Verification of Concurrent Recursive Programs
Verification of Concurrent Recursive Programs

(an unified view via split-width)

From her forthcoming PhD Thesis
Concurrent Recursive Programs
Concurrent Recursive Programs

Variables range over finite domains
Recursive Programs

Functions can be recursive

Variables range over finite domains
Concurrent Recursive Programs

- Functions can be recursive
- Variables range over finite domains
- Multi-threaded (shared state) or Distributed Systems
Recursive program = Pushdown system

```plaintext
func f1
{while <true>
{call f1 OR
 a OR
 exit;}
 return;}
```
Multi-threaded program = Multi-PDS

```
func f1
{while <true>
{call f1 OR
 a OR
 exit;}
 return;}

func f2
{while <true>
{call f2 OR
 a OR
 exit;}
 return;}

func f3
{while <true>
{call f3 OR
 a OR
 exit;}
 return;}
```
Communicating FSMs

Diagram showing processes and queues.
Communicating FSMs

Process 1

Queue 1

Queue 2

Process 2

Queue 3

Queue 4

Queue 5

Queue 6

Process 3

Monday 29 July 13
Communicating Recursive Processes
Communicating Recursive Processes
MPDSs, CFSMs, CRPs are all Turing Powerful.
The Verification Problem

MPDSs, CFSMs, CRPs are all Turing Powerful.

• **MPDSs** -- Restrictions on the stack access
  - Bounded Context
  - Bounded Phase
  - Bounded Scope
  - Ordered Stacks

Qadeer & Rehof,
LaTorre & Madhusudan & Parlato
LaTorre & Napoli
Atig & Bollig & Habermehl
The Verification Problem

MPDSs, CFSMs, CRPs are all Turing Powerful.
The Verification Problem

MPDSs, CFSMs, CRPs are all Turing Powerful.

- CFMs
  - Universally/Existentially bounded systems
    - Henrikson et al., Genest & Kuske & Muscholl
- Message Sequence Graphs (or HMSCs)
  - Madhusudan
The Verification Problem

MPDSs, CFSMs, CRPs are all Turing Powerful.
The Verification Problem

MPDSs, CFSMs, CRPs are all Turing Powerful.

- CRPs
  - Well-queueing Systems with context bounds,

  Heussner&Leroux,&Muscholl&Sutre, LaTorre&Madhusudan&Parlato
func f1
{while <true>
{call f1 OR
 a OR
 exit;}
 return;}

Nested word
= word + binary nesting relation

Behaviours as Graphs
Multiply Nested word (MNW)

= word + multiple nesting relations
Behaviours as Graphs

Process 1

- send a OR
- send b OR
- skip

Queue 1

- b
- b
- a

Process 2

- rec a OR
- rec b OR
- skip

Message Sequence Charts
Behaviours as Graphs

Message Sequence Charts with Nesting
And other beasts ...

 MSCNs enjoy a natural graphical representation.

Example 4. An MSCN over Architecture is shown in Figure 1.3. Each process is represented by a vertical line. The relation \( \text{proc} \) orders (top-down) consecutive events located on the same process line. The messages (\( \text{msg} \)) are depicted by straight edges (solid and dotted) connecting different lines and nesting edges (\( \text{nst} \)) are depicted by curved edges (solid and dotted) connecting within a line. More specifically, \( \text{1} \text{nst} \) is depicted by solid curved edges on process 1, \( \text{2} \text{nst} \) by dotted curved edges on process 1 and \( \text{3} \text{nst} \) is depicted by solid curved edges on process 3. \( \text{1} \text{msg} \) is depicted by solid straight edges from process 1 to process 2, \( \text{2} \text{msg} \) by dotted straight edges from process 1 to process 2, \( \text{3} \text{msg} \) by solid straight edges from process 2 to process 1, \( \text{4} \text{msg} \) by dotted straight edges from process 2 to process 1, \( \text{5} \text{msg} \) by solid straight edges from process 2 to process 3, \( \text{6} \text{msg} \) by solid straight edges from process 3 to process 2, and \( \text{7} \text{msg} \) is depicted by dotted straight edges from process 3 to process 2. The action labels, message labels and the labels on nesting edges are not shown as it is unary.

