haylu

Non-Sequential Theory of Distributed Systems

Lecture MPRI M2

Benedikt Bollig & Paul Gastin

October 31, 2018

Contents

CHAPTER 1

Introduction

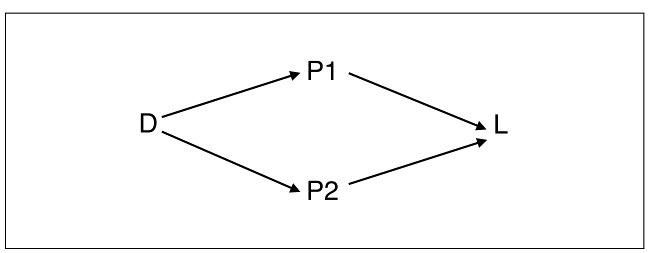
1st Lecture

Before we go into formal definitions, we give some examples of the kind of problems that are treated in the lecture.

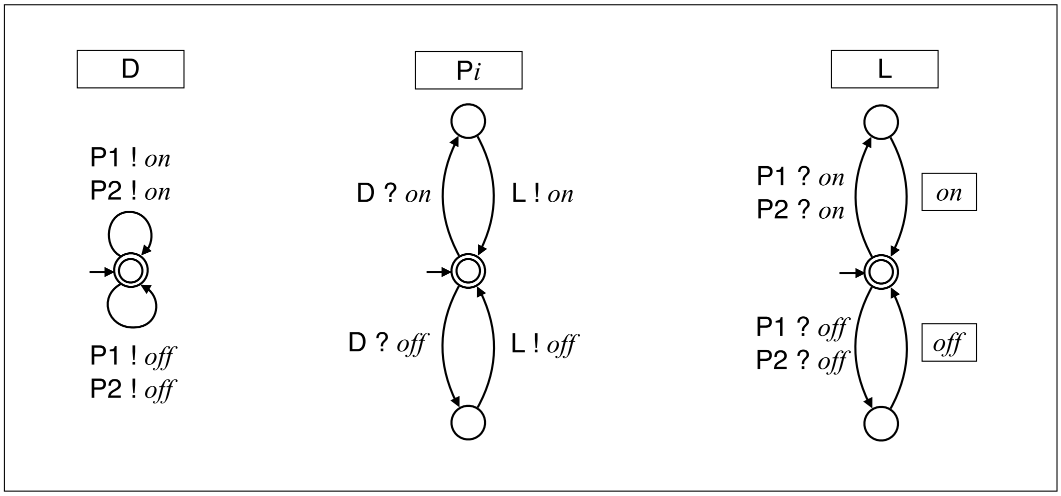
1.1 Synthesis and Control

1.1.1 A simple protocol

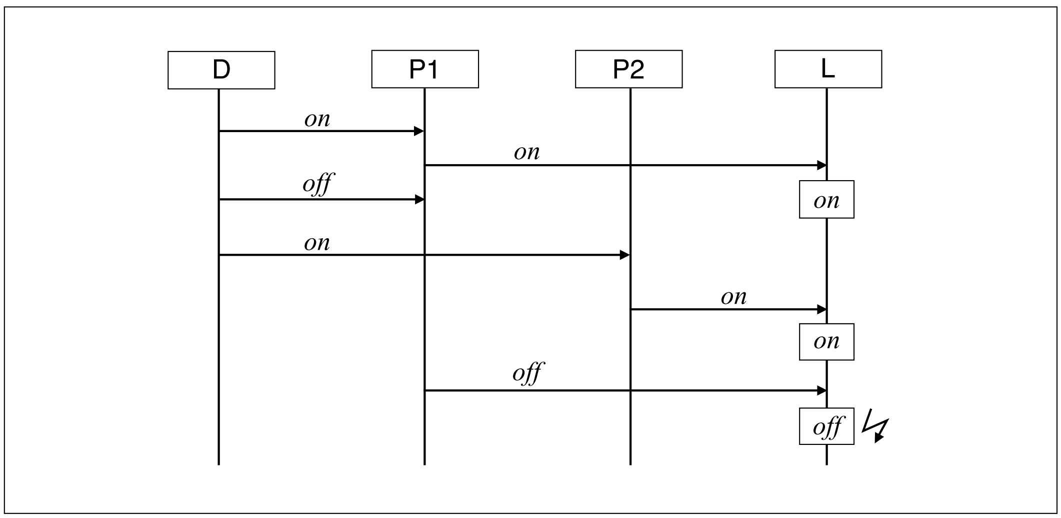
In this lecture, we consider systems that consist of a fixed finite number of processes that are connected via communication media. Consider the following architecture and assume that communication is synchronous.



Consider a process D (the device), which communicates with two processes, P1 and P2, sending them regularly its current status (*on* or *off*). Every status message is forwarded, by P1 or P2, to L (a lamp, or display). The latter is either *on* or *off* and, upon reception of a message, may change (or not) the current status. For simplicity, we will assume rendez-vous (i.e., handshake) communication. A naive implementation looks as follows:



However, this may result in the following communication scenario, which yields the "wrong" output:

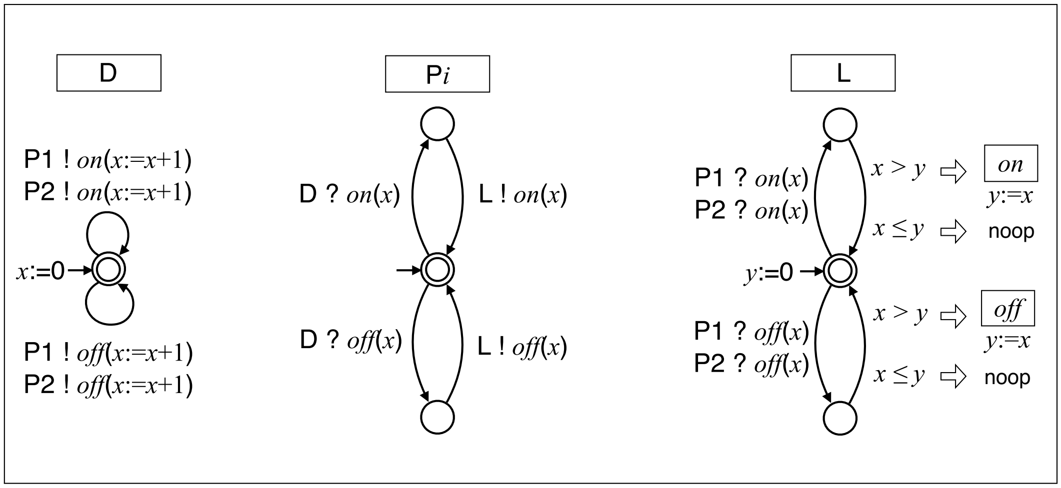


The problem is that L does not know which of the last two messages that it receives is the latest one.

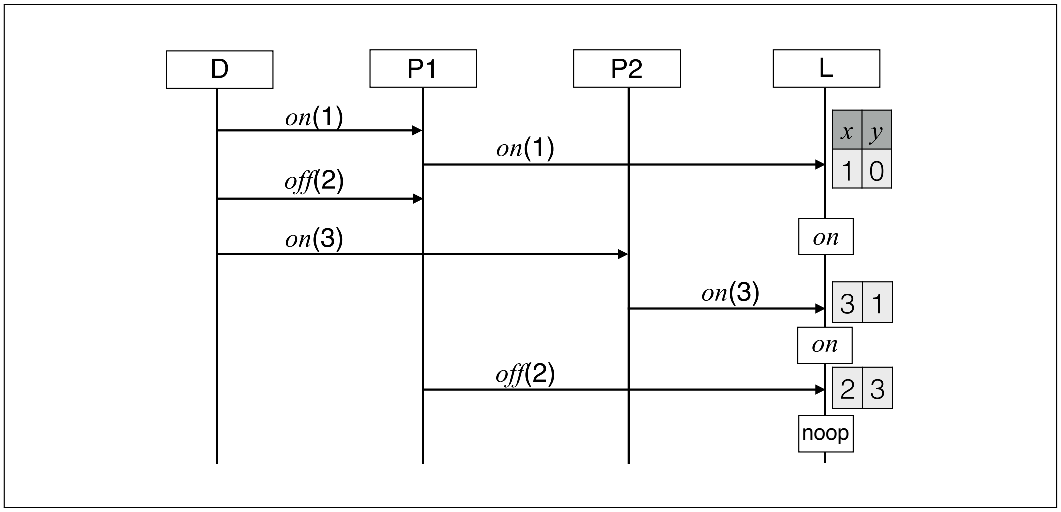
Suppose we are not allowed to change the architecture of the system. We are only allowed to add *local and deterministic* controllers to each process that add additional message contents. In particular, a controller should not block any of the possible actions of the given system. Is there still a way to control the protocol in a way such that L shows the correct result?

1.1.2 A first solution.

A first, rather obvious solution would consist in using timestamps. Strictly increasing timestamps are sent by D along with a status message, forwarded by P1 and P2, and then compared by L with its latest knowledge.



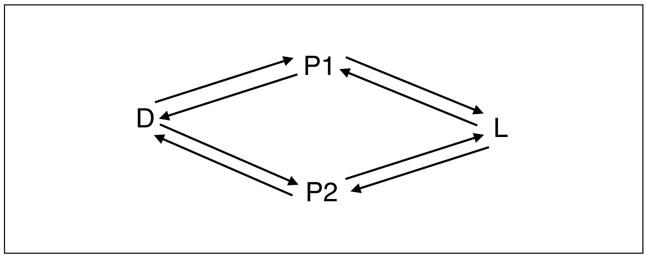
It is easy to see that the protocol is correct in the sense that L does not update its display based on outdated information. The following scenario shows that the error from the previous solution is indeed avoided.



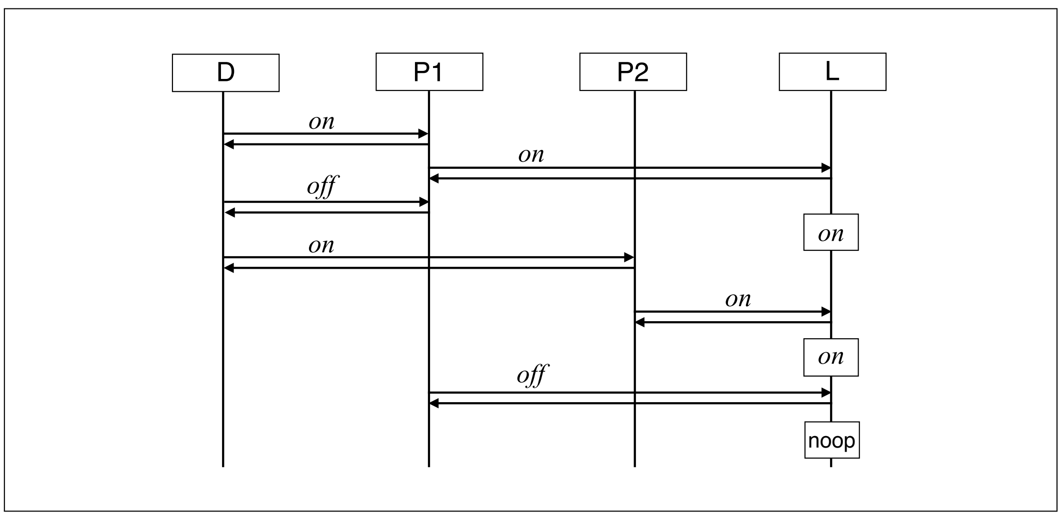
1.1.3 Towards finite-state solutions

The previous solution uses infinitely many states, since time stamps are strictly increasing. This may not be adequate in the realm of *reactive* processes, which are supposed to run forever. However, finding a finite-state solution is difficult. It is actually impossible for the above specification.

Fortunately, there are guaranteed solutions in the case where every message is immediately followed by an acknowledgment. So, let us assume the following architecture:



Moreover, the allowed communication scenarios look as follows, i.e., every message is *immediately* followed by an acknowledgment (which can be augmented with additional messages):



In this particular case, we get a finite-state local controller. However, it is not always easy to come up with a solution.

Exercise 1.1. Try to find a finite-state controller. ◊

The process of finding a controller is actually error-prone. So, it is natural to ask the following question:

$\boxed{\textit{Is there an automatic way to generate a controller from a specification?}}$

The answer is YES, provided that the specification satisfies some properties.

In our example, the specification $L_{spec} \subseteq \Sigma^*$ could be a word language over the following alphabet:

$$\begin{aligned}\Sigma = \quad & \{\langle \mathsf{D} \overset{on}{\rightleftarrows} \mathsf{P1}\rangle\,, \langle \mathsf{D} \overset{on}{\rightleftarrows} \mathsf{P2}\rangle\,, \langle \mathsf{D} \overset{off}{\rightleftarrows} \mathsf{P1}\rangle\,, \langle \mathsf{D} \overset{off}{\rightleftarrows} \mathsf{P2}\rangle\} \\ \cup \quad & \{\langle \mathsf{P1} \overset{on}{\rightleftarrows} \mathsf{L}\rangle\,, \langle \mathsf{P2} \overset{on}{\rightleftarrows} \mathsf{L}\rangle\,, \langle \mathsf{P1} \overset{off}{\rightleftarrows} \mathsf{L}\rangle\,, \langle \mathsf{P2} \overset{off}{\rightleftarrows} \mathsf{L}\rangle\} \\ \cup \quad & \{on, off, \mathsf{noop}\}\end{aligned}$$

Let $\Sigma_{\mathsf{P1}} \subseteq \Sigma$ be the subalphabet containing those actions involving $\mathsf{P1}$. Moreover, let $\Sigma_{\mathsf{P2}} \subseteq \Sigma$ contain those with $\mathsf{P2}$, and so on. In particular, $\{on, off, \mathsf{noop}\} \subseteq \Sigma_{\mathsf{L}}$. With this, L_{spec} is the set of words $w \in \Sigma^*$ satisfying the following:

(R1) The projection of w to Σ_{P1} is contained in

$$(\langle \mathsf{D} \overset{on}{\rightleftarrows} \mathsf{P1}\rangle\langle \mathsf{P1} \overset{on}{\rightleftarrows} \mathsf{L}\rangle + \langle \mathsf{D} \overset{off}{\rightleftarrows} \mathsf{P1}\rangle\langle \mathsf{P1} \overset{off}{\rightleftarrows} \mathsf{L}\rangle)^*\,.$$

(R2) The projection of w to Σ_{P2} is contained in

$$(\langle \mathsf{D} \overset{on}{\rightleftarrows} \mathsf{P2}\rangle\langle \mathsf{P2} \overset{on}{\rightleftarrows} \mathsf{L}\rangle + \langle \mathsf{D} \overset{off}{\rightleftarrows} \mathsf{P2}\rangle\langle \mathsf{P2} \overset{off}{\rightleftarrows} \mathsf{L}\rangle)^*\,.$$

(R3) The projection of w to Σ_{L} is contained in

$$\Big((\langle \mathsf{P1} \overset{on}{\rightleftarrows} \mathsf{L}\rangle + \langle \mathsf{P2} \overset{on}{\rightleftarrows} \mathsf{L}\rangle + \langle \mathsf{P1} \overset{off}{\rightleftarrows} \mathsf{L}\rangle + \langle \mathsf{P2} \overset{off}{\rightleftarrows} \mathsf{L}\rangle)(on + off + \mathsf{noop})\Big)^*\,.$$

(R4) "The display is updated iff the last (previous) status message emitted by D that has already been followed by a forward was not yet followed by a corresponding update by L."

