CBMs

MSO: Monadic Second Order Logic

\[ \varphi ::= \text{false} \mid a(x) \mid p(x) \mid x \leq y \mid x \triangleright^d y \mid x \rightarrow y \mid x \in X \mid \varphi \lor \varphi \mid \neg \varphi \mid \exists x \varphi \mid \exists X \varphi \]
Specification over CBMs
Obey the latest order
Specification over CBMs
Obey the latest order
Specification over CBMs
Obey the latest order

∀z (r(z) ∧ on(z)) ⇒ ∃y (p(y) ∧ y < z)

∀x (x < z ∧ p(x) ⇒ x ≤ y)
∧ ∃x (x → y ∧ on(x)))

For p

For r
Based on the word successor relation, and the word total order

\[(p, \text{on})(p, c_2!)(p, \text{off})(q_2, c_2?)(p, c_1!)(q_1, c_1?)\]

\[(q_2, c_4!)(p, \text{on})(p, c_2!)(p, \text{off})(r, c_4?)(r, \text{on})\]

\[(q_1, c_3!)(p, c_1!)(q_1, c_1?)(q_1, c_3!)(q_2, c_2?)(q_2, c_4!)\]

\[(r, c_4?)(r, \text{on})(r, c_3?)(r, \text{off}) \cdots\]

* Based on the word successor relation, and the word total order

* LTL over words, MSO over words
Specification over Linear Traces

(p, on)(p, c_2!)(p, off)(q_2, c_2?)(p, c_1!)(q_1, c_1?)
(q_2, c_4!)(p, on)(p, c_2!)(p, off)(r, c_4?)(r, on)
(q_1, c_3!)(p, c_1!)(q_1, c_1?)(q_1, c_3!)(q_2, c_2?)(q_2, c_4!)
(r, c_4?)(r, on)(r, c_3?)(r, off) \cdots

* Based on the word successor relation, and the word total order

* LTL over words, MSO over words

Process successor can be recovered

Data edges cannot in general
Obey the latest order not expressible in MSO over Linear Traces

Process successor can be recovered
Data edges cannot in general

* Based on the word successor relation, and the word total order
* LTL over words, MSO over words

\[(p, \text{on})(p, c_2!)(p, \text{off})(q_2, c_2?)(p, c_1!)(q_2, c_4!)(p, \text{on})(p, c_2!)(p, \text{off})(q_1, c_1?)(q_1, c_3!)(q_2, c_2?)(q_2, c_4!)(p, \text{on})(p, c_2!)(p, \text{off})(q_1, c_1?)(q_1, c_3!)(q_2, c_2?)(q_2, c_3!)(p, c_1!)\]
Graphs for Sequential Systems

Relate outer most call and returns
Graphs for Sequential Systems

Relate outermost call and returns

∀x, y \left( a(x - 1) \land x \triangleright y \land 
\neg \exists z, z' (z \triangleright z' \land z < x < z') \right) \Rightarrow a(y + 1)
Graphs for Sequential Systems

Relate outer most call and returns

a↑ba↑c↑aa↓↑b↓a↓cb↑a↓a↑b↑cb↓a↓↓ab↑b↑aa↑↓↓↓↓↓↓b
Graphs for Sequential Systems

Relate outer most call and returns

Not expressible in MSO over Linear Traces
without nesting relation even with visible alphabet
## Graphs vs Linear Traces

### Expressiveness of Specifications

<table>
<thead>
<tr>
<th>Linear Traces</th>
<th>Graphs (CBMs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Too weak for many natural specifications</td>
<td></td>
</tr>
<tr>
<td>- Difficult to write/understand</td>
<td></td>
</tr>
<tr>
<td>- Requires syntactical or semantical restrictions to be meaningful</td>
<td></td>
</tr>
<tr>
<td>- Powerful specifications</td>
<td></td>
</tr>
<tr>
<td>- Easy to write/understand</td>
<td></td>
</tr>
<tr>
<td>- meaningful, interactions built-in</td>
<td></td>
</tr>
</tbody>
</table>
Outline

- Models
- Behaviours
- Specifications
- Verification via graph decompositions
- Conclusion
Verification problems

* Emptiness or Reachability for CPDS

* Inclusion or Universality for CPDS

* Satisfiability $\phi$: Is there a CBM that satisfies $\phi$?

* Model Checking: $S \models \phi$: Does every CBM accepted by $S$ satisfy $\phi$?