Figure 1.3: An MSCN over Architecture
Graphs and MSO

Our graphs are
Our graphs are

- A finite number of linear orders ($<_p$)
- One or more nesting relation per linear order
  Corresponding to the stacks ($<_s$)
- Message relations between processes
  One per queue, assumed to be FIFO ($<_{pq}$)
Graphs and MSO

Our graphs are

- A finite number of linear orders ($<_p$)
- One or more nesting relation per linear order
  Corresponding to the stacks ($<_s$)
- Message relations between processes
  One per queue, assumed to be FIFO ($<_{pq}$)
Graphs and MSO

Our graphs are

• A finite number of linear orders ($<_p$)
• One or more nesting relation per linear order
  Corresponding to the stacks ($<_s$)
• Message relations between processes
  One per queue, assumed to be FIFO ($<_{pq}$)
Our graphs are

- A finite number of linear orders ($\prec_p$)
- One or more nesting relation per linear order
  Corresponding to the stacks ($\prec_s$)
- Message relations between processes
  One per queue, assumed to be FIFO ($\prec_{pq}$)

MSO has one binary relation symbol for each of these relations.
Graphs and MSO

Our graphs are

• A finite number of linear orders (\(\prec_p\))

• One or more nesting relation per linear order
  Corresponding to the stacks (\(\prec_s\))

• Message relations between processes
  One per queue, assumed to be FIFO (\(\prec_{pq}\))

MSO has one binary relation symbol for each of these relations.

Satisfiability is undecidable with 2 nesting relations / 2 processes connected by queues
Tree-width

Madhusudan/Parlato show that
Madhusudan/Parlato show that

- Runs of the restricted systems have bounded tree-width
- For any system, its set of restricted runs is MSO definable.
Split behaviors
Split behaviors

Size of the split = number of components = 4
Split behaviors
Split behaviors

Size of the split = number of components = 4
An algebra on Split behaviours

Basic splits:

Operations:

merge (binary)
shuffle (unary)
The **merge** Operation
The merge Operation
The merge Operation
The merge Operation
The merge Operation
The merge Operation
The **merge** Operation
The **merge** Operation
The **merge** Operation
The **merge** Operation
The Shuffle Operation
The Shuffle Operation
The Shuffle Operation
The Shuffle Operation
The Shuffle Operation
Shuffle Operation
Shuffle Operation
Shuffle Operation

Invalid!
Shuffle Operation
Shuffle Operation
Shuffle Operation
Shuffle Operation
Any behaviour can be generated by the algebra
Any behaviour can be generated by the algebra
Any behaviour can be generated by the algebra
Any behaviour can be generated by the algebra
Any behaviour can be generated by the algebra
Bounded split-width ($k$)

If a split-behaviour can be generated by the algebra, with the size of all the splits used $\leq k$
Example: an MSCN
Example: an MSCN
Example: an MSCN
Example: an MSCN
Example: an MSCN
Example: an MSCN
Model-Checking w.r.t Split-width k

Given a concurrent recursive program

Given two concurrent recursive programs
Model-Checking w.r.t Split-width k

Given a concurrent recursive program

• Is there an accepting run with split-width <= k?
• Does it accept all split-width <= k words?

Given two concurrent recursive programs

• Are the split-width k-behaviours of one contained in those of the other?
Model-Checking w.r.t Split-width k

Given a concurrent recursive program

- Is there an accepting run with split-width \( \leq k \)?
- Does it accept all split-width \( \leq k \) words?

Given two concurrent recursive programs

- Are the split-width \( k \)-behaviours of one contained in those of the other?

Abstract Derivation Trees
Split-width $\leq k$ Runs
Split-width \( \leq k \) Runs

ADTs representing split-width \( k \) derivation trees form a regular tree language.

Easy tree automaton construction
Split-width $\leq k$ Runs

ADTs representing split-width $k$ derivation trees form a regular tree language.

Easy tree automaton construction

ADTs representing derivation trees of split-width $k$ accepting runs of a CRP is a regular tree language.

Easy tree automaton construction. Size of the automaton is exponential in $k$. 
Decidability of Model-checking

**Input**

S : CRP over a given set of processes.
k : parameter (split-width)

<table>
<thead>
<tr>
<th></th>
<th>ExpTime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Emptiness</td>
<td></td>
</tr>
<tr>
<td>Universality</td>
<td>2-ExpTime</td>
</tr>
<tr>
<td>Inclusion</td>
<td>2-ExpTime</td>
</tr>
</tbody>
</table>
Model-checking MSO formulas

Given a formula $\phi$ over MSCNs we construct a formula $\psi$ over ADTs such that

The interpretation:
Model-checking MSO formulas

Given a formula $\varphi$ over MSCNs we construct a formula $\psi$ over ADTs such that

For any MSCN $M$, $M \models \varphi$ iff $T \models \psi$ for any ADT $T$ representing a split-width $k$ derivation of $M$.

