Asynchronously Communicating Visibly Pushdown Systems

Domagoj Babić

Google Research

joint work with Zvonimir Rakamarić
School of Computing, University of Utah

Overview

- Motivation
- Communicating Finite State Machines
- Relations
- Two Extensions: Synchronization and (Visible) Stack
- Decidability of Reachability
- Limitations and Future Work

Motivation

- Debugging of large-scale distributed systems:
  - Counters and statistics
  - Unit tests
  - Progressive deployment
  - Difficult localization of root causes of failures

Motivation

- Debugging of large-scale distributed systems:
  - Counters and statistics
  - Unit tests
  - Progressive deployment
  - Difficult localization of root causes of failures
- Penetration of formal methods

Motivation

- Debugging of large-scale distributed systems:
  - Counters and statistics
  - Unit tests
  - Progressive deployment
  - Difficult localization of root causes of failures
- Penetration of formal methods
  - Zero?

Motivation

- Debugging of large-scale distributed systems:
  - Counters and statistics
  - Unit tests
  - Progressive deployment
  - Difficult localization of root causes of failures
- Penetration of formal methods
  - Zero?
  - But there's some hope...

Motivation

- Debugging of large-scale distributed systems:
  - Counters and statistics
  - Unit tests
  - Progressive deployment
  - Difficult localization of root causes of failures
- Penetration of formal methods
  - Zero?
  - But there's some hope...
- Protocol Buffers (Google), Thrift (Facebook), Singularity (MSR)
  - Specification of message formats
  - Automated (de)serialization
  - Could we expand these to describe protocols?

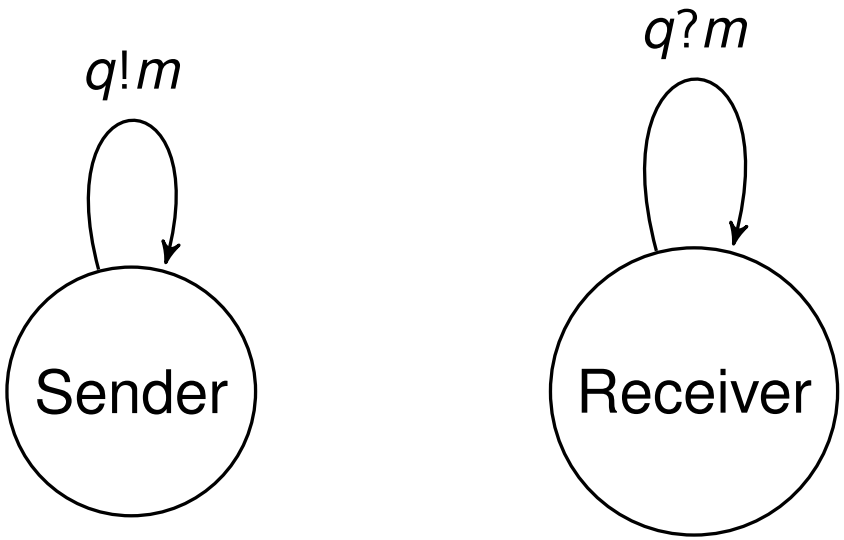
Motivation

- Debugging of large-scale distributed systems:
  - Counters and statistics
  - Unit tests
  - Progressive deployment
  - Difficult localization of root causes of failures
- Penetration of formal methods
  - Zero?
  - But there's some hope...
- Protocol Buffers (Google), Thrift (Facebook), Singularity (MSR)
  - Specification of message formats
  - Automated (de)serialization
  - Could we expand these to describe protocols?
- If so, which formal model would we use?

Desired Model Properties

- Asynchronous message passing
- Distributed concurrently executing processes (with stacks)
- Lossless (unbounded) FIFO message buffers
  - Network protocols mostly guarantees no loss of messages
  - Messages consumed mostly in-order (not in Erlang!)
  - Buffers practically unbounded (bounds often large, vary a lot)
  - Some loss possible when buffers overflow
    (usually indicates under-design or error)
  - Processes use multiple buffers

Communicating Finite State Machines



- Bounded number of
  - Processes (FSMs)
  - One-directional channels (unbounded FIFOs)
- Reachability undecidable
- Decidable if [Pac87]:
  - In every composite state, CFSM has a recognizable channel property
  - I.e., all queue languages are regular and form a recognizable relation