Exercise 1.2. Formalize the requirement (R4) in terms of a finite automaton or an MSO formula. ♢

Here are some example words to illustrate L_{spec}:

- $\langle \mathsf{D} \overset{on}{\rightleftarrows} \mathsf{P1}\rangle\langle \mathsf{D} \overset{off}{\rightleftarrows} \mathsf{P2}\rangle\langle \mathsf{P2} \overset{off}{\rightleftarrows} \mathsf{L}\rangle\, off\, \langle \mathsf{P1} \overset{on}{\rightleftarrows} \mathsf{L}\rangle \mathsf{noop} \quad \in \quad L_{spec}\,,$
- $\langle \mathsf{D} \overset{on}{\rightleftarrows} \mathsf{P1}\rangle\langle \mathsf{P1} \overset{on}{\rightleftarrows} \mathsf{L}\rangle\langle \mathsf{D} \overset{on}{\rightleftarrows} \mathsf{P1}\rangle\, on \quad \notin \quad L_{spec},$
- $\langle \mathsf{D} \overset{on}{\rightleftarrows} \mathsf{P1}\rangle\langle \mathsf{P1} \overset{on}{\rightleftarrows} \mathsf{L}\rangle\langle \mathsf{D} \overset{off}{\rightleftarrows} \mathsf{P2}\rangle\, on\, \langle \mathsf{P2} \overset{off}{\rightleftarrows} \mathsf{L}\rangle\, off \quad \in \quad L_{spec}\,,$
- $\langle \mathsf{D} \overset{on}{\rightleftarrows} \mathsf{P1}\rangle\langle \mathsf{P1} \overset{on}{\rightleftarrows} \mathsf{L}\rangle\, on\, \langle \mathsf{D} \overset{off}{\rightleftarrows} \mathsf{P2}\rangle\langle \mathsf{P2} \overset{off}{\rightleftarrows} \mathsf{L}\rangle\, off \quad \in \quad L_{spec}\,.$

Obviously, L_{spec} is a regular language. Moreover, it is closed under *permutation rewriting of independent events*. The latter means that changing the order of neighboring independent actions in a word does not affect membership in L_{spec}. This seems natural, since the order of independent event cannot be enforced by a distributed protocol. Here, two actions are said to be *independent* if they involve distinct processes. For example,

- $\langle \mathsf{D} \overset{on}{\rightleftarrows} \mathsf{P1}\rangle$ and on are independent,
- $\langle \mathsf{D} \overset{on}{\rightleftarrows} \mathsf{P1}\rangle$ and $\langle \mathsf{P2} \overset{on}{\rightleftarrows} \mathsf{L}\rangle$ are independent,
- $\langle \mathsf{D} \overset{on}{\rightleftarrows} \mathsf{P1}\rangle$ and $\langle \mathsf{D} \overset{off}{\rightleftarrows} \mathsf{P2}\rangle$ are *not* independent,
- $\langle \mathsf{D} \overset{on}{\rightleftarrows} \mathsf{P2}\rangle$ and $\langle \mathsf{P2} \overset{on}{\rightleftarrows} \mathsf{L}\rangle$ are *not* independent.

Exercise 1.3. Show that L_{spec} is closed under permutation rewriting. $\diamondsuit$

The following is a fundamental result of concurrency theory (yet informally stated):

Theorem [Zielonka 1987]:

Let L be a regular set of words that is closed under permutation rewriting of independent events. There is a deterministic finite-state distributed protocol that realizes L.

Thus, the specification L_{spec} could indeed be realized as a distributed program.

There are, however, regular specifications that are not realizable (and, therefore, are not closed under permutation rewriting). Consider the language

$$L = (\langle \mathsf{D} \overset{on}{\rightleftarrows} \mathsf{P1}\rangle\langle \mathsf{P2} \overset{on}{\rightleftarrows} \mathsf{L}\rangle)^* \,.$$

Though L is regular, it is not closed under permutation rewriting. Even worse, the closure under permutation is not regular anymore. Specification L says that there are as many messages from D to $\mathsf{P1}$ as from $\mathsf{P2}$ to L. Intuitively, it is clear that this cannot be realized by a finite-state system in a distributed fashion: There is no communication going on between $\mathsf{D}/\mathsf{P1}$ on the hand, and $\mathsf{P2}/\mathsf{L}$ on the other hand.

1.2 Modeling behaviors as graphs

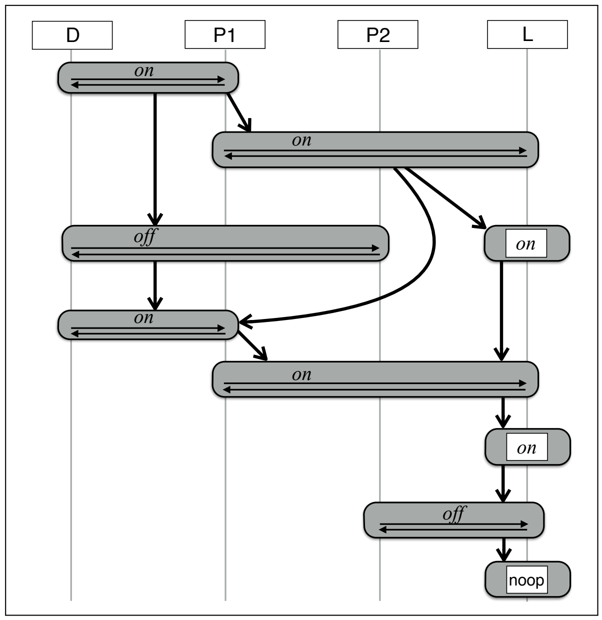
1.2.1 Partial orders

Requirement (R4) from the last section is somehow awkward to write down. The reason is that a word over Σ imposes an ordering of events that, actually, are not causally related in the corresponding execution. When we say "last", this refers to the "last" position in the word. Consider, for example, the word

$$\begin{aligned} w = \quad & \langle \mathsf{D} \overset{on}{\rightleftarrows} \mathsf{P1}\rangle\langle \mathsf{P1} \overset{on}{\rightleftarrows} \mathsf{L}\rangle\langle \mathsf{D} \overset{off}{\rightleftarrows} \mathsf{P2}\rangle\, on \\ & \langle \mathsf{D} \overset{on}{\rightleftarrows} \mathsf{P1}\rangle\langle \mathsf{P1} \overset{on}{\rightleftarrows} \mathsf{L}\rangle\, on \langle \mathsf{P2} \overset{off}{\rightleftarrows} \mathsf{L}\rangle \mathsf{noop} \in L_{spec} \,. \end{aligned}$$

The "last position" right before the on in the first line is actually in no way related to on. So, it is not natural (and not needed) to include it in what we mean by "last". In our reasoning, a more relaxed ordering has to be recovered from the word ordering. So, why not directly reason about the causal order as it is imposed by a distributed system?

In the following, we do not consider an execution of a system as a word, i.e., a total order, but rather as a partial order. The partial order is already suggested by the message diagrams that we used to argue about our distributed protocols. Consider the execution below, whose partial order is represented by its Hasse diagram. It corresponds to the above word w.



Suppose this partial order is denoted by $\leq$. Then, (R4) can be conveniently rephrased (as we do not need the part "... that has already been followed by ..." anymore):

(R4') "When L performs *on* (*off*, respectively), then the last (wrt. $\leq$) status message sent by D should also be *on* (*off*, respectively). Moreover, a display operation should be noop iff there has already been an acknowledgement between the latest status message and that operation."

This can very easily be expressed in monadic second-order (MSO) logic over partial orders, using the partial order $\leq$.

Exercise 1.4. Give an MSO formula for (R4'). ♢

An advantage of MSO logic that is directly interpreted over partial orders is the following theorem:

Theorem [Thomas 1990]:
Let L be an MSO-definable set of partial orders. There is a finite-state distributed protocol that realizes L.

1.2.2 Reasoning about recursive processes

The previous discussion somehow motivates the naming of our lecture. However, in our modeling we will go a step further. One single behavior is not just a partial order, but an (acyclic) graph, which is more general. The edges reflect causal dependencies, but they provide even more information. For example, they may connect a procedure call with the corresponding return, or the sending of a message with its receive.

Consider a system of two processes, $\mathsf{P1}$ and $\mathsf{P2}$, connected by two unbounded FIFO channels (one in each direction). From time to time, $\mathsf{P1}$ sends requests to $\mathsf{P2}$. The latter, when receiving a request, calls a procedure, performs some internal actions, and returns from the procedure, before it sends an acknowledgment. In the scope of a procedure, it may call several subprocedures. Thus, $\mathsf{P1}$ performs actions from the alphabet $\Sigma_1 = \{\langle !\mathsf{req}\rangle, \langle ?\mathsf{ack}\rangle\}$ and $\mathsf{P2}$ performs actions from $\Sigma_2 = \{\langle ?\mathsf{req}\rangle, \langle !\mathsf{ack}\rangle, \langle \mathsf{call}\rangle, \langle \mathsf{ret}\rangle\}$. Let $\Sigma = \Sigma_1 \cup \Sigma_2$.

Let us try to model the protocol in terms of a word language L. It should say that, whenever a request is received, $\mathsf{P2}$ should start a subroutine and send the acknowledgment immediately after returning from this subroutine. Thus, we shall have

$$w_1 = \langle !\mathsf{req}\rangle\, \underline{\langle ?\mathsf{req}\rangle}\, \langle \mathsf{call}\rangle\, \langle \mathsf{call}\rangle\, \langle \mathsf{ret}\rangle\, \langle \mathsf{call}\rangle\, \langle \mathsf{ret}\rangle\, \langle \mathsf{ret}\rangle\, \underline{\langle !\mathsf{ack}\rangle}\, \langle ?\mathsf{ack}\rangle \in L\,.$$

On the other hand, we should have

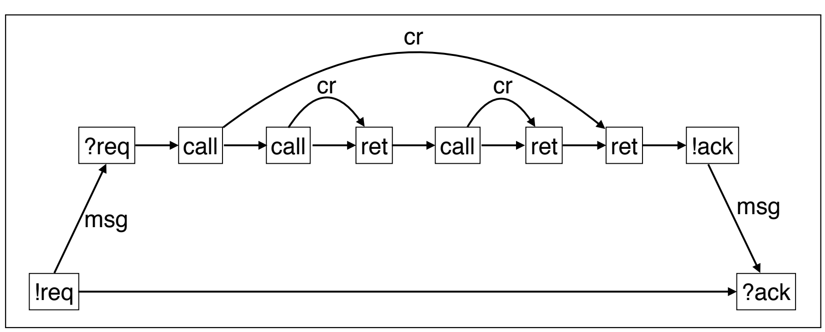
$$w_2 = \langle !\mathsf{req}\rangle\, \underline{\langle ?\mathsf{req}\rangle}\, \langle \mathsf{call}\rangle\, \langle \mathsf{call}\rangle\, \langle \mathsf{ret}\rangle\, \langle \mathsf{call}\rangle\, \langle \mathsf{ret}\rangle\, \underline{\langle !\mathsf{ack}\rangle}\, \langle ?\mathsf{ack}\rangle\, \langle \mathsf{ret}\rangle \notin L\,.$$

But how do we express this in a specification language over words such as MSO logic? Unfortunately, there is no such formula, since L is a non-regular property.

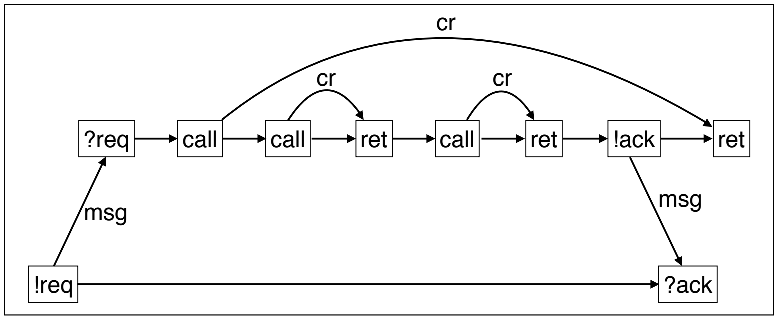
Exercise 1.5. Prove that L is not regular. ♢

The solution is to equip a word with additional information, which allows us to "jump" from a call to its *associated* return position. In other words, we add an edge from a call to the corresponding return:

Then, w_1 corresponds to:



Moreover, w_2 corresponds to:



From this point of view, we can now express our property easily in a suitable MSO logic:

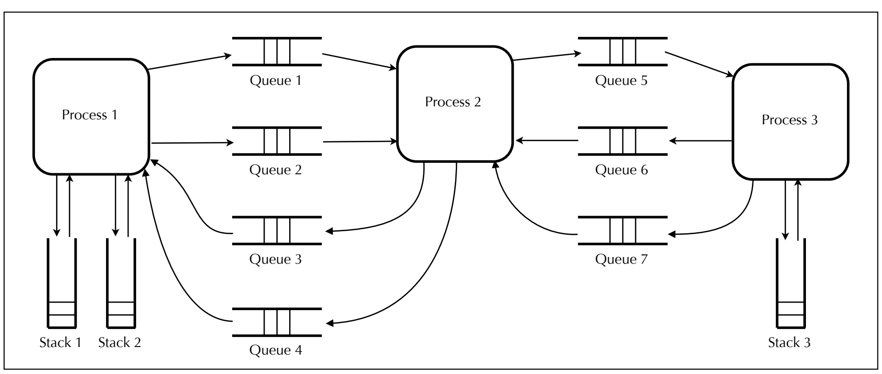
$$\forall x \, \langle ?\mathsf{req}\rangle(x) \Rightarrow \exists y_1, y_2, z\Big(x \to y_1 \wedge \mathsf{cr}(y_1, y_2) \wedge y_2 \to z \wedge \langle !\mathsf{ack}\rangle(z)\Big)$$

It is for similar obvious reasons that, in an asynchronous setting, we connect a send with a receive event.

1.3 Underapproximate Verification

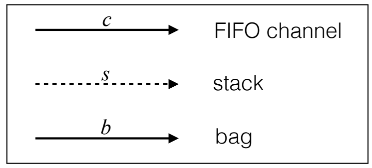
The lecture is concerned with distributed systems with a fixed architecture: There is a finite set of processes, which are connected via some communication media. We will consider stacks, FIFO channels, and bags (with the restriction that stacks connect a process with itself, with the purpose of modeling recursion).[1] A priori, we assume that all media are unbounded. To get decidability or expressivity results, however, we may sometimes impose a bound on channels or stacks. Recall that, in the introductory example, we assumed synchronous communication, which roughly corresponds to FIFO channels with capacity 0.

The following figure illustrates one possible architecture (source: Aiswarya Cyriac's thesis):

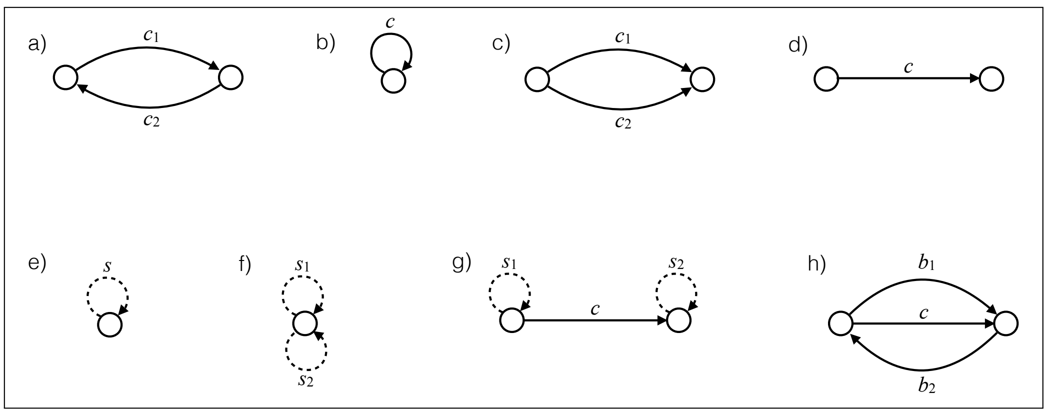


In the following, we will actually depict an architecture as a finite graph with directed edges of three types, depending on the type of the communication medium they represent. We will follow the following convention:

[1] We do not consider *lossy* channels. Cf. Cours 2.9.1.