The interpretation:

- The domain is the set of leaves.
- Message, Nesting are checked examining “common” parent.
- Only process successor needs little bit of work.
A main result of this paper is that, for the class of MSCNs with bounded split-width, satisfiability is decidable. The proof technique is to consider finitely labelled abstractions of MSCNs. A formula $\phi$ is satisfiable if it admits a decomposition-tree $D$ of a split-MSCN such that 1) leaves are labelled with basic split-MSCNs, 2) every internal node $p$ satisfies $\phi$, and 3) every internal node $y$ has a single child $q$ and 4) every internal node $x$ having two children $y_1$ and $y_2$.

Given a split-MSCN $S$, we can build a tree automaton $A$ accepting the language of $S$ and we can build a corresponding tree automaton $A'$. The above definition of split-width is inspired by the algebraic definition of tree-width. We choose split-width instead of tree-width because it is decomposable for some MSCNs of size at most $t^2$. A 4-split-MSCN is a binary tree $T$ with labels $x, y, z$. Figure 2.2 shows a 4-decomposition-tree and its abstraction $ADT(T)$. A 4-decomposition-tree $DT$ is shown on the left of Figure 2.2. Note that any split-MSCN is decomposable.
Nested words have split-width $\leq 3$
Nested Words

Nested words have split-width $\leq 3$
Nested Words

Nested words have split-width \( \leq 3 \)
Nested Words

Nested words have split-width $\leq 3$
Nested Words

Nested words have split-width ≤ 3
Nested Words

Nested words have split-width $\leq 3$

**Theorem.** MSO is decidable over nested words (VPLs).
A *context* is a set of consecutive positions which involves at most one stack.
Bounded scope multi-pushdown systems

A *context* is a set of consecutive positions which involves at most one stack.
A *context* is a set of consecutive positions which involves at most one stack.

**Bounded Scope MNWs:** Fix parameter $m$. For any nesting edges, no more than $m$ different contexts between its source and target.

---

Bounded scope multi-pushdown systems

A context is a set of consecutive positions which involves at most one stack.

Bounded Scope MNWs: Fix parameter m. For any nesting edges, no more than m different contexts between its source and target.

Theorem. S-W at most m + 2.

A *phase* is a set of consecutive positions which involves at most one stack.
A *phase* is a set of consecutive positions which involves at most one stack.
A phase is a set of consecutive positions which involves at most one stack.

**Bounded Phase MNWs:** Fix parameter $p$. At most $p$ phases.

Bounded phase multi-pushdown systems

A *phase* is a set of consecutive positions which involves at most one stack.

*Bounded Phase MNWs*: Fix parameter $p$. At most $p$ phases.

**Theorem.** $S$-$W$ at most $2^p$.

Ordered MNWs: Priority among the stacks. Returns agree with the priority. When a stack pops, all higher priority stacks are empty.


Ordered MNWs: Priority among the stacks. Returns agree with the priority. When a stack pops, all higher priority stacks are empty.

**Theorem.** $S-W$ at most $2^s$. 


HMSCs (or MSGs)

Split-width bounded by the maximum split-width of constituent MSCs
HMSCs (or MSGs)

Split-width bounded by the maximum split-width of constituent MSCs

Unlike CRPs, language is not MSO definable.
Add one process and edges to it in each node.

Language of this HMSC is MSO definable.

Obvious translation for MSO formulas via relativization.
Tree-width/Clique-width

MSO decidability follows.

Technical argument, normalizing derivation trees.

Split-width is a “special case” that is easier to use in the case of behaviours of CRPs.
Tree-width/Clique-width

• Easy translation from split-width to Tree/Clique width
  MSO decidability follows.

• Clique-width to Split-width with linear blow up
  Technical argument, normalizing derivation trees.

Split-width is a “special case” that is easier to use in the case of behaviours of CRPs.
Conclusions

• Split-width: a metric for under-approximate verification

  Equivalent to tree width in power

• Provides a simple technique to prove decidability of all known classes.

  Visual, simple inductive reasoning, limited number of cases to consider.

• Different view, suggests new “natural” classes.

• Schedulable subclasses.

  Restrict to only verified behaviours