Limitations of Recognizable Relations

- Processes can't send multiple messages to different channels in sequence
- No (unbounded) message counting
- Can't express remote procedure calls (and returns)

Relations

$$\underset{\text{[Tho90]}}{\overset{\checkmark}{Rec}} \subset \underset{\text{[EES69]}}{\overset{\checkmark}{Sync}} \subset \underset{\text{[RS59]}}{\overset{\oslash}{DRat}} \subset \underset{\text{[RS59]}}{\overset{\oslash}{Rat}}$$

- Recognizable relations
  - n-tape automaton, each tape operates independently, has its own memory
  - Finite unions of cross-products of regular languages
  - Even when concatenated, still regular
- Synchronized relations
  - n-tape automaton, all heads move in lock-step
  - Can express languages like (a^n, b^n)
- Rational relations — finite n-tape automata
- ($\checkmark$ — denotes that containment is decidable)

Extending CFSMs

1. From FSMs to Visibly Pushdown Automata
2. From Rec to $Sync$

Communicating Visibly Pushdown Transducers

Definition (CVPT)

A CVPT is a tuple $T = (\Sigma_{rcv}, \Sigma_{snd}, Q, S, I, F, \Gamma, \Delta)$ of finite sets, where Σ_{rcv} is an input alphabet, Σ_{snd} an output alphabet, Q a set of unbounded FIFO queues, S a set of states, $I \subseteq S$ a set of initial states, $F \subseteq S$ a set of final states, Γ an alphabet of stack symbols, and Δ a transition relation.

Composite Configuration

Configuration

$C = (s, \sigma, \vec{\varrho}) = (s, \sigma, \varrho_1, \ldots, \varrho_{|Q|}) \in S \times \Gamma^* \times \bigtimes q \in Q(\Sigma_{snd_q} \cup \Sigma_{rcv_q})^*$

Composite Configuration

$\vec{C} = (C_1, \ldots, C_n)$

Transition Relation for CVPTs

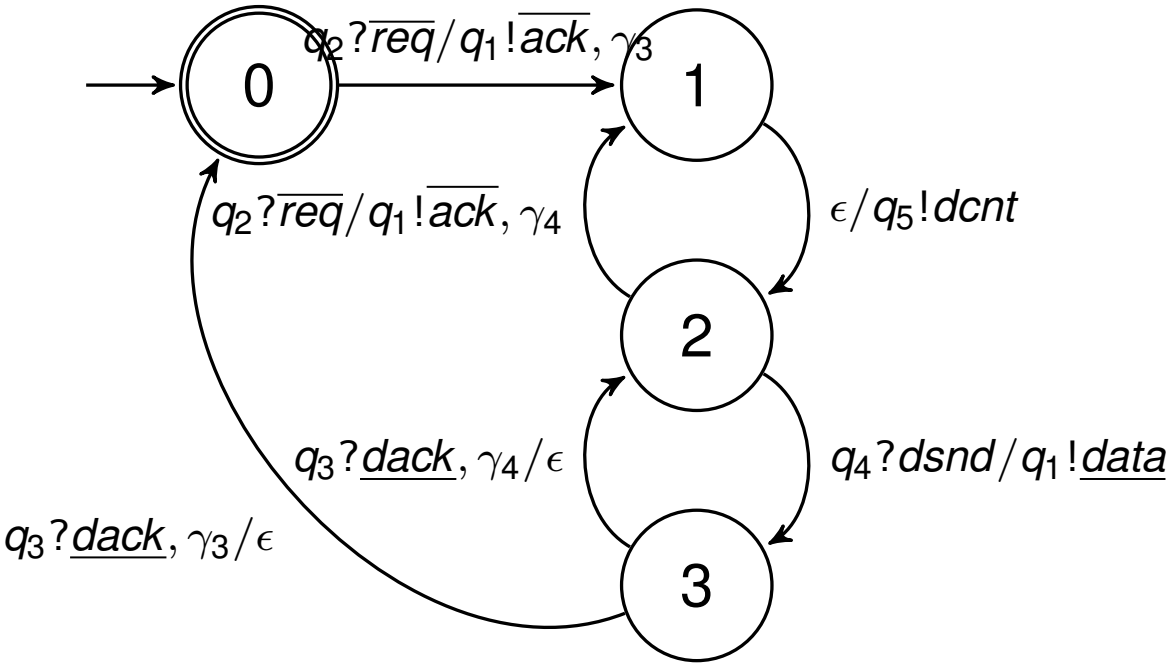
- Σ_{rcv} partitioning: $\Sigma_{rcv} = \Sigma_c \cup \Sigma_r \cup \Sigma_i$ ($\cup$ is disjoint uinion)
- $\Delta = \delta_c \cup \delta_r \cup \delta_i$