The most basic verification question is the *reachability problem*, i.e., to ask whether some control state of some process is reachable. Let us examine the decidability status of the reachability problem for the following architectures:



a) Undecidable. A system can simulate a Turing machine TM as follows: Via c_1, process 1 (on the left) sends the current configuration of the TM to process 2. Process 2 only sends every message that it receives immediately back to 1. When 1 receives the configuration, it locally modifies it simulating a transition of TM.

b) Undecidable. Similarly to a), the process sends the current configuration through the channel c. When receiving a configuration from c, it modifies it locally.

c) Undecidable. This can be shown by reduction from PCP (Post's correspondence problem). Process 1 guesses a solution (a sequence of indices, and it sends the corresponding words via channels c_1 and c_2, respectively. Process 2 will then just check if the sequences send through c_1 and c_2 coincide. To do this, it reads alternately from both channels and checks whether both symbols coincide.

d) Decidable. This case can be reduced to synchronous communication and, therefore, reachability in a finite-state system.

e) Decidable. This case corresponds to emptiness of pushdown automata.

f) Undecidable. One can easily simulate a two-counter machine. Equivalently, we may use the concatenation of two stacks to simulate the unbounded tape of a Turing machine.

g) Undecidable. We may use a reduction from the intersection-emptiness problem for two pushdown automata. Modify the two pushdown automata (e.g., using Greibach's normal form) such that they write nondeterministically some accepted word on the stack. So, both pushdown automata will first both choose words w_1 and w_2, respectively. While doing so, process 1 clears its stack sending w_1 letter by letter to process 2. Whenever process 2 receives a letter, it compares it with its stack content, which is then removed.

h) Undecidable. We use a reduction from case b). Process 1 simulates send transitions through channel c. To simulate a receive transition through c, it puts a token into bag b_1, whose value is the message to be received, say m. Process 1 can then only proceed when it finds an acknowledgment token in bag b_2. The latter is provided by process 2 after removing m from c. Note that this procedure can be implemented even when communication between both processes is synchronous.

We conclude that almost all verification problems are undecidable even for very simple system architectures.

In this lecture, we therefore perform *underapproximate verification*: We restrict the behavior of a given system in a non-trivial way that still allows us to reason about it and deduce correctness/faultiness wrt. interesting properties. Let us illustrate some restrictions using some of the undecidable architectures above:

- In all cases, we may assume that communication media have capacity B (existentially B-bounded), for some fixed B.
- In case f), assuming an order on the stacks, we can only pop from the first nonempty stack.
- In case f), we may also impose a bound on the number of *contexts*. In turn, there are several possible definitions of what is allowed in a context:
  - We can only touch one stack.
  - We can only pop from one stack.
  - Many more ...

Under all these restrictions, most standard verification problems (even model checking against MSO-definable properties) becomes decidable, with varying complexities.

In the lecture, we will take a uniform approach to underapproximate verification.

CHAPTER 2

Concurrent Processes with Data Structures

Notation is taken from [AG14].

2.1 The Model

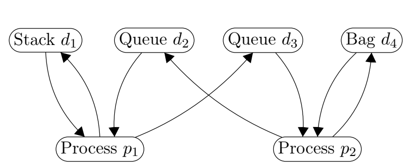
Definition 2.1 (Architecture). *An architecture is a tuple*

$$\mathfrak{A} = (\mathsf{Procs}, \mathsf{DS}, \mathsf{Writer}, \mathsf{Reader})$$

- Procs *finite set of* processes
- $\mathsf{DS} = \mathsf{Stacks} \uplus \mathsf{Queues} \uplus \mathsf{Bags}$ *finite set of* data structures
- $\mathsf{Writer} : \mathsf{DS} \to \mathsf{Procs}$
- $\mathsf{Reader} : \mathsf{DS} \to \mathsf{Procs}$

such that $\mathsf{Writer}(s) = \mathsf{Reader}(s)$ *for all* $s \in \mathsf{Stacks}$. ♢

Example 2.2. Consider the following architecture:



We have $\mathsf{Procs} = \{p_1, p_2\}$, $\mathsf{Stacks} = \{d_1\}$, $\mathsf{Queues} = \{d_2, d_3\}$, and $\mathsf{Bags} = \{d_4\}$. E.g., $\mathsf{Writer}(d_1) = \mathsf{Reader}(d_1) = p_1$. ♢

Definition 2.3 (CPDS). *A* system of concurrent processes with data structures *(*CPDS*) over* $\mathfrak{A}$ *and an alphabet* Σ *is a tuple*

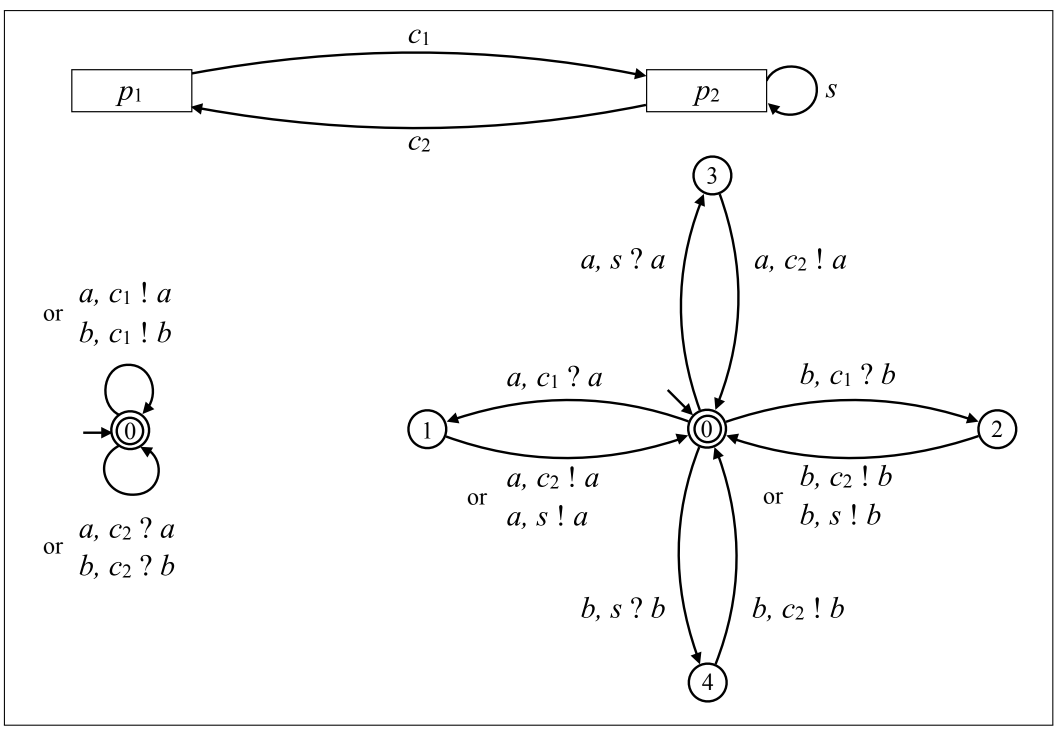
$$\mathcal{S} = (\mathsf{Locs}, \mathsf{Val}, (\rightarrow_p)_{p \in \mathsf{Procs}}, \ell_{in}, \mathsf{Fin})$$

- Locs *nonempty finite set of* locations
- Val *nonempty finite set of* values
- $\ell_{in} \in \mathsf{Locs}$ initial location
- $\mathsf{Fin} \subseteq \mathsf{Locs}^{\mathsf{Procs}}$ global final locations
- $\rightarrow_p$ *transitions of* p
  - *internal transition* $\ell \xrightarrow{a}_p \ell'$
  - *write transition* $\ell \xrightarrow{a,d!v}_p \ell'$ *with* $\mathsf{Writer}(d) = p$
  - *read transition* $\ell \xrightarrow{a,d?v}_p \ell'$ *with* $\mathsf{Reader}(d) = p$

  where $\ell, \ell' \in \mathsf{Locs}$*,* $a \in \Sigma$*,* $d \in \mathsf{DS}$*, and* $v \in \mathsf{Val}$

We let $\mathsf{CPDS}(\mathfrak{A}, \Sigma)$ *be the set of* CPDSs *over* $\mathfrak{A}$ *and* Σ. $\diamondsuit$

Example 2.4. Let $\mathfrak{A}$ be given by $\mathsf{Procs} = \{p_1, p_2\}$, $\mathsf{Queues} = \{c_1, c_2\}$, $\mathsf{Stacks} = \{s\}$, $\mathsf{Bags} = \emptyset$, with $\mathsf{Writer}(c_1) = \mathsf{Reader}(c_2) = p_1$ and $\mathsf{Writer}(c_2) = \mathsf{Reader}(c_1) = p_2$ and $\mathsf{Writer}(s) = \mathsf{Reader}(s) = p_2$. Moreover, let $\Sigma = \{a, b\}$. Consider the client-server system $\mathcal{S}_{\mathrm{cs}}$ over $\mathfrak{A}$ and Σ given as follows:



Process p_1, the client, sends requests of type a or b to process p_2, the server. The latter may acknowledge the request immediately, or put it on its stack (either, because it is busy or because the request does not have a high priority). At any time, however, the server may pop a task from the stack and acknowledge it.

We have $\mathsf{Locs} = \{0,1,2,3,4\}$, $\ell_{\text{in}} = 0$, $\mathsf{Fin} = \{(0,0)\}$, and $\mathsf{Val} = \{a,b\}$. $\diamondsuit$

2.2 Operational Semantics

$\mathcal{S}$ defines (infinite) automaton $\mathcal{A}_{\mathcal{S}} = (\mathsf{States}, \Longrightarrow, s_{\text{in}}, F)$ over $\Gamma = (\mathsf{Procs} \times \Sigma) \cup (\mathsf{Procs} \times \Sigma \times \mathsf{DS} \times \{!, ?\})$

- $\mathsf{States} = \mathsf{Locs}^{\mathsf{Procs}} \times (\mathsf{Val}^*)^{\mathsf{DS}}$
  for $(\overline{\ell}, \overline{z}) \in \mathsf{States}$, we denote $\overline{\ell} = (\ell_p)_{p \in \mathsf{Procs}}$ and $\overline{z} = (z_d)_{d \in \mathsf{DS}}$
- $s_{\text{in}} = ((\ell_{\text{in}}, \ldots, \ell_{\text{in}}), (\varepsilon, \ldots, \varepsilon))$
- $F = \mathsf{Fin} \times \{\varepsilon\}^{\mathsf{DS}}$
- $\Longrightarrow \subseteq \mathsf{States} \times \Gamma \times \mathsf{States}$
  - internal transition $\quad (\overline{\ell}, \overline{z}) \overset{p,a}{\Longrightarrow} (\overline{\ell'}, \overline{z})$
    if $\ell_p \xrightarrow{a}_p \ell'_p$ and $\ell'_q = \ell_q$ for all $q \neq p$,
  - write transition $\quad (\overline{\ell}, \overline{z}) \overset{p,a,d!}{\Longrightarrow} (\overline{\ell'}, \overline{z}')$
    if there is $v \in \mathsf{Val}$:
    $\ell_p \xrightarrow{a,d!v}_p \ell'_p$ and $\ell'_q = \ell_q$ for all $q \neq p$ and
    $z'_d = z_d v$ and $z'_c = z_c$ for all $c \neq d$
  - read transition $\quad (\overline{\ell}, \overline{z}) \overset{p,a,d?}{\Longrightarrow} (\overline{\ell'}, \overline{z}')$
    if there is $v \in \mathsf{Val}$:
    $\ell_p \xrightarrow{a,d?v}_p \ell'_p$ and $\ell'_q = \ell_q$ for all $q \neq p$ and
    $$\left\{ \begin{array}{lll} d \in \mathsf{Stacks}: & z_d = uv \text{ and } z'_d = u \\ d \in \mathsf{Queues}: & z_d = vw \text{ and } z'_d = w \\ d \in \mathsf{Bags}: & z_d = uvw \text{ and } z'_d = uw \end{array} \right\} \text{ for some } u, w \in \mathsf{Val}^*$$
    and $z'_c = z_c$ for all $c \neq d$

We let $L_{\text{op}}(\mathcal{S}) := L(\mathcal{A}_{\mathcal{S}}) \subseteq \Gamma^*$ (discarding the empty word).

Example 2.5. In our client-server system, $L_{\text{op}}(\mathcal{S}_{\text{cs}})$ contains:

- $(p_1, a, c_1!)(p_2, a, c_1?)(p_2, a, c_2!)(p_1, a, c_2?)$
- $(p_1, a, c_1!)(p_1, b, c_1!)(p_2, a, c_1?)(p_2, a, s!)(p_2, b, c_1?)(p_2, b, c_2!)$
  $(p_1, b, c_2?)(p_2, a, s?)(p_2, a, c_2!)(p_1, a, c_2?)$ $\diamondsuit$

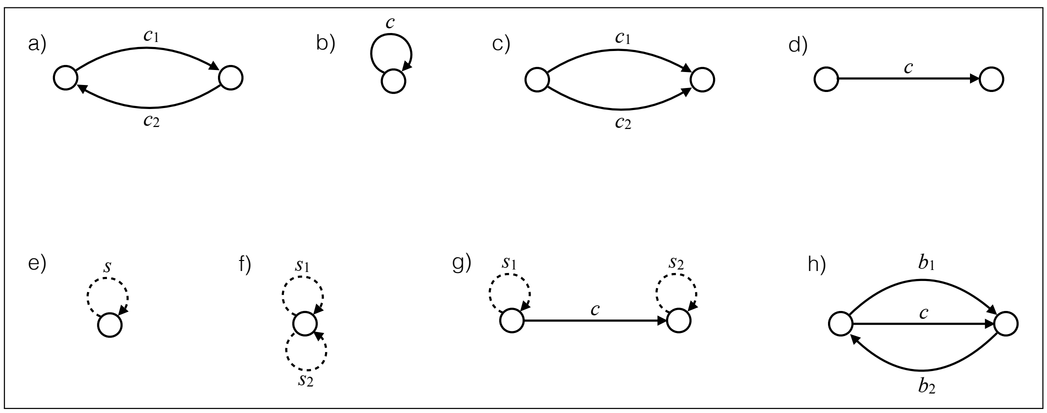
Exercise 2.6. Show that $L_{\text{op}}(\mathcal{S}_{\text{cs}})$ is not regular. $\diamondsuit$

2.2.1 Nonemptiness/Reachability Checking

For an architecture $\mathfrak{A}$ and an alphabet Σ, consider the following problem:

Nonemptiness$(\mathfrak{A}, \Sigma)$:	
Instance:	$\mathcal{S} \in \mathsf{CPDS}(\mathfrak{A}, \Sigma)$
Question:	$L_{\mathrm{op}}(\mathcal{S}) \neq \emptyset\,?$

Theorem 2.7. *Let $\mathfrak{A}$ be any of the following architectures: a, b, c, f, g, h. Then,* Nonemptiness$(\mathfrak{A}, \Sigma)$ *is undecidable.*