* Monadic second order logic

* Propositional dynamic logics

* Temporal logics
Reachability reduces to model checking
Reachability reduces to model checking

Model checking reduces to Reachability ...

... when specifications can be translated to systems

... this is not possible in general for graphs

\[ S \models \phi \]

\[ S \cap S_{\neg \phi} = \emptyset \]
Theorem:
From any CPDS $S$ we can construct a MSO formula $\phi_S$ such that a CBM satisfies $\phi_S$ iff it is accepted by $S$.

i.e. $\text{L}(S) = \text{L}(\phi_S)$

* Emptiness of CPDS reduces to satisfiability for MSO

$S$ accepts a CBM iff $\phi_S$ is satisfiable

* Model Checking reduces to satisfiability for MSO.

$S \models \phi$ iff $\neg \phi \land \phi_S$ is not satisfiable

* Similarly for language containment and universality
Verification problems

* Emptiness or Reachability for CPDS
* Inclusion or Universality for CPDS
* Satisfiability $\phi$: Is there a CBM that satisfies $\phi$?
* Model Checking: $S \models \phi$

* Temporal logics
* Propositional dynamic logics
* Monadic second order logic
Verification problems

* Emptiness or Reachability for CPDS
* Inclusion or Universality for CPDS
* Satisfiability $\phi$: Is there a CBM that satisfies $\phi$?
* Model Checking: $S \models \phi$

* Temporal logics
* Propositional dynamic logics
* Monadic second order logic

undecidable in general
Under-approximate Verification

Satisfiability problem:

C: class of behaviors

ϕ: Specification
Under-approximate Verification

Satisfiability problem:

C: class of behaviors

φ: Specification

Is φ satisfiable in C?
Under-approximate Verification

Emptiness or reachability problem:

C: class of behaviors

S: CPDS
Under-approximate Verification

Emptiness or reachability problem:

Is there an accepting run of $S$ on some behavior from $C$?

C: class of behaviors

S: CPDS
Under-approximate Verification

Model checking problem: $S \models_C \phi$

C: class of behaviors

$\phi$: Specification

S: CPDS
Under-approximate Verification

Model checking problem: $S \models^C \phi$

Do all behaviors from $C$ accepted by $S$ satisfy $\phi$?

- $C$: class of behaviors
- $\phi$: Specification
- $S$: CPDS
Under-approximate Verification

Mainly for reachability

* Emptiness or Reachability
* Inclusion or Universality
* Temporal logics: $S \models \phi$
  * Propositional dynamic logics
  * Monadic second order logic

* Bounded data structures
* Existentially bounded [Genest et al.]
* Acyclic Architectures [La Torre et al., Heußner et al. Clemente et al.]
* Bounded context switching [Qadeer, Rehof], [LaTorre et al.], ...
* Bounded phase [LaTorre et al.]
* Bounded scope [LaTorre et al.]
* Priority ordering [Atig et al., Saivasan et al.]
Under-approximate Verification

Mainly for reachability

- Emptiness or Reachability
- Inclusion or Universality

Satisfiability

\[ \mathcal{S} \models \varphi \]

Temporal logics

- Propositional dynamic logics
- Monadic second order logic

Bounded data structures

Existentially bounded [Genest et al.]

Acyclic Architectures [La Torre et al., Heußner et al. Clemente et al.]

Bounded context switching [Qadeer, Rehof], [LaTorre et al.], ...

Bounded phase [LaTorre et al.]

Bounded scope [LaTorre et al.]

Priority ordering [Atig et al., Saivasan et al.]

Reduction to MSO/Automata over trees.

undecidable
Several future directions are interesting. First, the class of multiple nested word languages with a bounded number of phases is of bounded tree-width (this is the property that allows us to embed them in trees). It would be interesting to characterize naturally the exact class of multiple nested words that have bounded tree-width. Secondly, we believe that our results have applications to other areas in verification, for instance in checking parallel programs that communicate with each other using unbounded FIFO queues, as multiple stacks can be used to simulate queues.
Outline

* Models
* Behaviours
* Specifications
* Verification via graph decompositions
* Conclusion
Under-approximate Verification

The Tree Width of Auxiliary Storage

P. Madhusudan
University of Illinois at Urbana-Champaign, USA
madhu@illinois.edu

Gennaro Parlato
LIAFA, CNRS and University of Paris Diderot, France.
gennaro@liafa.jussieu.fr

Abstract

We propose a generalization of results on the decidability of emptiness for several restricted classes of sequential and distributed automata with auxiliary storage (stacks, queues) that have recently been proved. Our generalization relies on reducing emptiness of these automata to finite-state graph automata (without storage) restricted to monadic second-order (MSO) definable graphs of bounded tree-width, where the graph structure encodes the mechanism to derive emptiness algorithms for automata, explaining and simplifying several existing results, as well as proving new decidability results.