$$\begin{aligned} \text{Call} \quad & \delta_c \subseteq S \times \Sigma_c \times (\Sigma_{snd} \cup \{\epsilon\}) \times \Gamma \times S \\ \text{Return} \quad & \delta_r \subseteq S \times \Sigma_r \times \Gamma \times (\Sigma_{snd} \cup \{\epsilon\}) \times S \\ \text{Internal} \quad & \delta_i \subseteq S \times (\Sigma_i \cup \{\epsilon\}) \times (\Sigma_{snd} \cup \{\epsilon\}) \times S \end{aligned}$$

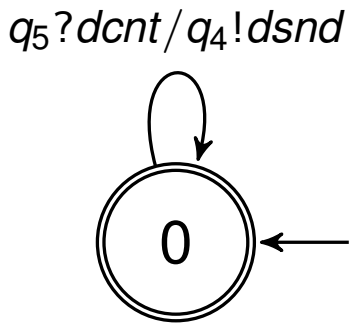
- If $(s_1, q_1?\overline{m}_1, q_2!m_2, \gamma, s_2) \in \delta_c$, then
  $C \xrightarrow[c]{q_1?\overline{m}_1 / q_2!m_2, \gamma} C[s_1 \leftarrow s_2, \sigma \leftarrow \gamma \cdot \sigma, \overline{m}_1 \cdot \varrho_1 \leftarrow \varrho_1, \varrho_2 \leftarrow \varrho_2 \cdot m_2]$
- If $(s_1, q_1?\underline{m}_1, \gamma, q_2!m_2, s_2) \in \delta_r$, then
  $C \xrightarrow[r]{q_1?\underline{m}_1, \gamma / q_2!m_2} C[s_1 \leftarrow s_2, \gamma \cdot \sigma \leftarrow \sigma, \underline{m}_1 \cdot \varrho_1 \leftarrow \varrho_1, \varrho_2 \leftarrow \varrho_2 \cdot m_2]$
- If $(s_1, q_1?m_1, q_2!m_2, s_2) \in \delta_i$, then
  $C \xrightarrow[i]{q_1?m_1 / q_2!m_2} C[s_1 \leftarrow s_2, m_1 \cdot \varrho_1 \leftarrow \varrho_1, \varrho_2 \leftarrow \varrho_2 \cdot m_2]$

Example of a System of CVPTs

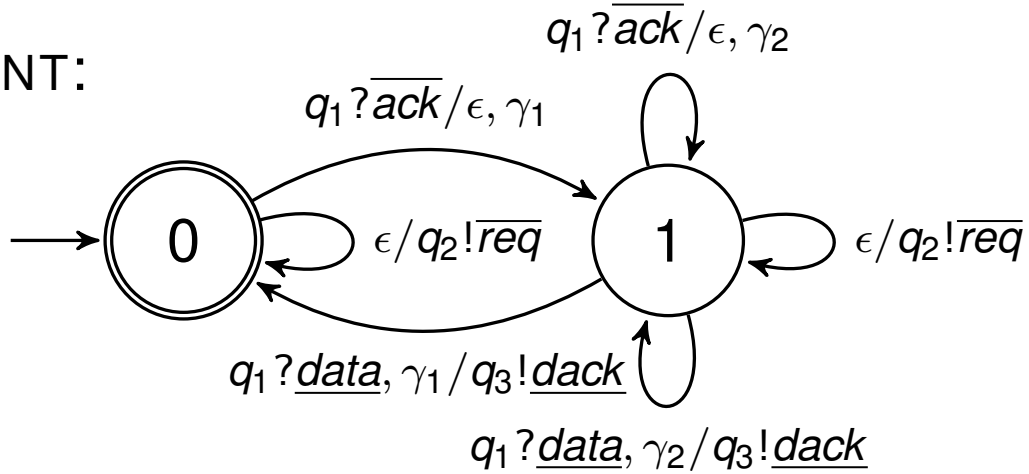
Server:



Database:



Client:



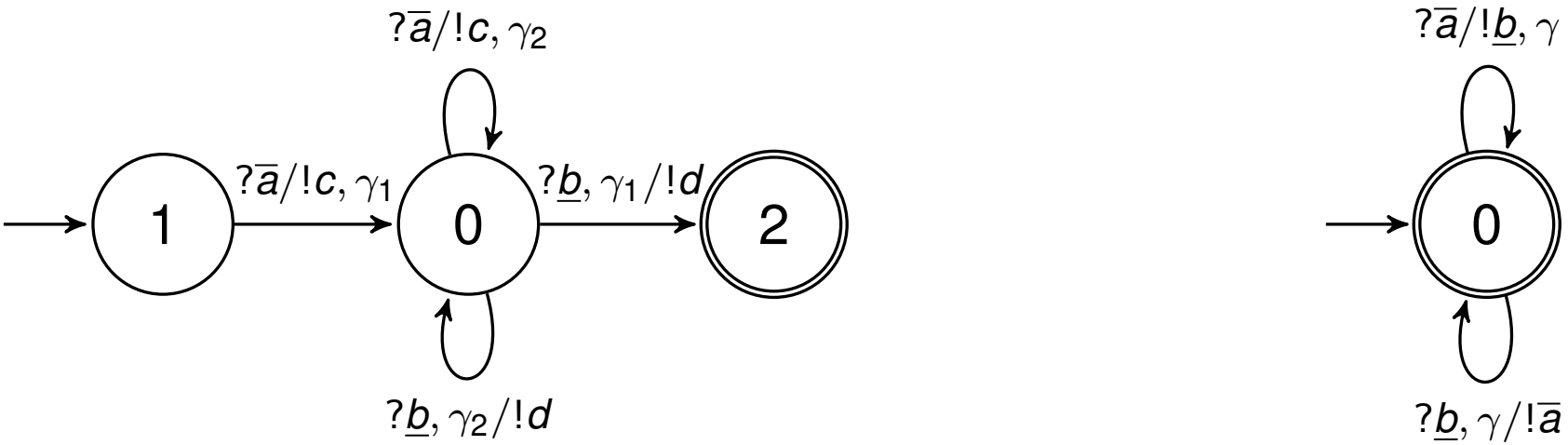
CVPT Composition

- Even for VPL inputs, CVPTs can generate context-free outputs [RS08]

Definition (Composition Property)

A CVPT T_j is said to be *composable* if a projection of its output language (i.e., a transduction of some VPL L) onto the input alphabet of any T_i is a VPL.

Examples of CVPTs not Satisfying the Composition Property



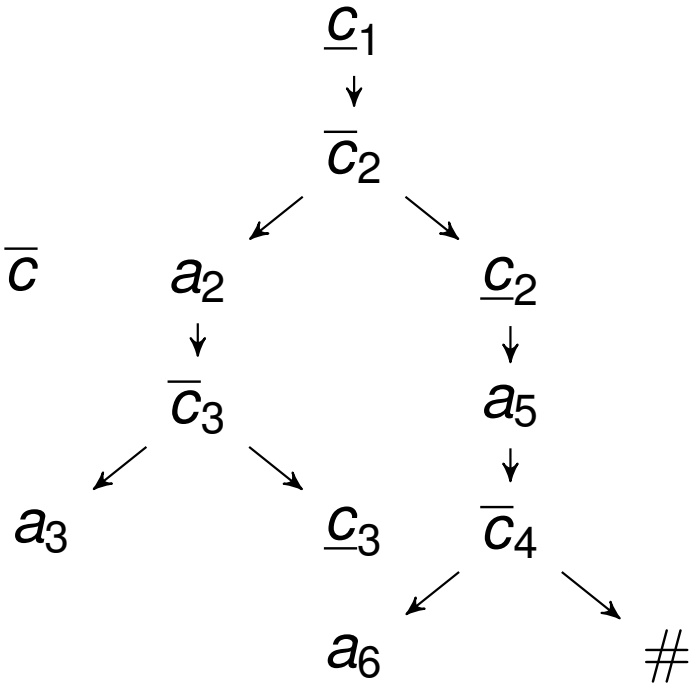
- The left one generates a context-free language $a^n b^n$
- The right one reverses calls and returns, generating $\underline{b}^n \overline{a}^n$

Synchronized Tree Relations (Trees)