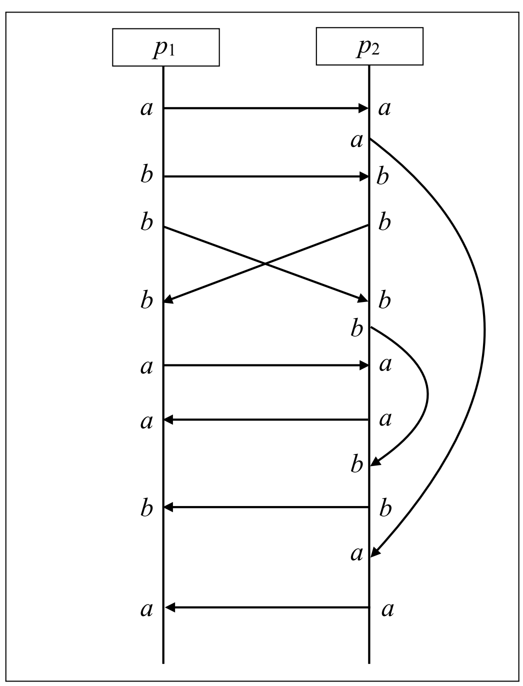


The following table summarizes some special cases:

$\mathfrak{A}$	automata type	CBM	NONEMPTINESS($\mathfrak{A}, \Sigma$)
$\lvert\mathsf{Procs}\rvert = 1$ $\lvert\mathsf{DS}\rvert = 0$	finite automaton	word	decidable
$\lvert\mathsf{Procs}\rvert = \lvert\mathsf{DS}\rvert = 1$ $\mathsf{DS} = \mathsf{Stacks}$	(visibly) pushdown automaton	nested word	decidable
$\lvert\mathsf{Procs}\rvert = 1$ $\lvert\mathsf{DS}\rvert \geq 2$ $\mathsf{DS} = \mathsf{Stacks}$	multi-pushdown automaton	multiply nested word	undecidable
$\mathsf{DS} = \mathsf{Bags}$	$\approx$ Petri net		decidable
$\lvert\mathsf{Procs}\rvert \geq 2$ $\mathsf{DS} = \mathsf{Queues}$ $= (\mathsf{Procs} \times \mathsf{Procs}) \setminus \mathrm{Id}$ where $\mathrm{Id} = \{(p,p) \mid p \in \mathsf{Procs}\}$ $\mathsf{Writer}(p,q) = p$ $\mathsf{Reader}(p,q) = q$	message-passing automaton	message sequence chart (MSC)	undecidable

2.3 Graph Semantics

Example 2.8. Let us represent behaviors as graphs. We start with an example. The following graph will be in the language of $\mathcal{S}_{\mathrm{cs}}$. The source and the target of an edge represent the exchange of a value through some data structure. In the example, their labeling (a or b) is the same. However, this is not always the case.



$\diamondsuit$

Definition 2.9. *A* concurrent behavior with matching *(*CBM*) over* $\mathfrak{A}$ *and* Σ *is a tuple*

$$\mathcal{M} = ((w_p)_{p \in \mathsf{Procs}}, (\rhd^d)_{d \in \mathsf{DS}})$$

- $w_p \in \Sigma^*$ *sequence of actions on process* p

Notation:
$\mathcal{E}_p = \{(p,i) \mid 1 \leq i \leq \lvert w_p \rvert\}$ *events of process* p
$\mathcal{E} = \bigcup_{p \in \mathsf{Procs}} \mathcal{E}_p$
$(p,i) \to (p,i+1)$ *if* $1 \leq i < \lvert w_p \rvert$
for $e = (p,i) \in \mathcal{E}_p$*, let* $\mathsf{pid}(e) = p$ *and* $\lambda(e) \in \Sigma$ *be the* i*-th letter of* w_p

- $\rhd^d \subseteq \mathcal{E}_{\mathsf{Writer}(d)} \times \mathcal{E}_{\mathsf{Reader}(d)}$ *such that:*
  - *if* $e_1 \rhd^d e_2$ *and* $e_3 \rhd^{d'} e_4$ *are different edges* $(d \neq d'$ *or* $(e_1,e_2) \neq (e_3,e_4))$*, then they are disjoint* $(\lvert\{e_1,e_2,e_3,e_4\}\rvert = 4)$
  - $< \; = (\to \cup \rhd)^+ \subseteq \mathcal{E} \times \mathcal{E}$ *is a strict partial order*[1] *where* $\rhd = \bigcup_{d \in \mathsf{DS}} \rhd^d$
  - $\forall d \in \mathsf{Stacks}$ *(LIFO):* $e_1 \rhd^d f_1$ and $e_2 \rhd^d f_2$ and $e_1 < e_2 < f_1 \implies f_2 < f_1$
  - $\forall d \in \mathsf{Queues}$ *(FIFO):* $e_1 \rhd^d f_1$ and $e_2 \rhd^d f_2$ and $e_1 < e_2 \implies f_1 < f_2$

We let $\mathsf{CBM}(\mathfrak{A}, \Sigma)$ *be the set of* CBMs *over* $\mathfrak{A}$ *and* Σ*.* ♢

Run:

Consider a mapping $\rho : \mathcal{E} \to \mathsf{Locs}$.

Define $\rho^- : \mathcal{E} \to \mathsf{Locs}$ by

$$\rho^-(e) = \begin{cases} \rho(e') & \text{if } e' \to e \\ \ell_{\text{in}} & \text{if } e \text{ is minimal on its process} \end{cases}$$

Now, ρ is a *run* of $\mathcal{S}$ on $\mathcal{M}$ if the following hold:

- for all internal events e: $\rho^-(e) \xrightarrow{\lambda(e)}_{\mathsf{pid}(e)} \rho(e)$
- for all $e \rhd^d f$, there is $v \in \mathsf{Val}$ such that:
  - $\rho^-(e) \xrightarrow{\lambda(e), d!v}_{\mathsf{pid}(e)} \rho(e)$ and

[1] For a binary relation R, we let $R^* = \bigcup_{n \geq 0} R^n$ and $R^+ = \bigcup_{n \geq 1} R^n$.

– $\rho^-(f) \xrightarrow{\lambda(f),d?v}_{\mathsf{pid}(f)} \rho(f)$

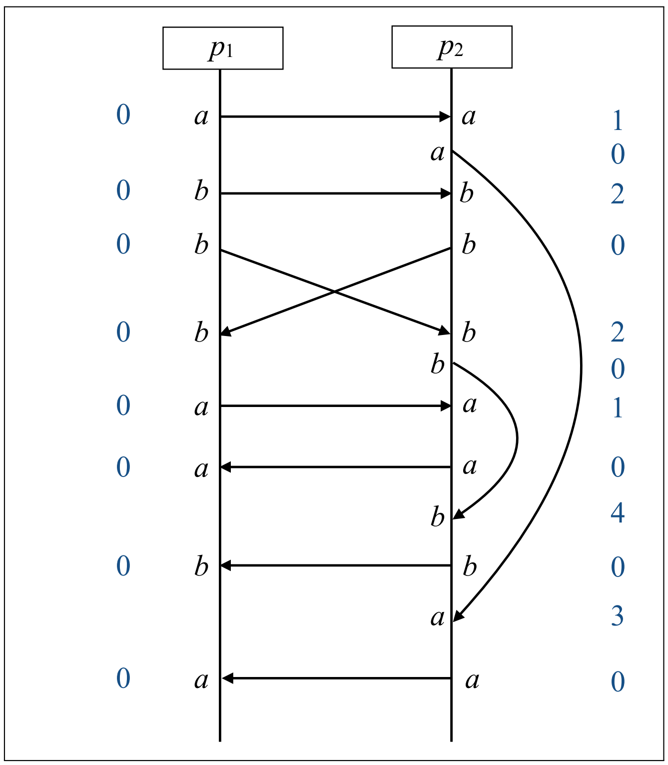
Accepting:

A run ρ is accepting if $(\ell_p)_{p\in\mathsf{Procs}} \in \mathsf{Fin}$ where

$$\ell_p = \begin{cases} \ell_{\text{in}} & \text{if } \mathcal{E}_p = \emptyset \\ \rho((p, |w_p|)) & \text{otherwise} \end{cases}$$

We let $L(\mathcal{S})$ denote the set of CBMs accepted by $\mathcal{S}$.

Example 2.10. The following figure depicts a run of $\mathcal{S}_{\text{cs}}$:



$\diamondsuit$

Relation between operational and graph semantics:

Every CBM $\mathcal{M} = ((w_p)_{p\in\mathsf{Procs}}, (\rhd^d)_{d\in\mathsf{DS}})$ defines a set of words over Γ.

Let $\gamma_{\mathcal{M}} : \mathcal{E} \to \Gamma \quad \big(= (\mathsf{Procs} \times \Sigma) \cup (\mathsf{Procs} \times \Sigma \times \mathsf{DS} \times \{!, ?\})\big)$ be defined by

$$\gamma_{\mathcal{M}}(e) = \begin{cases} (\mathsf{pid}(e), \lambda(e)) & \text{if } e \text{ is internal} \\ (\mathsf{pid}(e), \lambda(e), d!) & \text{if } e \rhd^d f \\ (\mathsf{pid}(e), \lambda(e), d?) & \text{if } f \rhd^d e \end{cases}$$

A *linearization* of $\mathcal{M}$ is any (strict) total order $\sqsubset \subseteq \mathcal{E} \times \mathcal{E}$ such that $< \subseteq \sqsubset$. (recall that $< = (\rightarrow \cup \rhd)^+$).

Suppose $\mathcal{E} = \{e_1, \ldots, e_n\}$ and $e_1 \sqsubset \ldots \sqsubset e_n$.

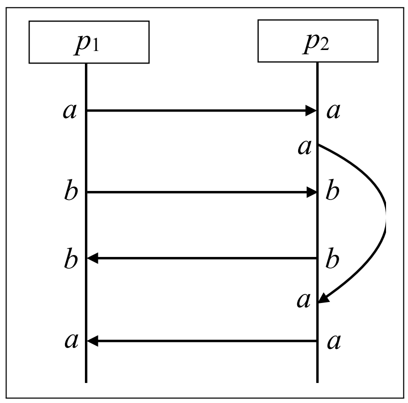
Then, $\sqsubset$ induces the word $\gamma_{\mathcal{M}}(e_1) \ldots \gamma_{\mathcal{M}}(e_n) \in \Gamma^*$.

Let $\mathsf{Lin}(\mathcal{M}) \subseteq \Gamma^*$ be the set of words that are induced by the linearisations of $\mathcal{M}$.

Remark 2.11.

- If $\mathsf{Bags} = \emptyset$, then for every $w \in \Gamma^*$, there is at most one $\mathcal{M} \in \mathsf{CBM}(\mathfrak{A}, \Sigma)$ such that $w \in \mathsf{Lin}(\mathcal{M})$.
- If $\mathsf{Bags} = \{d\}$, this is not the case: $(p, a, d!)(p, a, d!)(p, a, d?)(p, a, d?)$ is a linearization of two different CBMs.

Example 2.12. Let $\mathcal{M}$ be the following CBM.



Then, $\mathsf{Lin}(\mathcal{M})$ contains:

- $(p_1, a, c_1!)(p_1, b, c_1!)$
  $(p_2, a, c_1?)(p_2, a, s!)(p_2, b, c_1?)(p_2, b, c_2!)(p_2, a, s?)(p_2, a, c_2!)$
  $(p_1, b, c_2?)(p_1, a, c_2?)$
- $(p_1, a, c_1!)(p_2, a, c_1?)(p_2, a, s!)(p_1, b, c_1!)$
  $(p_2, b, c_1?)(p_2, b, c_2!)(p_2, a, s?)(p_2, a, c_2!)$
  $(p_1, b, c_2?)(p_1, a, c_2?)$

Actually, $\mathcal{M}$ has 9 linearizations. $\diamondsuit$

Theorem 2.13. *For all $\mathcal{S} \in \mathsf{CPDS}(\mathfrak{A}, \Sigma)$, we have $\mathsf{Lin}(L(\mathcal{S})) = L_{\mathrm{op}}(\mathcal{S})$.*

Without proof.

CHAPTER 3

Monadic Second-Order Logic

3.1 Monadic Second-Order Logic

Example: $\forall x(a(x) \Rightarrow \exists y(x \rhd y \wedge b(y)))$

Syntax:

Let $\mathrm{Var} = \{x, y, \ldots\}$ be an infinite set of first-order variables.

Let $\mathrm{VAR} = \{X, Y, \ldots\}$ be an infinite set of second-order variables.

The set $\mathsf{MSO}(\mathfrak{A}, \Sigma)$ of formulas from *monadic second-order logic* is given by the grammar:

$$\varphi ::= a(x) \mid p(x) \mid x = y \mid x \rhd^d y \mid x \to y \mid x \in X \mid \varphi \vee \varphi \mid \neg\varphi \mid \exists x\, \varphi \mid \exists X\, \varphi$$

where $x, y \in \mathrm{Var}$, $X \in \mathrm{VAR}$, $a \in \Sigma$, $p \in \mathsf{Procs}$, $d \in \mathsf{DS}$.

The fragment $\mathsf{EMSO}(\mathfrak{A}, \Sigma)$ consists of the formulas of the form $\exists X_1 \ldots \exists X_n \varphi$ where φ is a first-order formula, i.e., it does not contain any second-order quantification.

Semantics:

Let $\mathcal{M} = ((w_p)_{p \in \mathsf{Procs}}, (\rhd^d)_{d \in \mathsf{DS}})$ be a CBM. An $\mathcal{M}$-interpretation is a function $\mathcal{I}$ that maps every

- $x \in \mathrm{Var}$ to some element of $\mathcal{E}$
- $X \in \mathrm{VAR}$ to some subset of $\mathcal{E}$

Satisfaction $\mathcal{M} \models_{\mathcal{I}} \varphi$ is defined inductively as follows:

- $\mathcal{M} \models_{\mathcal{I}} a(x)$ if $\lambda(\mathcal{I}(x)) = a$
- $\mathcal{M} \models_{\mathcal{I}} p(x)$ if $\mathsf{pid}(\mathcal{I}(x)) = p$
- $\mathcal{M} \models_{\mathcal{I}} x = y$ if $\mathcal{I}(x) = \mathcal{I}(y)$
- $\mathcal{M} \models_{\mathcal{I}} x \triangleright^d y$ if $\mathcal{I}(x) \triangleright^d \mathcal{I}(y)$
- $\mathcal{M} \models_{\mathcal{I}} x \to y$ if $\mathcal{I}(x) \to \mathcal{I}(y)$
- $\mathcal{M} \models_{\mathcal{I}} x \in X$ if $\mathcal{I}(x) \in \mathcal{I}(X)$
- $\mathcal{M} \models_{\mathcal{I}} \varphi \vee \psi$ if $\mathcal{M} \models_{\mathcal{I}} \varphi$ or $\mathcal{M} \models_{\mathcal{I}} \psi$
- $\mathcal{M} \models_{\mathcal{I}} \neg\varphi$ if $\mathcal{M} \not\models_{\mathcal{I}} \varphi$
- $\mathcal{M} \models_{\mathcal{I}} \exists x\varphi$ if there is $e \in \mathcal{E}$ such that $\mathcal{M} \models_{\mathcal{I}[x \mapsto e]} \varphi$
- $\mathcal{M} \models_{\mathcal{I}} \exists X\varphi$ if there is $E \subseteq \mathcal{E}$ such that $\mathcal{M} \models_{\mathcal{I}[X \mapsto E]} \varphi$

Here, $\mathcal{I}[x \mapsto e]$ maps x to e and coincides with $\mathcal{I}$ on $(\mathrm{Var} \setminus \{x\}) \cup \mathrm{VAR}$.
When φ is a sentence, then $\mathcal{I}$ is irrelevant, and we write $\mathcal{M} \models \varphi$ instead of $\mathcal{M} \models_{\mathcal{I}} \varphi$.
We let $L(\varphi) := \{\mathcal{M} \in \mathsf{CBM}(\mathfrak{A}, \Sigma) \mid \mathcal{M} \models \varphi\}$.