However, the various identified decidable restrictions on the automata are, for the most part, awkward in their definitions e.g. emptiness of multi-stack pushdown automata where push to any stack is allowed at any time, but popping is restricted to the first non-empty stack is decidable! [8]. Yet, relaxing the definitions to more natural ones seems to either destroy decidability or their power. It is hence natural to ask: why do these automata have decidable emptiness problems? Is there a common underlying principle that explains their decidability?
Tree-width of Graphs

* A measure of the connectivity of a graph.
* How close is the graph to being a tree?
Tree-width of Graphs

* A measure of the connectivity of a graph.
* How close is the graph to being a tree?

A concept from Graph theory that is very useful computer science

St. Jude of CS
Tree-decompositions of Graphs

* A tree whose nodes are labeled by sets of vertices (bags)
* Every vertex appears in some bag.
* \((u,v)\) is an edge then there is a bag containing \(u,v\)
* For any \(u\) the bags containing \(u\) form connected part of the tree.
Tree-decompositions of Graphs

* A tree whose nodes are labeled by sets of vertices (bags)
* Every vertex appears in some bag.
* (u,v) is an edge then there is a bag containing u,v
* For any u the bags containing u form connected part of the tree.
Tree-decompositions of Graphs

* A tree whose nodes are labeled by sets of vertices (bags)
* Every vertex appears in some bag.
* \((u,v)\) is an edge then there is a bag containing \(u,v\)
* For any \(u\) the bags containing \(u\) form connected part of the tree.
Tree-decompositions of Graphs

* A tree whose nodes are labeled by sets of vertices (bags)
* Every vertex appears in some bag.
* (u,v) is an edge then there is a bag containing u,v
* For any u the bags containing u form connected part of the tree.
Tree-decompositions of Graphs

* A tree whose nodes are labeled by sets of vertices (bags)
* Every vertex appears in some bag.
* \((u,v)\) is an edge then there is a bag containing \(u,v\)
* For any \(u\) the bags containing \(u\) form connected part of the tree.

Size of the Decomposition = Size of largest bag - 1
Tree-decompositions of Graphs

A tree-decomposition of size 2
The smallest $n$ such that the graph has a decomposition of size $n$
The smallest $n$ such that the graph has a decomposition of size $n$.
A class of graphs has tree-width bounded by $k$ if every graph in the class has tree-width bounded by $k$. A class of graphs has bounded tree-width if it has tree-width bounded by some $k$. 
An Example: Nested Words

The class of nested words has tree-width bounded by 3
The class of nested words has tree-width bounded by 3.
An Example: Nested Words

The class of nested words has tree-width bounded by 3
The class of nested words has tree-width bounded by 3
An Example: Nested Words

The class of nested words has tree-width bounded by 3
An Example: Nested Words

The class of nested words has tree-width bounded by 3
The class of nested words has tree-width bounded by 3.
An Example: Nested Words

The class of nested words has tree-width bounded by 3

A bit more work gives a bound of 2
Example: Bounded Context Runs

* A CPDS with a single process and collection of Stacks
* A context is a segment where only one stack is accessed.
Example: Bounded Context Runs

* A CPDS with a single process and collection of Stacks
* A context is a segment where only one stack is accessed.

A behaviour with 5 contexts
Example: Bounded Context Runs

* A CPDS with a single process and collection of Stacks
* A context is a segment where only one stack is accessed.

What is the tree-width of the set of nested words with at most k contexts?
Tree-width of Bounded Context Runs
Tree-width of Bounded Context Runs
Tree-width of Bounded Context Runs
Tree-width of Bounded Context Runs
Tree-width of Bounded Context Runs

Add 1,3,5,8 to all other bags as well.
Tree-width of Bounded Context Runs

The set of nested words with at most k contexts has tree-width $\leq 2 + (k-1) = k+1$

Add 1,3,5,8 to all other bags as well.
- **Multi-stack pushdown automata with bounded context-switching:** This is the class of multi-stack automata where each computation of the automaton can be divided into $k$ stages, where in each stage the automaton touches only one stack (proved decidable first in [14]). We show that they can be simulated by graph automata on graphs of tree-width $O(k)$.