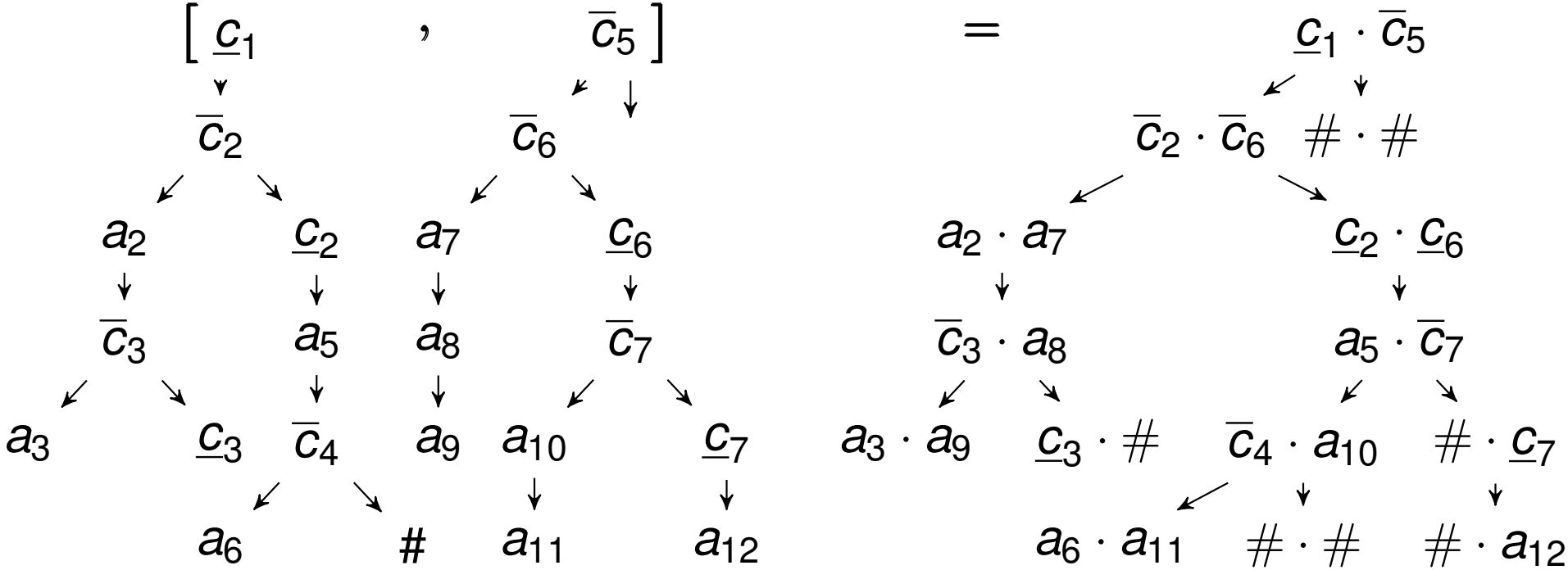
- Mapping η for a Word:
  $\underline{c}_1 \cdot \overline{c}_2 \cdot a_2 \cdot \overline{c}_3 \cdot a_3 \cdot \underline{c}_3 \cdot \underline{c}_2 \cdot a_5 \cdot \overline{c}_4 \cdot a_6$