Example 3.1. We use the following abbreviations:

- $\varphi \wedge \psi \;=\; \neg(\neg\varphi \vee \neg\psi) \qquad \forall x\varphi \;=\; \neg\exists x\neg\varphi \qquad \varphi \Rightarrow \psi \;=\; \neg\varphi \vee \psi$
- $x \triangleright y = \bigvee_{d \in \mathsf{DS}}(x \triangleright^d y)$
- $\mathit{write}(x) = \exists y(x \triangleright y) \qquad \mathit{read}(x) = \exists y(y \triangleright x)$
- $\mathit{local}(x) = \neg\mathit{write}(x) \wedge \neg\mathit{read}(x)$
- $\mathit{min}(x) = \neg\exists y(y \to x) \qquad \mathit{max}(x) = \neg\exists y(x \to y)$
- "$\mathcal{E}_p = \emptyset$" $= \neg\exists x\, p(x)$
- $x \leq y = \forall X(x \in X \wedge \forall z \forall z'((z \in X \wedge (z \to z' \vee z \triangleright z')) \Rightarrow z' \in X) \Rightarrow y \in X)$
- On CBMs, the latter formula is equivalent to

$$\begin{array}{lll} \exists X\ [& & x \in X \\ & \wedge & y \in X \\ & \wedge & \forall z \in X : \\ & & \quad z = y \vee (\ \exists z' \in X : z \to z' \vee z \triangleright z'\ )\ )\ ] \end{array}$$

$\diamondsuit$

Example 3.2. We consider some formulas for $\mathcal{S}_{\text{cs}}$:

- $\varphi_1 = \forall x(a(x) \Rightarrow \exists y(x \leq y \wedge b(y)))$
- $\mathit{req\text{-}ack}(x, y) = p_1(x) \wedge \left(\begin{array}{ll} & \exists x_1, x_2(x \rhd^{c_1} x_1 \rightarrow x_2 \rhd^{c_2} y) \\ \vee & \exists x_1, \ldots, x_4(x \rhd^{c_1} x_1 \rightarrow x_2 \rhd^{s} x_3 \rightarrow x_4 \rhd^{c_2} y) \end{array}\right)$
- $\varphi_2 = \forall x, y\Big(\mathit{req\text{-}ack}(x, y) \Rightarrow \big((a(x) \wedge a(y)) \vee (b(x) \wedge b(y))\big)\Big)$

For the client-server system $\mathcal{S}_{\text{cs}}$ from the previous chapter, we have $L(\mathcal{S}_{\text{cs}}) \not\subseteq L(\varphi_1)$ and $L(\mathcal{S}_{\text{cs}}) \subseteq L(\varphi_2)$. $\diamondsuit$

3.2 Expressive Power of MSO Logic

Recall a theorem from the sequential case:

Theorem 3.3 (Büchi-Elgot-Trakhtenbrot [Büc60, Elg61, Tra62]). *Suppose $|\mathsf{Procs}| = 1$ and $\mathsf{DS} = \emptyset$. Let $L \subseteq \mathsf{CBM}(\mathfrak{A}, \Sigma)$, which can be seen as a word language $L \subseteq \Sigma^*$. Then, the following are equivalent:*

- *There is $\mathcal{S} \in \mathsf{CPDS}(\mathfrak{A}, \Sigma)$ such that $L(\mathcal{S}) = L$.*
- *There is a sentence $\varphi \in \mathsf{MSO}(\mathfrak{A}, \Sigma)$ such that $L(\varphi) = L$.*

The theorem also holds for $|\mathsf{Procs}| = 1$, $|\mathsf{DS}| = 1$, and $\mathsf{DS} = \mathsf{Stacks}$ [AM09].

One direction is actually independent of achitecture: 3rd Lecture

Theorem 3.4. *For every $\mathcal{S} \in \mathsf{CPDS}(\mathfrak{A}, \Sigma)$, there is a sentence $\Phi_{\mathcal{S}} \in \mathsf{EMSO}(\mathfrak{A}, \Sigma)$ such that $L(\Phi_{\mathcal{S}}) = L(\mathcal{S})$.*

Proof. Fix $\mathcal{S} = (\mathsf{Locs}, \mathsf{Val}, (\rightarrow_p)_{p\in\mathsf{Procs}}, \ell_{\mathrm{in}}, \mathsf{Fin}) \in \mathsf{CPDS}(\mathfrak{A}, \Sigma)$. We let $\mathsf{Trans} = \bigcup_{p\in\mathsf{Procs}} \rightarrow_p$. For a transition $t \in \mathsf{Trans}$, we let $\mathsf{src}(t)$, resp. $\mathsf{tgt}(t)$, be the source, resp. target, location of t. For a data structure $d \in \mathsf{DS}$ and a value $v \in \mathsf{Val}$, we let $\mathsf{Send}(d, v)$, resp. $\mathsf{Receive}(d, v)$, be the set of transitions of the form $\ell \xrightarrow{a,d!v}_p \ell'$ where $p = \mathsf{Writer}(d)$, resp. $\ell \xrightarrow{a,d?v}_p \ell'$ where $p = \mathsf{Reader}(d)$. Finally, for $p \in \mathsf{Procs}$ and $a \in \Sigma$ we let $\mathsf{Trans}(p, a)$ be the set of transisions on process p with label a. We define

$$
\begin{array}{rcl}
\Phi_{\mathcal{S}} & = & \exists (X_t)_{t\in\mathsf{Trans}} \Big[\mathit{Partition}((X_t)_{t\in\mathsf{Trans}}) \\
& & \wedge \quad \forall x \bigwedge_{p\in\mathsf{Procs}, a\in\Sigma} \big(p(x) \wedge a(x) \Rightarrow \bigvee_{t\in\mathsf{Trans}(p,a)} X_t(x)\big) \\
& & \wedge \quad \forall x, x' \big(x \rightarrow x' \Rightarrow \bigvee_{t,t'\in\mathsf{Trans} \mid \mathsf{tgt}(t)=\mathsf{src}(t')} X_t(x) \wedge X_{t'}(x')\big) \\
& & \wedge \quad \forall x, x' \bigwedge_{d\in\mathsf{DS}} \big(x \rhd^d x' \Rightarrow \bigvee_{v\in\mathsf{Val}, t\in\mathsf{Send}(d,v), t'\in\mathsf{Receive}(d,v)} X_t(x) \wedge X_{t'}(x')\big) \\
& & \wedge \quad \forall x \big(\mathit{min}(x) \Rightarrow \bigvee_{t\in\mathsf{Trans} \mid \mathsf{src(t)}=\ell_{\mathrm{in}}} X_t(x)\big) \\
& & \wedge \quad \bigvee_{\substack{(\ell_p)_{p\in\mathsf{Procs}}\in\mathsf{Fin}, P\subseteq\mathsf{Procs} \\ \mid\ P\subseteq\{p\in\mathsf{Procs} \mid \ell_p=\ell_{\mathrm{in}}\}}} \left(\begin{array}{l} \bigwedge_{p\notin P} \ \exists x\,(p(x) \wedge \mathit{max}(x) \wedge \bigvee_{t \mid \mathsf{tgt(t)}=\ell_{\mathsf{p}}} X_t(x)) \\ \wedge \bigwedge_{p\in P} \ \neg\exists x\, p(x) \end{array} \right) \Big]
\end{array}
$$

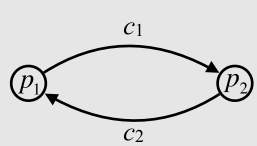
where

$$
\mathit{Partition}((X_t)_{t\in\mathsf{Trans}}) \; = \; \forall x \bigvee_{t\in\mathsf{Trans}} \Big(X_t(x) \quad \wedge \quad \bigwedge_{t'\neq t} \neg X_{t'}(x)\Big)
$$

This completes the construction of the formula $\Phi_{\mathcal{S}}$. We have $L(\Phi_{\mathcal{S}}) = L(\mathcal{S})$. ∎

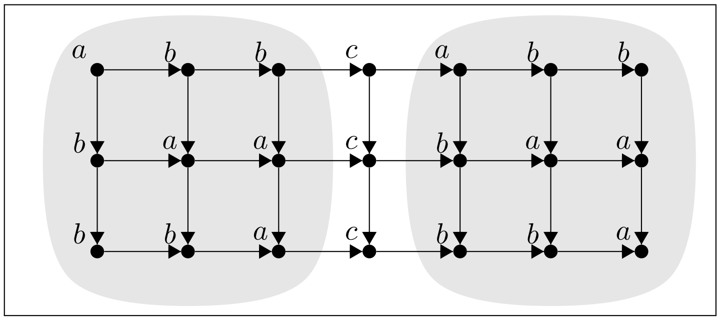
Unfortunately, the other direction does not hold in general:

Theorem 3.5. *Suppose $\Sigma = \{a, b, c\}$. Suppose that $\mathfrak{A}$ is given by $\mathsf{Procs} = \{p_1, p_2\}$ and $\mathsf{DS} = \mathsf{Queues} = \{c_1, c_2\}$ with $\mathsf{Writer}(c_1) = \mathsf{Reader}(c_2) = p_1$ and $\mathsf{Writer}(c_2) = \mathsf{Reader}(c_1) = p_2$:*



There is a sentence $\varphi \in \mathsf{MSO}(\mathfrak{A}, \Sigma)$ such that, for all $\mathcal{S} \in \mathsf{CPDS}(\mathfrak{A}, \Sigma)$, we have $L(\mathcal{S}) \neq L(\varphi)$.

Proof. To illustrate the proof idea, which goes back to [Tho96] we consider pictures over $\Sigma = \{a, b, c\}$. Here is an example:



Consider the set $P_=$ of pictures that are of the form ACA where

- A is a nonempty square picture with labels in $\{a, b\}$, and
- C is a c-labeled column.

The above picture is a member of $P_=$.

The language $P_=$ is definable by an MSO formula $\Phi_=$ over pictures using predicates:

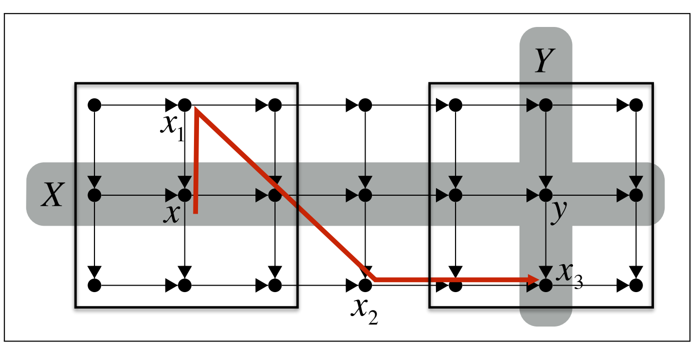
$$\mathsf{go\text{-}right}(x, y) \qquad\qquad \mathsf{go\text{-}down}(x, y)$$

The formula $\Phi_=$ is easy to obtain once we have a "matching" predicate $\mu(x, y)$ that relates two coordinates x and y iff they refer to identical positions in the two different square grids.

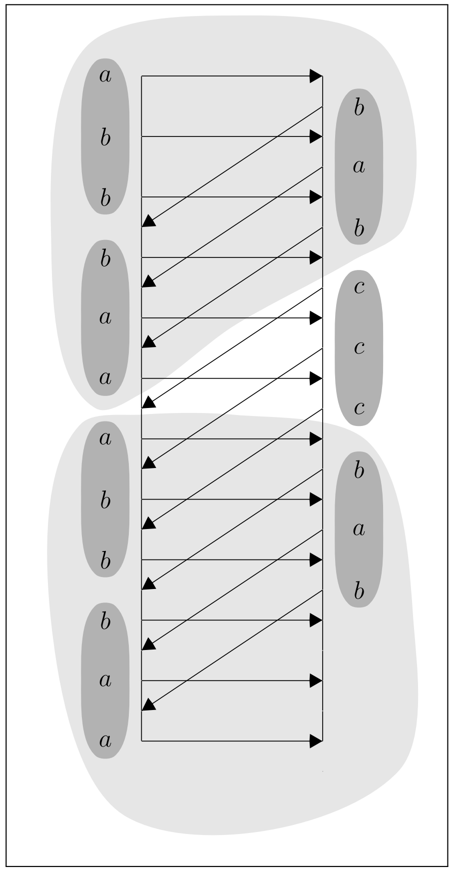
Essentially, $\mu(x, y)$ says that there are sets X, Y and nodes x_1, x_2, x_3 such that

- X contains all those nodes that are in the same row as x,
- x_1 is the topmost node on the same column as x,
- x_2 is on the intersection of the bottom row and the right-down diagonal starting from x_1,
- x_3 is reached from x_2 by going two steps to the right,
- Y contains all those nodes that are in the same column as x_3.

Then, we have $\mu(x, y)$ iff $y \in X \cap Y$. The idea for $\mu(x, y)$ is illustrated below:



Next, we encode pictures into CBMs over $\mathfrak{A}$ and Σ. The above picture is encoded as follows:



We obtain a formula $\widetilde{\Phi_=} \in \mathsf{MSO}(\mathfrak{A}, \Sigma)$ for the encodings of the above picture language $P_=$ inductively:

- $\widetilde{\exists x \varphi} \;=\; \exists x(\mathit{write}(x) \wedge \widetilde{\varphi})$
- $\widetilde{\mathsf{go\text{-}right}}(x, y) \;=\; \exists z(x \rhd z \to y)$
- $\widetilde{\mathsf{go\text{-}down}}(x, y) \;=\; \neg \mathit{bottom}(x) \wedge (x \to y \vee \exists z(x \to z \to y \wedge \neg \mathit{write}(z)))$

Here, $\mathit{bottom}(x)$ says that x is an element that is located on the last row (left as an exercise). Other formulas remain unchanged.

Using Theorem 3.4, we can moreover determine a formula ψ_{pict} that describes the encodings of (arbitrary) pictures.

Let $\varphi = \psi_{\text{pict}} \wedge \widetilde{\Phi_=}$.

Towards a contradiction, suppose that there is $\mathcal{S} = (\mathsf{Locs}, \mathsf{Val}, (\rightarrow_p)_{p \in \mathsf{Procs}}, \ell_{\text{in}}, \mathsf{Fin}) \in \mathsf{CPDS}(\mathfrak{A}, \Sigma)$ such that $L(\mathcal{S}) = L(\varphi)$.

An accepting run of $\mathcal{S}$ has to transfer all the information it has about the upper part of the CBM along the middle part of size $2n$ (where n is the length of a column), to the lower part.

However, there are

- 2^{n^2} square pictures of width/height n, and
- $|\mathsf{Locs}|^{2n}$-many assignments of states to the middle part.

Thus, for sufficiently large n, we can find an accepting run of $\mathcal{S}$ on a CBM $\mathcal{M}$ whose upper part and lower part do not match, i.e., $\mathcal{M} \notin L(\varphi)$. ∎

However, there is a fragment of MSO that allows for a positive result (we do not present the proof).