- **Multi-stack pushdown automata with bounded phases:** These are automata that generalize the bounded-context-switching ones: the computations must be dividable into $k$ phases, for a fixed $k$, where in each phase the automaton can push onto any stack, but can pop only from one stack (proved decidable recently in [11]). We show that graph automata on graphs of tree-width $O(2^k)$ can simulate them.

- **Ordered multi-stack pushdown automata:** The restriction here is that there a finite number of stacks that are ordered, and at any time, the automaton can push onto any stack, but pop only from the first non-empty stack. Note that the computation is not cut into phases, as in the above two restrictions. We show that automata on graphs of tree-width $O(n \cdot 2^n)$ (where $n$ is the number of stacks) can simulate them.

- **Distributed queue automata on polyforest architectures:** Distributed queue automata is a model where finite-state processes at $n$ sites work by communicating to each other using FIFO channels, modeled as queues. It was shown recently, that when the architecture is a polyforest (i.e. the underlying undirected graph is a forest), the emptiness problem is decidable (and for other architectures, it is undecidable) [12]. We
- Multi-stack pushdown automata with bounded context-switching: This is the class of multi-stack automata where each computation of the automaton can be divided into $k$ stages, where in each stage the automaton touches only one stack (proved decidable first in [14]). We show that they can be simulated by graph automata on graphs of tree-width $O(k)$.

- Multi-stack pushdown automata with bounded phases: These are automata that generalize the bounded-context-switching ones: the computations must be dividable into $k$ phases, for a fixed $k$, where in each phase the automaton can push onto any stack, but can pop only from one stack (proved decidable recently in [11]). We show that graph automata on graphs of tree-width $O(2^k)$ can simulate them.

- Ordered multi-stack pushdown automata: The restriction here is that there a finite number of stacks that are ordered, and at any time, the automaton can push onto any stack, but pop only from the first non-empty stack. Note that the computation is not cut into phases, as in the above two restrictions. We show that automata on graphs of tree-width $O(n \cdot 2^n)$ (where $n$ is the number of stacks) can simulate them.

- Distributed queue automata on polyforest architectures: Distributed queue automata is a model where finite-state processes at $n$ sites work by communicating to each other using FIFO channels, modeled as queues. It was shown recently, that when the architecture is a polyforest (i.e. the underlying graph is a forest), the emptiness problem is decidable (and for other architectures, it is)

Non-trivial arguments are necessary
Why Tree-width?

Theorem: (D. Seese)
1. If the MSO(2) satisfiability problem for a class C of graphs is decidable then C has bounded tree-width.
2. For any k, checking MSO(2) satisfiability among the class of graphs with tree-width at most k is decidable.
Why Tree-width?

Theorem: (D. Seese)
1. If the MSO(2) satisfiability problem for a class C of graphs is decidable then C has bounded tree-width.
2. For any k, checking MSO(2) satisfiability among the class of graphs with tree-width at most k is decidable.

Corollary:
If C is any MSO(2) definable family of graphs then, for any k, checking MSO(2) satisfiability among graphs in C with tree-width at most k is decidable.
Applying Seese’s Theorem

There is a formula $\Phi_{\text{CBM}}$ in MSO over graphs which describes the class of CBMs.

* Satisfiability of $\Phi$ over CBMs is equivalent to satisfiability of $\Phi \land \Phi_{\text{CBM}}$ over graphs.
Applying Seese’s Theorem

There is a formula $\Phi_{\text{CBM}}$ in MSO over graphs which describes the class of CBMs.

* Satisfiability of $\Phi$ over CBMs is equivalent to satisfiability of $\Phi \land \Phi_{\text{CBM}}$ over graphs.