- First, we translate VPL words to trees

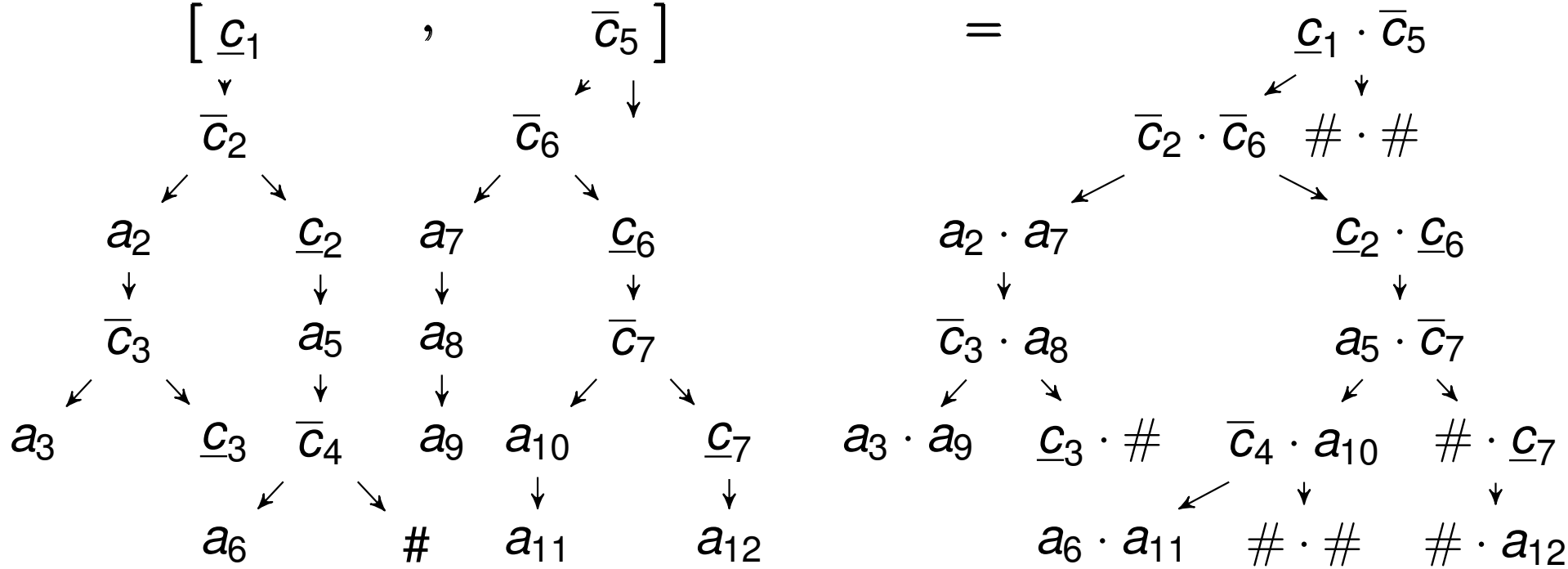
$$
\begin{aligned}
\eta(\epsilon) &= \#; \\
\eta(\overline{c}w) &= \overline{c}(\eta(w), \#), \text{ if no } \underline{c} \text{ matching } \overline{c} \\
\eta(\overline{c}w\underline{c}w') &= \overline{c}(\eta(w), \eta(\underline{c}w')) \\
\eta(aw) &= a(\eta(w)), \text{ if } a \in \Sigma_i \cup \Sigma_r
\end{aligned}
$$



Synchronized Tree Relations (Overlap)



Synchronized Tree Relations (Overlap)



Definition (Synchronized Configuration Property)

An asynchronous system of CVPTs has the *synchronized configuration property* iff in every reachable composite control state $\vec{s}$, the encoding $[\eta\left(\mathbf{L}_{qs}\left(\vec{s}\right)\right)]$ is a synchronized tree relation, i.e.,

$\{[\eta(\sigma_1), \ldots, \eta(\sigma_n), \eta(\varrho_1), \ldots, \eta(\varrho_k)] \mid (\sigma_1, \ldots, \sigma_n, \varrho_1, \ldots, \varrho_k) \in \mathbf{L}_{qs}\left(\vec{s}\right)\} \in Sync^{\mathcal{V}}$

(where $\mathbf{L}_{qs}\left(\vec{s}\right) = \left\{\vec{C}.\varrho, \vec{C}.\sigma \mid \vec{C}_0 \xrightarrow{*} \vec{C} \wedge \vec{C}.s = \vec{s}\right\}$)

Decidability of Reachability

Theorem

Reachability is decidable for a system of composable CVPTs with the synchronized configuration property.

Consistency

Definition (Consistency)

Let $M = (T_1, \ldots, T_n)$ be a system of CVPTs. We say that relation $\mathbf{L} \subseteq \prod_{1 \leq i \leq n} S_i \times \prod_{q \in Q} \Sigma^*_{rcv_q} \times \prod_{1 \leq i \leq n} \Gamma_i^*$ is *consistent* (with respect to M) if $\vec{C}_1 \xrightarrow{*} \vec{C}_2$ and $\left(\vec{C}_1.\varrho, \vec{C}_1.\sigma\right) \in \mathbf{L}(\vec{C}_1.s)$ imply $\left(\vec{C}_2.\varrho, \ \vec{C}_2.\sigma\right) \in \mathbf{L}(\vec{C}_2.s)$.

Proof Sketch

- Existential proof
- Two semi-algorithms
- If $\vec{C}$ reachable — can be reached by BFS
- If $\vec{C}$ unreachable — must exist a consistent synchronized relation **L**, such that $\left(\vec{C}.\varrho, \vec{C}.\sigma\right) \notin \mathbf{L}(\vec{C}.s)$.
- Since synchronized tree languages can be identified by enumeration [Gol67], we can enumerate consistent relations

Algorithm Based on the Proof?