Theorem 3.6 ([BL06]). *Suppose* $\mathsf{DS} = \mathsf{Queues}$*. Then, for every sentence* $\varphi \in \mathsf{EMSO}(\mathfrak{A}, \Sigma)$*, there is a CPDS* $\mathcal{S}$ *such that* $L(\mathcal{S}) = L(\varphi)$*.*

Exercise 3.7. Prove that CPDSs are, in general, not closed under complementation: Suppose $\Sigma = \{a, b, c\}$ and assume the architecture $\mathfrak{A}$ from Theorem 3.5. Show that there is $\mathcal{S} \in \mathsf{CPDS}(\mathfrak{A}, \Sigma)$ such that, for all $\mathcal{S}' \in \mathsf{CPDS}(\mathfrak{A}, \Sigma)$, we have $L(\mathcal{S}') \neq \mathsf{CBM}(\mathfrak{A}, \Sigma) \setminus L(\mathcal{S})$. ◇

Exercise 3.8. Show that Theorem 3.5 also holds when $|\mathsf{Procs}| = 1$, $|\mathsf{DS}| = 2$, and $\mathsf{DS} = \mathsf{Stacks}$. ◇

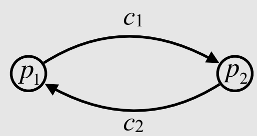
3.3 Satisfiability and Model Checking

For an architecture $\mathfrak{A}$ and an alphabet Σ, consider the following problems:

MSO-Satisfiability($\mathfrak{A}, \Sigma$):	
Instance:	$\varphi \in \mathsf{MSO}(\mathfrak{A}, \Sigma)$
Question:	$L(\varphi) \neq \emptyset$?

MSO-ModelChecking($\mathfrak{A}, \Sigma$):	
Instance:	$\mathcal{S} \in \mathsf{CPDS}(\mathfrak{A}, \Sigma)$; $\varphi \in \mathsf{MSO}(\mathfrak{A}, \Sigma)$
Question:	$L(\mathcal{S}) \subseteq L(\varphi)$?

Theorem 3.9. *Let $\mathfrak{A}$ be given as follows (and Σ be arbitrary):*



Then, all the abovementioned problems are undecidable.

CHAPTER 4

Underapproximate Verification

4th Lecture

Recall that most verification problems such as nonemptiness, global-state reachability, and model checking are undecidable even for very simple architectures.

4.1 Principles of Underapproximate Verification

To get decidability, we will restrict decision problems to a subclass $\mathcal{C} \subseteq \mathsf{CBM}(\mathfrak{A}, \Sigma)$ of CBMs:

MSO-Validity($\mathfrak{A}, \Sigma, \mathcal{C}$):	
Instance:	$\varphi \in \mathsf{MSO}(\mathfrak{A}, \Sigma)$
Question:	$\mathcal{C} \subseteq L(\varphi)$?

MSO-ModelChecking($\mathfrak{A}, \Sigma, \mathcal{C}$):	
Instance:	$\mathcal{S} \in \mathsf{CPDS}(\mathfrak{A}, \Sigma)$; $\varphi \in \mathsf{MSO}(\mathfrak{A}, \Sigma)$
Question:	$L(\mathcal{S}) \cap \mathcal{C} \subseteq L(\varphi)$?

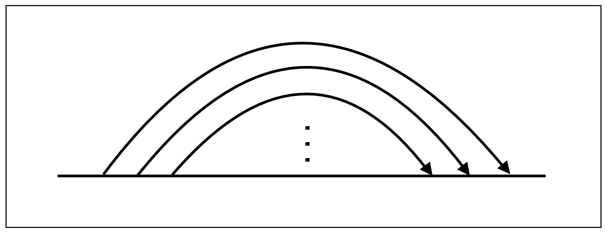
For example, we may only consider the CBMs that can be executed when the data structures have bounded capacity:

Definition 4.1. *Let $k \geq 0$. A CBM $\mathcal{M}$ is called k-*existentially *bounded (k-$\exists$B for short) if there is a linearization $w \in \mathsf{Lin}(\mathcal{M})$ such that, for every prefix u of w, the number of unmatched writes in u is at most k. A class $\mathcal{C} \subseteq \mathsf{CBM}(\mathfrak{A}, \Sigma)$ is k-$\exists$B if $\mathcal{M}$ is k-$\exists$B for every $\mathcal{M} \in \mathcal{C}$. Finally, $\mathcal{C}$ is called $\exists$B if it is k-$\exists$B for some k.* $\diamondsuit$

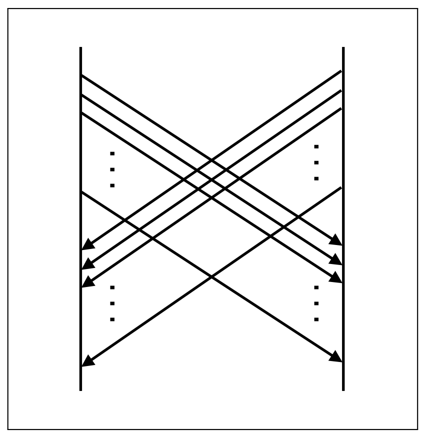
Example 4.2. We will give some examples:

(a) The CBM from Example 2.8 is 3-∃B.

(b) The class of nested words ($|\mathsf{Procs}| = 1$, $|\mathsf{DS}| = 1$, and $\mathsf{DS} = \mathsf{Stacks}$) is not ∃B, as illustrated by the following figure:



(c) The class of MSCs ($|\mathsf{Procs}| \geq 2$, $\mathsf{DS} = \mathsf{Queues} = \mathsf{Procs} \times \mathsf{Procs} \setminus \{(p,p) \mid p \in \mathsf{Procs}\}$, $\mathsf{Writer}(p,q) = p$, and $\mathsf{Reader}(p,q) = q$) is not ∃B:



◇

Exercise 4.3. Consider the encoding of pictures as CBMs from Section 3.2. Show that the encoding of a picture of height k yields a CBM that is k-∃B. ◇

We are looking for "reasonable" classes of CBMs that are suitable for underapproximate verification.

Definition 4.4. *Let $\mathcal{C} = (\mathcal{C}_k)_{k \geq 0}$ with $\mathcal{C}_k \subseteq \mathsf{CBM}(\mathfrak{A}, \Sigma)$ be a family of classes of* CBMs*. Then, $\mathcal{C}$ is called*

- monotone *if $\mathcal{C}_k \subseteq \mathcal{C}_{k+1}$ for all $k \geq 0$,*
- complete *if $\bigcup_{k \geq 0} \mathcal{C}_k = \mathsf{CBM}(\mathfrak{A}, \Sigma)$,*
- decidable *if the usual decision problems are decidable when the domain of* CBMs *is restricted to $\mathcal{C}_k$,*
- MSO-definable *if, for all $k \geq 0$, there is a sentence $\varphi_k \in \mathsf{MSO}(\mathfrak{A}, \Sigma)$ such that $L(\varphi_k) = \mathcal{C}_k$, and*

- CPDS-definable *if, for all $k \geq 0$, there is a CPDS $\mathcal{S}_k \in \mathsf{CPDS}(\mathfrak{A}, \Sigma)$ such that $L(\mathcal{S}_k) = \mathcal{C}_k$.* ◇

Below, we first present a generic family, which is based on the notion of *special tree-width.*

4.2 Graph-Theoretic Approach

In the following, we will use tools from graph theory. Actually, the pair $(\mathfrak{A}, \Sigma)$ defines a *signature* of unary and binary relation symbols. Thus, one can consider general graphs over $(\mathfrak{A}, \Sigma)$, with node labels from Σ and edge labels from $\Gamma = \{\mathsf{proc}\} \cup \mathsf{DS}$. Here, proc stands for *process successor*, and $d \in \mathsf{DS}$ is the labeling of an edge that connects a write and a read event. Those graphs that satisfy the axioms from Definition 2.9 can then be considered as CBMs.

Proposition 4.5. Let $\mathfrak{A}$ be an architecture. The class $\mathsf{CBM}(\mathfrak{A}, \Sigma)$ is MSO-definable, i.e., there is an $\mathsf{MSO}(\mathfrak{A}, \Sigma)$ sentence Φ_{cbm} such that for all (Σ, Γ)-labeled graphs G we have $G \models \Phi_{\mathsf{cbm}}$ iff $G \in \mathsf{CBM}(\mathfrak{A}, \Sigma)$.

Proof. Let $G = (\mathcal{E}, \rightarrow, (\rhd^d)_{d \in \mathsf{DS}}, \mathsf{pid}, \lambda)$ be a (Σ, Γ)-labelled graph. The formula Φ_{cbm} has to check that all conditions of Definition 2.9 are satisfied.

This is left as an exercise. ■

Proposition 4.6. Fix a class $\mathcal{C} \subseteq \mathsf{CBM}(\mathfrak{A}, \Sigma)$. The following problems are inter-reducible:

1. MSO-Validity$(\mathfrak{A}, \Sigma, \mathcal{C})$
2. MSO-ModelChecking$(\mathfrak{A}, \Sigma, \mathcal{C})$

Proof. For the reduction from 1. to 2., let $\varphi \in \mathsf{MSO}(\mathfrak{A}, \Sigma)$ be a sentence. Let $\mathcal{S}_{\mathsf{univ}}$ be the *universal* CPDS, satisfying $L(\mathcal{S}_{\mathsf{univ}}) = \mathsf{CBM}(\mathfrak{A}, \Sigma)$. Note that we can define $\mathcal{S}_{\mathsf{univ}}$ such that $|\mathsf{Locs}| = 1 = |\mathsf{Val}|$ and where we have full transition tables. We have:

$$\mathcal{C} \subseteq L(\varphi) \quad \text{iff} \quad L(\mathcal{S}_{\mathsf{univ}}) \cap \mathcal{C} \subseteq L(\varphi)$$

For the reduction from 2. to 1., let $\mathcal{S} \in \mathsf{CPDS}(\mathfrak{A}, \Sigma)$ and $\varphi \in \mathsf{MSO}(\mathfrak{A}, \Sigma)$. Let $\varphi_{\mathcal{S}} \in \mathsf{MSO}(\mathfrak{A}, \Sigma)$ such that $L(\varphi_{\mathcal{S}}) = L(\mathcal{S})$ (cf. Theorem 3.4). We have:

$$\begin{array}{ll} & L(\mathcal{S}) \cap \mathcal{C} \subseteq L(\varphi) \\ \text{iff} & L(\varphi_{\mathcal{S}}) \cap \mathcal{C} \subseteq L(\varphi) \\ \text{iff} & \mathcal{C} \subseteq L(\varphi \vee \neg\varphi_{\mathcal{S}}) \end{array}$$ ■

The decidability of the MSO theory of classes of graphs has been extensively studied (cf. the book by Courcelle and Engelfriet [CE12]):

Theorem 4.7. *Let $\mathcal{C}$ be a class of bounded degree graphs which is MSO-definable. The following statements are equivalent:*

1. *$\mathcal{C}$ has a decidable MSO theory.*
2. *$\mathcal{C}$ can be interpreted in binary trees.*
3. *$\mathcal{C}$ has bounded tree-width.*
4. *$\mathcal{C}$ has bounded clique-width.*

For a class $\mathcal{C} \subseteq \mathsf{CBM}(\mathfrak{A}, \Sigma)$ that is MSO-definable, we prove *bounded special tree-width*

- to get decidability,
- to get the interpretation in binary trees,
- to reduce verification problems to problems on tree automata, and
- to get efficient algorithms with optimal complexity.

In the theorem above, graphs are interpreted in binary trees. We need to identify which trees are *valid encodings*, i.e., do encode graphs in the class $\mathcal{C}$. This is why we assumed the class of graphs to be MSO-definable. From this, we can build a tree automaton for the valid encodings. Actually, we can replace MSO-definability of the class $\mathcal{C}$ by the existence of a tree automaton $\mathcal{A}_{\mathcal{C}}$ for the *valid encodings* of CBMs in $\mathcal{C}$. It is often better to define the tree automaton directly. Its size has a direct impact on the decision procedures arising from the tree-interpretation.

4.3 Graph (De)composition and Tree Interpretation

Note that Subsection 4.3 is not part of this year's lecture.

We will illustrate the concept of tree interpretation by means of *cographs*. Undirected and labeled cographs are generated by *cograph terms*. A cograph term is built from the grammar (cograph algebra)

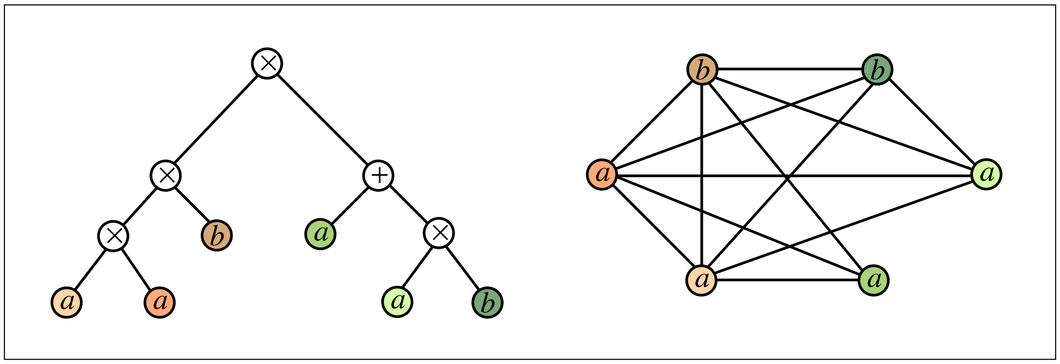
$$C \ ::= \ a \ \mid \ C \oplus C \ \mid \ C \otimes C$$

where $a \in \Sigma$. A term C defines a cograph $[\![C]\!] = (V, E, \lambda)$ as follows:

- $[\![a]\!]$ is the graph $(\{1\}, \emptyset, 1 \mapsto a)$ with one a-labeled vertex and no edges
- if $[\![C_i]\!] = (V_i, E_i, \lambda_i)$ for $i = 1, 2$, with $V_1 \cap V_2 = \emptyset$, then
$$[\![C_1 \oplus C_2]\!] = (V_1 \cup V_2, E_1 \cup E_2, \lambda_1 \cup \lambda_2)$$
- if $[\![C_i]\!] = (V_i, E_i, \lambda_i)$ for $i = 1, 2$, with $V_1 \cap V_2 = \emptyset$, then
$$[\![C_1 \otimes C_2]\!] = (V_1 \cup V_2, E_1 \cup E_2 \cup \underbrace{(V_1 \times V_2) \cup (V_2 \times V_1)}_{\text{undirected graphs}}, \lambda_1 \cup \lambda_2)$$

  (this is called the *complete join*)

Example 4.8. Consider the cograph term $C = ((a \otimes a) \otimes b) \otimes (a \oplus (a \otimes b))$. The figure below shows a tree representation of C as well as the cograph $[\![C]\!]$ defined by C.



Note that the tree representation offers a top-down decomposition of a cograph. $\diamondsuit$

Remark 4.9. One can also include edge labelings d, using operators $\otimes_d$ with the expected meaning.

Clearly, the set of cograph terms (considered as trees) is a regular tree language:

Lemma 4.10. The set of cograph terms is MSO-definable in binary trees. Alternatively, there is a tree automaton which accepts the set of binary trees that are cograph terms.

Proof. The set of "valid" binary trees is defined by the sentence

$$\varphi_{\text{cograph}} = \forall x \left(\begin{array}{cc} & \mathit{leaf}(x) \to \bigvee_{a \in \Sigma} a(x) \\ \wedge & \neg \mathit{leaf}(x) \to \oplus(x) \vee \otimes(x) \end{array} \right).$$

∎

Tree interpretation:

Next, we demonstrate that a cograph can be *recovered* from its cograph term using MSO formulas, i.e., an *MSO interpretation.* Let C be a cograph term (i.e., a binary tree), and let $G = [\![C]\!] = (V, E, \lambda)$.