The MSO theory of nested words is decidable.
Applying Seese’s Theorem

* Identify an MSO definable under-approximation class C of behaviours which guarantees a bound on the tree-width.
  
  eg. all behaviours with at most k context-switches

* MSO satisfiability w.r.t. C is decidable via Seese’s Theorem.
  
  eg. MSO satisfiability w.r.t k context-bounded CBMs

* Model-checking restricted to the class C is decidable via Seese’s Theorem.
  
  eg. model checking CPDS w.r.t. k context-bounded behaviours.
Applying Seese’s Theorem

* Identify an MSO definable under-approximation class $C$ of behaviours which guarantees a bound on the tree-width.
  
  eg. all behaviours with at most $k$ context-switches

* MSO satisfiability w.r.t. $C$ is decidable via Seese’s Theorem.
  
  eg. MSO satisfiability w.r.t $k$ context-bounded CBMs

* Model-checking restricted to the class $C$ is decidable via Seese’s Theorem.
  
  eg. model checking CPDS w.r.t. $k$ context-bounded behaviours.

Emptiness, universality, containment is decidable w.r.t. $k$ context-bounded behaviours.
Automata for better complexity

Theorem: (Madhusudan Parlato 2011)
Emptiness problem for graph automata over any MSO definable class of graphs of tree-width $\leq k$ is decidable in time exponential in $k$.

- The formula for the class is fixed and so plays no role in the complexity.
- Convert the graph automaton running on CBMS into a tree automaton running on their tree-decompositions.
Automata for better complexity

Theorem: (Madhusudan Parlato 2011)
Emptiness problem for graph automata over any MSO definable class of graphs of tree-width $\leq k$ is decidable in time exponential in $k$.

* The formula for the class is fixed and so plays no role in the complexity.
* Convert the graph automaton into a tree automaton running on their tree decompositions.

Yields the same complexity as handcrafted algorithms in almost all cases.
Graph Structure and Monadic Second-Order Logic
A Language-Theoretic Approach

BRUNO COURCELLE
Université de Bordeaux

JOOST ENGELFRIET
Universiteit Leiden
Other Measures: Desiderata

* MSO definable under-approximation classes of behaviours with bounded measure.
  
    eg. all behaviours with at most $k$ context-switches

* Decidable MSO satisfiability for such bounded classes via ?? Theorem.
  
    eg. MSO satisfiability w.r.t $k$ context-bounded CBMs

* Model-checking ...

With a translation to tree-automata to obtain efficient solutions.
Clique-width

* An algebraic measure for graphs.

* An algebra to construct graphs, each expression has a size.

* Size of a graph is the size of the smallest expression generating it.

* MSO decidability for graphs with bounded measure.

Translation to tree-automata: the expression trees can be used to interpret the graphs.
Co-graphs: An example

* Family of graphs generated by the following algebra:

\[ G ::= a \in \Sigma \mid G \oplus G \mid G \otimes G \]
Co-graphs: An example

* Family of graphs generated by the following algebra:

\[
G ::= a \in \Sigma | G \oplus G | G \otimes G
\]

Single vertex labelled a
Co-graphs: An example

* Family of graphs generated by the following algebra:

\[ G ::= a \in \Sigma \mid G \oplus G \mid G \otimes G \]

Single vertex labelled a
Co-graphs: An example

* Family of graphs generated by the following algebra:

\[ G ::= \quad a \in \Sigma \mid G \oplus G \mid G \otimes G \]

- Single vertex labelled a
- Disjoint union
Co-graphs: An example

* Family of graphs generated by the following algebra:

\[ G ::= a \in \Sigma \mid G \oplus G \mid G \otimes G \]

- Single vertex labelled a
- Disjoint union
Co-graphs: An example

* Family of graphs generated by the following algebra:

\[ G ::= \ a \in \Sigma \ | \ G \oplus G \ | \ G \otimes G \]

- Single vertex labelled a
- Disjoint union
- Disjoint union and connect all pairs
Co-graphs to Trees

Every co-graph has an expression generating it.
Co-graphs to Trees

Every co-graph has an expression generating it.

\[((b \oplus b) \otimes a) \otimes (b \otimes c)\]
Co-graphs to Trees

Every co-graph has an expression generating it.

\[((b \oplus b) \otimes a) \otimes (b \otimes c)\]

Vertices of the graph correspond to leaves of the tree.
Co-graphs to Trees

Every co-graph has an expression generating it.

Vertices of the graph correspond to leaves of the tree.

Edges are introduced by $\otimes$ nodes between leaves in its two subtrees.

Expression: $((b \oplus b) \otimes a) \otimes (b \otimes c)$
Every co-graph has an expression generating it.

$$((b \oplus b) \otimes a) \otimes (b \otimes c)$$

Vertices of the graph correspond to leaves of the tree.