- Would be hopelessly inefficient, but. . .

Algorithm Based on the Proof?

- Would be hopelessly inefficient, but. . .
- Seems like regular (tree) model checking is the way to go

Algorithm Based on the Proof?

- Would be hopelessly inefficient, but. . .
- Seems like regular (tree) model checking is the way to go
- If programmers provided the invariant for at least one edge in every loop in the composite state graph — model checking can be done efficiently

Resynchronizable Relations

- Accepted by n-tape automata with an *a-priori* bounded delay among heads
- Heads may not be perfectly synchronized
- Can be reduced to a finite union of component-wise products of synchronized relations and finite sets
- Example: $R = (b^m aab^k, c^m d^k)$

  k=0 $(b^m, c^m) \cdot (a^2, \epsilon^2)$
  k=1 $(b^m, c^m) \cdot (a^2 b, d \cdot \epsilon^2)$
  k=2 $(b^m, c^m) \cdot (a^2 bb, d^2 \cdot \epsilon^2)$
  k¿2 $(b^m aab^{k-2}, c^m d^k) \cdot (bb, \epsilon^2)$

Switching Automata

- Generalization of resynchronizable languages
- Tapes can be switched on and off
- If the number of switches is bounded, accepted relations can be translated into a synchronized relation

Limitations and Future Work

- Synchronized relations still somewhat limited
  - Not necessarily perfectly synchronized
  - Some relaxations possible (e.g., resynchronizable relations)

Limitations and Future Work

- Synchronized relations still somewhat limited
  - Not necessarily perfectly synchronized
  - Some relaxations possible (e.g., resynchronizable relations)
  - Are there more expressive natural families of relations in between $Sync$ and $DRat$?

Limitations and Future Work

- Synchronized relations still somewhat limited
  - Not necessarily perfectly synchronized
  - Some relaxations possible (e.g., resynchronizable relations)
  - Are there more expressive natural families of relations in between $Sync$ and $DRat$?
  - More general theory of resynchronizable relations needed

Limitations and Future Work

- Synchronized relations still somewhat limited
  - Not necessarily perfectly synchronized
  - Some relaxations possible (e.g., resynchronizable relations)
  - Are there more expressive natural families of relations in between $Sync$ and $DRat$?
  - More general theory of resynchronizable relations needed
- Checking that relation is synchronized is undecidable

Limitations and Future Work

- Synchronized relations still somewhat limited
  - Not necessarily perfectly synchronized
  - Some relaxations possible (e.g., resynchronizable relations)
  - Are there more expressive natural families of relations in between $Sync$ and $DRat$?
  - More general theory of resynchronizable relations needed
- Checking that relation is synchronized is undecidable
  - Is it possible to guarantee (say, by a type system) that all relations are synchronized by construction?

Limitations and Future Work

- Synchronized relations still somewhat limited
  - Not necessarily perfectly synchronized
  - Some relaxations possible (e.g., resynchronizable relations)
  - Are there more expressive natural families of relations in between $Sync$ and $DRat$?
  - More general theory of resynchronizable relations needed
- Checking that relation is synchronized is undecidable
  - Is it possible to guarantee (say, by a type system) that all relations are synchronized by construction?
- Couldn't find a real system that naturally fits into the model
  - Large majority of systems too simple
  - Suggestions?

References

S. Eilenberg, C. C. Elgot, and J. C. Shepherdson.
Sets recognized by n-tape automata.
Journal of Algebra, 13:447–464, 1969.

E. Mark Gold.
Language identification in the limit.
Info. and Control, 10(5):447–474, 1967.

Jan K. Pachl.
Protocol description and analysis based on a state transition model with channel expressions.
In *Intl. Conf. on Protocol Specification, Testing and Verification (PSTV)*, pages 207–219, 1987.

M. O. Rabin and D. Scott.
Finite automata and their decision problems.
IBM Journal of Research and Development, 3:114–125, 1959.

Jean-François Raskin and Frédéric Servais.
Visibly pushdown transducers.
In *Intl. Colloquium on Automata, Languages and Programming (ICALP), Part II*, pages 386–397, 2008.

Wolfgang Thomas.
On logical definability of trace languages.
In *ASMICS workshop, Technical University of Munich, Report TUM-I9002*, pages 172–182, 1990.