- The nodes in V correspond to the leaves of C:

$$\varphi_{\mathsf{vertex}}(x) = \mathit{leaf}(x) = \neg \exists y (x \downarrow_0 y \ \vee \ x \downarrow_1 y)$$

where $\downarrow_0$ stands for "left child" and $\downarrow_1$ for "right child".

- The set E of edges is defined by

$$\begin{aligned} & \varphi_{\mathsf{edge}}(x, y) \\ = \ & \text{"}\mathit{least\text{-}common\text{-}ancestor}(x, y) \text{ is labeled } \otimes \text{"} \\ = \ & \exists z, x', y' \left(\begin{array}{cc} & z \downarrow_0 x' \ \wedge \ z \downarrow_1 y' \\ \vee & z \downarrow_1 x' \ \wedge \ z \downarrow_0 y' \end{array} \right) \wedge (x' \downarrow^* x) \wedge (y' \downarrow^* y) \wedge \otimes(z) \end{aligned}$$

Proposition 4.11. MSO formulas over cographs can be "translated" to MSO formulas over cograph terms: For all sentences $\varphi \in \mathsf{MSO}(\Sigma, E)$, there is a sentence $\widetilde{\varphi} \in \mathsf{MSO}(\Sigma \cup \{\oplus, \otimes\}, \downarrow_0, \downarrow_1)$ such that, for every cograph term C, say with $G = [\![C]\!]$, we have

$$G \models \varphi \quad \text{iff} \quad C \models \widetilde{\varphi}\,.$$

Proof. We proceed by induction on φ:

- $\widetilde{a(x)} = a(x)$
- $\widetilde{xEy} = \varphi_{\mathsf{edge}}(x, y)$
- $\widetilde{x \in X} = x \in X$
- $\widetilde{x = y} = (x = y)$
- $\widetilde{\neg\varphi} = \neg\,\widetilde{\varphi}$
- $\widetilde{\varphi_1 \vee \varphi_2} = \widetilde{\varphi_1} \vee \widetilde{\varphi_2}$
- $\widetilde{\exists x\varphi} = \exists x(\varphi_{\text{vertex}}(x) \wedge \widetilde{\varphi})$
- $\widetilde{\exists X\varphi} = \exists X((\forall x(x \in X \to \varphi_{\text{vertex}}(x))) \wedge \widetilde{\varphi})$

Note that, in the correctness proof, we have to deal with free variables. Actually, the inductive statement is as follows: For all $\varphi \in \mathsf{MSO}(\Sigma, E)$, there is $\widetilde{\varphi} \in \mathsf{MSO}(\Sigma \cup \{\oplus, \otimes\}, \downarrow_0, \downarrow_1)$ such that, for every cograph term C and every interpretation $\mathcal{I}$ of $\mathrm{Var} \cup \mathrm{VAR}$ in $\mathit{Leaves}(C) = \mathit{Vertices}(G = [\![C]\!])$, we have $G \models_{\mathcal{I}} \varphi$ iff $C \models_{\mathcal{I}} \widetilde{\varphi}$. ■

Corollary 4.12. The MSO theory of cographs is decidable.

Proof. Let $\varphi \in \mathsf{MSO}(\Sigma, E)$. Then, φ is valid on cographs iff $\varphi_{\text{cograph}} \to \widetilde{\varphi}$ is valid on binary trees (cf. Lemma 4.10). Note that, moreover, φ is satisfiable on cographs iff $\varphi_{\text{cograph}} \wedge \widetilde{\varphi}$ is satisfiable on binary trees. The result from the corollary follows, since MSO validity is decidable on binary trees. Indeed, the problem can be reduced to tree-automata emptiness [TW68]. ■

4.4 Special tree-width

In this section, we introduce special tree terms (STTs) and their semantics as labeled graphs. A special tree term using at most k colors (k-STT) defines a graph of special tree-width at most $k-1$. Special tree-width is similar to tree-width. See [Cou10] for more details on special tree-width and tree-width. We also give an MSO interpretation of the graph G_τ defined by a special tree term τ in the binary tree associated with τ.

A (Σ,Γ)-labeled graph is a tuple $G=(V,(E_\gamma)_{\gamma\in\Gamma},\lambda)$ where $\lambda\colon V\to\Sigma$ is the vertex labeling and $E_\gamma\subseteq V^2$ is the set of edges for each label $\gamma\in\Gamma$.

Special tree terms form an algebra to define labeled graphs. The syntax of k-STTs over (Σ,Γ) is given by

$$\tau ::= (i,a) \mid \mathsf{Add}_{i,j}^{\gamma}\,\tau \mid \mathsf{Forget}_i\,\tau \mid \mathsf{Rename}_{i,j}\,\tau \mid \tau\oplus\tau$$

where $a\in\Sigma$, $\gamma\in\Gamma$ and $i,j\in[k]=\{1,\ldots,k\}$ are colors.

Each k-STT represents a colored graph $[\![\tau]\!]=(G_\tau,\chi_\tau)$ where G_τ is a (Σ,Γ)-labeled graph and $\chi_\tau\colon[k]\to V$ is a partial injective function assigning a vertex of G_τ to some colors.

- $[\![(i,a)]\!]$ consists of a single a-labeled vertex with color i.
- $\mathsf{Add}_{i,j}^{\gamma}$ adds a γ-labeled edge to the vertices colored i and j (if such vertices exist).

  Formally, if $[\![\tau]\!]=(V,(E_\gamma)_{\gamma\in\Gamma},\lambda,\chi)$ then $[\![\mathsf{Add}_{i,j}^{\alpha}\,\tau]\!]=(V,(E'^{\gamma})_{\gamma\in\Gamma},\lambda,\chi)$ with $E'^{\gamma}=E_\gamma$ if $\gamma\neq\alpha$ and $E'^{\alpha}=\begin{cases}E_\alpha & \text{if } \{i,j\}\not\subseteq\mathsf{dom}(\chi)\\ E_\alpha\cup\{(\chi(i),\chi(j))\} & \text{otherwise.}\end{cases}$
- Forget_i removes color i from the domain of the color map.

  Formally, if $[\![\tau]\!]=(V,(E_\gamma)_{\gamma\in\Gamma},\lambda,\chi)$ then $[\![\mathsf{Forget}_i\,\tau]\!]=(V,(E_\gamma)_{\gamma\in\Gamma},\lambda,\chi')$ with $\mathsf{dom}(\chi')=\mathsf{dom}(\chi)\setminus\{i\}$ and $\chi'(j)=\chi(j)$ for all $j\in\mathsf{dom}(\chi')$.
- $\mathsf{Rename}_{i,j}$ exchanges the colors i and j.

  Formally, if $[\![\tau]\!]=(V,(E_\gamma)_{\gamma\in\Gamma},\lambda,\chi)$ then $[\![\mathsf{Rename}_{i,j}\,\tau]\!]=(V,(E_\gamma)_{\gamma\in\Gamma},\lambda,\chi')$ with $\chi'(\ell)=\chi(\ell)$ if $\ell\in\mathsf{dom}(\chi)\setminus\{i,j\}$, $\chi'(i)=\chi(j)$ if $j\in\mathsf{dom}(\chi)$ and $\chi'(j)=\chi(i)$ if $i\in\mathsf{dom}(\chi)$.
- Finally, $\oplus$ constructs the disjoint union of the two graphs provided they use different colors. This operation is undefined otherwise.

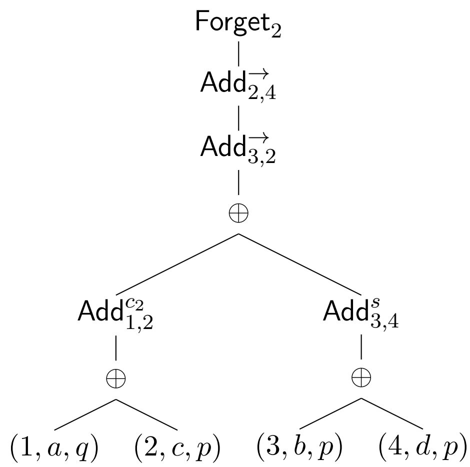
  Formally, if $[\![\tau_i]\!]=(G_i,\chi_i)$ for $i=1,2$ and $\mathsf{dom}(\chi_1)\cap\mathsf{dom}(\chi_2)=\emptyset$ then $[\![\tau_1\oplus\tau_2]\!]=(G_1\uplus G_2,\chi_1\uplus\chi_2)$. Otherwise, $\tau_1\oplus\tau_2$ is not a valid STT.

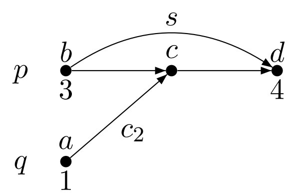
The special tree-width of a graph G is the least k such that $G=G_\tau$ for some $(k+1)$-STT τ.

Example 4.13. For CBMs, we have process edges and data edges, so we take $\Gamma = \{\rightarrow\} \cup \mathsf{DS}$. Also, vertices of CBMs are labeled with a letter from Σ and a process from Procs. Hence the labels of vertices are pairs in $\Sigma \times \mathsf{Procs}$ and the atomic STTs used to define CBMs are of the form (i, a, p) with $i \in [k]$, $a \in \Sigma$ and $p \in \mathsf{Procs}$. Consider the following 4-STT:

$$\tau = \mathsf{Forget}_2\, \mathsf{Add}^{\rightarrow}_{2,4}\, \mathsf{Add}^{\rightarrow}_{3,2}(\mathsf{Add}^{c_2}_{1,2}((1, a, q) \oplus (2, c, p)) \oplus \mathsf{Add}^{s}_{3,4}((3, b, p) \oplus (4, d, p)))$$

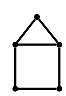
It is depicted below (left) as a tree and the graph G_τ is given on the right. $\diamondsuit$





Definition 4.14. *Let* $\mathsf{CBM}^{k\text{-}\mathsf{stw}}(\mathfrak{A}, \Sigma)$ *denote the set of CBMs with special tree-width bounded by* k. $\diamondsuit$

Exercise 4.15. Prove that trees have special tree-width (at most) 1. $\diamondsuit$

Exercise 4.16. Give a 3-STT for the following graph:  $\diamondsuit$

Exercise 4.17. Show that the rename operation is redundant. More precisely, show that for every k-STT τ (possibly using the rename operation) we can construct a k-STT τ' which does not use the rename operation and such that $[\![\tau]\!] = [\![\tau']\!]$, i.e., $G_\tau = G_{\tau'}$ and $\chi_\tau = \chi_{\tau'}$. $\diamondsuit$

The decomposition game for special tree-width is a two player turn based game $\mathsf{Arena}(\Sigma, \Gamma) = (\mathsf{Pos}_\exists \uplus \mathsf{Pos}_\forall, \mathsf{Moves})$. Eve's set of positions $\mathsf{Pos}_\exists$ consists of marked (colored) graphs (G, U) where $G = (V, (E_\gamma)_{\gamma\in\Gamma}, \lambda)$ is a (Σ, Γ)-labeled graph and $U \subseteq V$ is the subset of marked vertices. Adam's set of positions $\mathsf{Pos}_\forall$ consists of pairs of marked graphs. The edges Moves of Arena reflect the moves of the players. Eve's moves from (G, U) consist in

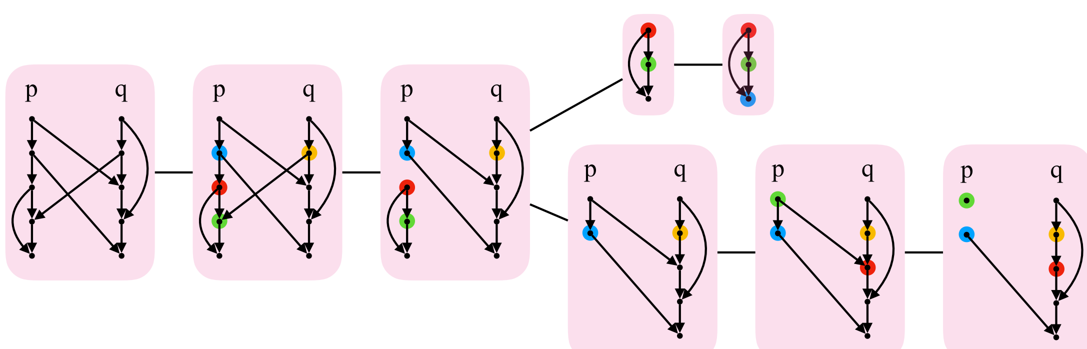
1. marking some vertices of the graph resulting in (G, U') with $U \subseteq U' \subseteq V$,
2. removing edges whose endpoints are marked, resulting in (G', U),
3. dividing (G, U) in (G_1, U_1) and (G_2, U_2) such that G is the disjoint union of G_1 and G_2 (in particular $V_1 \cap V_2 = \emptyset$ and $V = V_1 \cup V_2$) and marked nodes are inherited ($U_1 = U \cap V_1$ and $U_2 = U \cap V_2$).

Adam's moves amount to choosing one of the two marked graphs. Terminal positions of the game are graphs where all vertices are marked. Neither Eve nor Adam can move from terminal positions which are winning for Eve.

A play is a path in Arena starting from some marked graph (G, U) and leading to a terminal position. The cost of the play is the maximum number of marked vertices in the positions of the path. Eve's objective is to minimize the cost and Adam's objective is to maximize the cost.

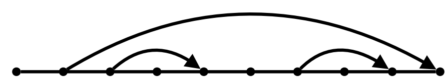
A (positional) strategy for Eve starting from a marked graph (G, U) is k-winning if all plays starting from (G, U) and following the strategy have cost at most k. The special tree-width of a graph G is the least k such that Eve has a $(k + 1)$-winning strategy starting from $(G, \emptyset)$ (initially G is unmarked).

Exercise 4.18. Prove that a k-winning strategy for Eve starting from (G, U) can be described with a valid k-STT τ with $[\![\tau]\!] = (G, \chi)$ and $\mathsf{dom}(\chi) = U$. $\diamondsuit$



Example 4.19. The CBM on the left of the Figure above has STW at most 3. The beginning of a 4-winning strategy for Eve is depicted as a tree. She starts by marrking four nodes, then she removes two edges and the resulting graph is disconnected. The component with three vertices can easily be made terminal by marking the last node. On the second component (bottom branch), Eve marks two more vertices and removes two edges so that the green node is disconnected.

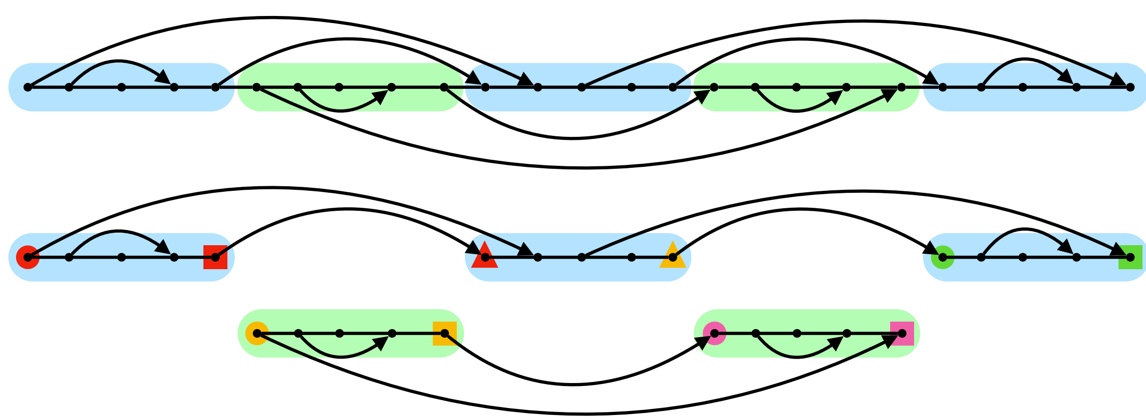
On the remaining component, she marks the last node and removes one edge so that the blue node is disconnected. Then she marks the second to last node and removes one edge disconnecting the last node. Finally, she marks the first node reaching a terminal position. ♢



Example 4.20. Nested words have special tree-width bounded by 3.