Edges are introduced by $\otimes$ nodes between leaves in its two subtrees.
Co-graphs to Trees

Every co-graph has an expression generating it.

\[((b \oplus b) \otimes a) \otimes (b \otimes c)\]

Vertices of the graph correspond to leaves of the tree.

Edges are introduced by $\otimes$ nodes between leaves in its two subtrees.
### Interpretation on Trees

**Graph**

<table>
<thead>
<tr>
<th><strong>There is a vertex x</strong></th>
<th><strong>There is a leaf x</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>There is a set of vertices X</td>
<td>There is a set of leaves X</td>
</tr>
<tr>
<td>a(x)</td>
<td>a(x)</td>
</tr>
<tr>
<td>E(x,y)</td>
<td>There is path from x to y whose highest node is a $\otimes$</td>
</tr>
</tbody>
</table>

**Tree**

$((b \oplus b) \otimes a) \otimes (b \otimes c)$
Interpretation on Trees

There is a vertex $x$

There is a leaf $x$

There is a set of vertices $X$

There is a set of leaves $X$

$a(x)$

$a(x)$

$E(x,y)$

There is path from $x$ to $y$ whose highest node is $a \otimes$

$$((b \oplus b) \otimes a) \otimes (b \otimes c)$$
Interpretation on Trees

Graph | Tree
---|---
There is a vertex x | There is a leaf x
There is a set of vertices X | There is a set of leaves X
a(x) | a(x)
E(x,y) | There is path from x to y whose highest node is a ⊗

((b ⊕ b) ⊗ a) ⊗ (b ⊗ c)
Interpretation on Trees

There is a vertex $x$
There is a leaf $x$
There is a set of vertices $X$
There is a set of leaves $X$
a($x$)
a($x$)
$E(x,y)$

There is path from $x$ to $y$ whose highest node is $a \otimes$

\((b \oplus b) \otimes a) \otimes (b \otimes c)\)
Interpretation on Trees

<table>
<thead>
<tr>
<th>Graph</th>
<th>Tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>There is a vertex x</td>
<td>There is a leaf x</td>
</tr>
<tr>
<td>There is a set of vertices X</td>
<td>There is a set of leaves X</td>
</tr>
<tr>
<td>a(x)</td>
<td>a(x)</td>
</tr>
<tr>
<td>E(x,y)</td>
<td>There is path from x to y whose highest node is a ⊗</td>
</tr>
</tbody>
</table>

\[(b \oplus b) \otimes a \otimes (b \otimes c)\]
Interpretation on Trees

Graph

<table>
<thead>
<tr>
<th>There is a vertex x</th>
<th>There is a leaf x</th>
</tr>
</thead>
<tbody>
<tr>
<td>There is a set of vertices X</td>
<td>There is a set of leaves X</td>
</tr>
<tr>
<td>a(x)</td>
<td>a(x)</td>
</tr>
<tr>
<td>E(x,y)</td>
<td>There is path from x to y whose highest node is a ( \otimes )</td>
</tr>
</tbody>
</table>
Interpretation on Trees

There is a vertex $x$

There is a leaf $x$

There is a set of vertices $X$

There is a set of leaves $X$

There is a path from $x$ to $y$ whose highest node is $a$

We translate a formula $\Phi$ over graphs labelled with $\Sigma$ to a formula $\Phi_{tree}$ over trees labelled with $\Sigma \cup \{\oplus, \otimes\}$. 

<table>
<thead>
<tr>
<th>Graph</th>
<th>Tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>There is a vertex $x$</td>
<td>There is a leaf $x$</td>
</tr>
<tr>
<td>There is a set of vertices $X$</td>
<td>There is a set of leaves $X$</td>
</tr>
<tr>
<td>$a(x)$</td>
<td>$a(x)$</td>
</tr>
<tr>
<td>$E(x,y)$</td>
<td>There is path from $x$ to $y$ whose highest node is $a$ (\otimes)</td>
</tr>
</tbody>
</table>
**MSO decidability for Co-graphs**

* The collection of trees labelled by $\Sigma \cup \{\oplus, \otimes\}$ that constitute valid co-graph expressions is a regular tree language.
  
  Expressible in MSO over trees ($\varphi_{\text{co}}$)

A formula $\Phi$ in MSO over graphs is satisfiable over co-graphs iff

the formula $\Phi_{\text{tree}} \land \varphi_{\text{co}}$ is satisfiable over trees.

The MSO theory of co-graphs is decidable.