We describe a 4-winning strategy for Eve. First she marks the first and last point of the word. Then she repeats the following steps until reaching a terminal position:

1. If the first point is not a push (the source of a $\rhd$) then she marks the second point and she removes the first linear edge. The graph is disconnected. One component is terminal. The other one is a nested word with the endpoints marked.
2. If there is a $\rhd$-edge from the first point to the last point, then Eve removes this edge and continues as above.
3. If there is a $\rhd$-edge from the first point, call it e, to some middle point, call it f, then Eve marks f and its linear successor g. Then she removes the matching edge $e \rhd f$ and the successor edge $f \to g$. The resulting graph is disconnected. Both connected components are nested words. Each one has its endpoints marked. ♢



Example 4.21. Multiply nested words with at most $k \geq 2$ contexts have special tree-width bounded by $2k - 1$.

Eve marks each endpoint of each context. Doing so, she marks at most $2k$ vertices. Then, she removes the successor edges between contexts. The graph is disconnected. Each connected component is a (simply) nested word with at most $2\lceil \frac{k}{2} \rceil$ ($\leq 2k-2$) marked vertices. Using two extra marks, Eve applies on each connected component the strategy described in Example 4.20. ♢

Example 4.22. Let $n, k \geq 1$. CBMs over n processes and which are k-existentially bounded have special tree-width bounded by $k + n$. ♢

4.6 Main Results

5th Lecture

Consider the following problems:

stw-Nonemptiness$(\mathfrak{A}, \Sigma)$:	
Instance:	$\mathcal{S} \in \mathsf{CPDS}(\mathfrak{A}, \Sigma)$; $k \geq 0$
Question:	$L(\mathcal{S}) \cap \mathsf{CBM}^{k\text{-stw}} \neq \emptyset$?

stw-Inclusion$(\mathfrak{A}, \Sigma)$:	
Instance:	$\mathcal{S}, \mathcal{S}' \in \mathsf{CPDS}(\mathfrak{A}, \Sigma)$; $k \geq 0$
Question:	$L(\mathcal{S}) \cap \mathsf{CBM}^{k\text{-stw}} \subseteq L(\mathcal{S}')$?

stw-Universality$(\mathfrak{A}, \Sigma)$:	
Instance:	$\mathcal{S} \in \mathsf{CPDS}(\mathfrak{A}, \Sigma)$; $k \geq 0$
Question:	$\mathsf{CBM}^{k\text{-stw}} \subseteq L(\mathcal{S})$?

stw-Satisfiability$(\mathfrak{A}, \Sigma)$:	
Instance:	$\varphi \in \mathsf{MSO}(\mathfrak{A}, \Sigma)$; $k \geq 0$
Question:	$L(\varphi) \cap \mathsf{CBM}^{k\text{-stw}} \neq \emptyset$?

stw-ModelChecking$(\mathfrak{A}, \Sigma)$:	
Instance:	$\mathcal{S} \in \mathsf{CPDS}(\mathfrak{A}, \Sigma)$; $\varphi \in \mathsf{MSO}(\mathfrak{A}, \Sigma)$; $k \geq 0$
Question:	$L(\mathcal{S}) \cap \mathsf{CBM}^{k\text{-stw}} \subseteq L(\varphi)$?

In the following, we will prove the following:

Theorem 4.23. *All these problems are decidable.*

The proof technique is via an interpretation of $\mathsf{CBM}^{k\text{-stw}}$ in binary trees and reduction to problems on tree automata. This is actually similar to decidability for cographs as explained in Section 4.3.

In the following, we introduce

- $\mathcal{A}_{\mathsf{cbm}}^{k\text{-stw}}$: a tree automaton accepting k-STTs denoting graphs in $\mathsf{CBM}^{k\text{-stw}}$.
- $\mathcal{A}_{\mathcal{S}}^{k\text{-stw}}$: a tree automaton for $\mathcal{S} \in \mathsf{CPDS}(\mathfrak{A}, \Sigma)$ such that for all $\tau \in L(\mathcal{A}_{\mathsf{cbm}}^{k\text{-stw}})$, τ is accepted by $\mathcal{A}_{\mathcal{S}}^{k\text{-stw}}$ iff $G_\tau \in L(\mathcal{S})$.
- $\mathcal{A}_{\varphi}^{k\text{-stw}}$: a tree automaton for $\varphi \in \mathsf{MSO}(\mathfrak{A}, \Sigma)$ such that for all $\tau \in L(\mathcal{A}_{\mathsf{cbm}}^{k\text{-stw}})$, τ is accepted by $\mathcal{A}_{\varphi}^{k\text{-stw}}$ iff $G_\tau \models \varphi$.

We show now that a graph G defined by an STT τ can be interpreted in the binary tree τ. Notice that the vertices of G are in bijection with the leaves of τ. The main difficulty is to interpret the edge relations E_γ of G in the tree τ.

Let us denote by Λ^k the alphabet of k-STTs (we do not include the rename operation since it is redundant):

$$\Lambda^k = \{\oplus, (i,a), \mathsf{Add}^{\gamma}_{i,j}, \mathsf{Forget}_i \mid i,j \in [k], a \in \Sigma, \gamma \in \Gamma\}\,.$$

Clearly, the set of k-STTs considered as binary trees over Λ^k is a regular tree language.

Lemma 4.24. There is a formula $\Phi^{k\text{-stt}}_{\mathsf{valid}} \in \mathsf{MSO}(\Lambda^k, \downarrow_0, \downarrow_1)$ which defines the set of *valid* k-STTs.

Proof. First, the binary tree is correctly labeled: leaves should have labels in $[k]\times\Sigma$, unary nodes should have labels in $\{\mathsf{Add}^{\gamma}_{i,j}, \mathsf{Forget}_i \mid i,j \in [k]$ and $\gamma \in \Gamma\}$ and binary nodes should be labeled $\oplus$. Moreover, for the k-STT to be valid, the children of a binary node should have disjoint sets of *active* colors. This can be expressed with

$$\neg\exists x, x', y, y', z \bigvee_{\substack{1\le i\le k\\ a,b\in\Sigma}} P_{(i,a)}(x) \wedge P_{(i,b)}(y) \wedge \oplus(z) \wedge z{\downarrow_0}x' \wedge \beta_i(x,x') \wedge z{\downarrow_1}y' \wedge \beta_i(y,y')$$

where $\beta_i(x,z)$ is a macro stating that z is an ancestor of x in the tree and that Forget_i does not occur in the tree between node x and node z. This formula can be written in MSO with a transitive closure. ∎

Exercise 4.25. Give a tree automaton $\mathcal{A}^{k\text{-stt}}_{\mathsf{valid}}$ with 2^k states that accepts the set of *valid* k-STTs. ♢

Proposition 4.26 (MSO interpretation). For all sentences $\varphi \in \mathsf{MSO}(\Sigma,\Gamma)$ and all $k > 0$, there is a sentence $\widetilde{\varphi}^k \in \mathsf{MSO}(\Lambda^k, \downarrow_0, \downarrow_1)$ such that, for every valid k-STT τ with $[\![\tau]\!] = (G,\chi)$, we have

$$G \models \varphi \quad\text{iff}\quad \tau \models \widetilde{\varphi}^k\,.$$

Proof. We proceed by induction on φ. Hence we also have to deal with free variables. We denote by $\mathcal{I}$ an interpretation of variables to (sets of) vertices of G which are identified with leaves of τ. Hence, we prove by induction that for all formulas $\varphi \in \mathsf{MSO}(\Sigma,\Gamma)$ and all $k > 0$, there is a formula $\widetilde{\varphi}^k \in \mathsf{MSO}(\Lambda^k, \downarrow_0, \downarrow_1)$ such that, for all valid k-STT τ with $[\![\tau]\!] = (G,\chi)$, and all interpretations $\mathcal{I}$ in vertices of G, we have

$$G \models_{\mathcal{I}} \varphi \quad\text{iff}\quad \tau \models_{\mathcal{I}} \widetilde{\varphi}^k\,.$$

The difficult case is to translate the edge relations. We define

$$\widetilde{x\, E_\gamma\, y}^k = \exists z \bigvee_{\substack{1\le i,j\le k\\ a,b\in\Sigma}} P_{(i,a)}(x) \wedge P_{(j,b)}(y) \wedge \mathsf{Add}^{\gamma}_{i,j}(z) \wedge \beta_i(x,z) \wedge \beta_j(y,z)\,.$$

The other cases are easy:

$$\widetilde{P_a}^k(x) = \bigvee_{1\le i\le k} P_{(i,a)}(x) \qquad \widetilde{\neg\varphi}^k = \neg\,\widetilde{\varphi}^k \qquad \widetilde{\varphi_1 \vee \varphi_2}^k = \widetilde{\varphi_1}^k \vee \widetilde{\varphi_2}^k$$

$$\widetilde{\exists x\varphi}^k = \exists x\,(\widetilde{\varphi}^k \wedge \mathsf{leaf}(x)) \qquad \widetilde{x \in X}^k = x \in X$$

$$\widetilde{\exists X\varphi}^k = \exists X\,(\widetilde{\varphi}^k \wedge \forall x(x \in X \rightarrow \mathsf{leaf}(x))) \qquad \widetilde{x = y}^k = (x = y)$$

This concludes the proof. ∎

Corollary 4.27. The MSO theory of graphs of special tree-width at most k is decidable.

Proof. Let $\varphi \in \mathsf{MSO}(\Sigma,\Gamma)$. Then, φ is valid on graphs of special tree-width at most $k-1$ iff $\Phi^{k\text{-stt}}_{\mathsf{valid}} \rightarrow \widetilde{\varphi}^k$ is valid on binary trees. The result of the corollary follows, since MSO validity is decidable on binary trees. Indeed, the problem can be reduced to tree-automata emptiness [TW68]. Note that, moreover, φ is satisfiable on graphs of special tree-width at most $k-1$ iff $\Phi^{k\text{-stt}}_{\mathsf{valid}} \wedge \widetilde{\varphi}^k$ is satisfiable on binary trees. ∎

Exercise 4.28. Construct a tree automaton $\mathcal{A}^k_\gamma$ with $\mathcal{O}(k^2)$ states which accepts a valid k-STT τ with two marked leaves x and y iff there is a γ-edge between x and y in the graph $[\![\tau]\!]$. ♢

Solution: $\mathcal{A}^k_\gamma$ is a deterministic bottom-up tree automaton. It keeps in its state a pair of colors $(i,j) \in \{0,1,\ldots,k\}^2$ where i is the color at the current node of leaf x, with $i=0$ if x is not in the current subtree. Same for j and y. The state is initialized at leaves. It is updated at $\oplus$-nodes. The automaton goes to an accepting state if it is in state (i,j) when reading a node labeled $\mathsf{Add}^\gamma_{i,j}$. On the other hand, it goes to a rejecting state at a node Forget_ℓ if it is in state (i,j) with $\ell \in \{i,j\}$.

Exercise 4.29. Construct a tree-walking automaton $\mathcal{B}^k_\gamma$ with $\mathcal{O}(k)$ states which runs on a valid k-STT τ starting from a leaf (say x) and accepts when reaching a leaf (say y) such that there is a γ-edge between x and y in the graph $[\![\tau]\!]$. ♢

Solution: First, walking up the tree, the automaton $\mathcal{B}^k_\gamma$ keeps in its state the color of the leaf x. It makes sure that the color is not forgotten, until it reaches a node labeled $\mathsf{Add}^\gamma_{i,j}$ where i is the color in its state. Then, it updates its state with color j and it enters a second phase where it walks down the tree (non-deterministically at $\oplus$-nodes), making sure that the color is not forgotten, until it reaches a leaf having the color that corresponds to its state.

<u>Remark:</u> The automaton $\mathcal{B}^k_\gamma$ can be made deterministic if there is at most one γ edge with source x. Indeed, walking up the tree is deterministic and we can search for the target leaf y using a DFS.

Recall that the class $\mathsf{CBM}(\mathfrak{A},\Sigma)$ is $\mathsf{MSO}(\mathfrak{A},\Sigma)$-definable by a sentence Φ_{cbm} (Proposition 4.5).

Corollary 4.30. The problem STW-SATISFIABILITY and STW-VALIDITY are decidable and the complexity is non-elementary.

Proof. Let $\varphi \in \mathsf{MSO}(\mathfrak{A},\Sigma)$ and let $k > 0$.

Using Proposition 4.26, we construct the formulas $\widetilde{\varphi}^k, \widetilde{\Phi_{\mathsf{cbm}}}^k \in \mathsf{MSO}(\Lambda^k, \downarrow_0, \downarrow_1)$. Consider the formula $\Phi_{\mathsf{valid}}^{k\text{-stt}} \in \mathsf{MSO}(\Lambda^k, \downarrow_0, \downarrow_1)$ given by Lemma 4.24. Then, we define $\Phi = \Phi_{\mathsf{valid}}^{k\text{-stt}} \wedge \widetilde{\Phi_{\mathsf{cbm}}}^k \wedge \widetilde{\varphi}^k \in \mathsf{MSO}(\Lambda^k, \downarrow_0, \downarrow_1)$.

We prove that φ is satisfiable over $\mathsf{CBM}^{k\text{-stw}}$ iff Φ is satisfiable over Λ^k-labeled binary trees.

Indeed, let τ be a binary tree over Λ^k such that $\tau \models \Phi$. Then, τ is a k-STT and $[\![\tau]\!] = (G, \chi)$ where G is a CBM and $G \models \varphi$. Conversely, Let $G \in \mathsf{CBM}^{k\text{-stw}}$ be such that $G \models \varphi$. Let τ be any k-STT with $[\![\tau]\!] = (G, \chi)$. We have $\tau \models \Phi$.

From [TW68], we can construct a tree automaton $\mathcal{A}_\Phi$ equivalent to Φ and then check $\mathcal{A}_\Phi$ for emptiness. Recall that emptiness for tree automata can be solved in PTIME.

The decidability of STW-VALIDITY follows since validity of φ over $\mathsf{CBM}^{k\text{-stw}}$ is equivalent to non-satifiability of $\neg\varphi$ over $\mathsf{CBM}^{k\text{-stw}}$. ∎

Corollary 4.31. The problem STW-NONEMPTINESS is decidable and EXPTIME-complete.

Proof. Let $\mathcal{S} \in \mathsf{CPDS}(\mathfrak{A},\Sigma)$ and let $k > 0$. Let $\Phi_{\mathcal{S}} \in \mathsf{MSO}(\mathfrak{A},\Sigma)$ be the equivalent formula given by Theorem 3.4. Then, we have $L(\mathcal{S}) \cap \mathsf{CBM}^{k\text{-stw}} \neq \emptyset$ iff $\Phi_{\mathcal{S}}$ is satisfiable over $\mathsf{CBM}^{k\text{-stw}}$. We conclude using Corollary 4.30.

We will prove the EXPTIME upper-bound later using direct constructions of tree automata. ∎

Corollary 4.32. The problem STW-MODELCHECKING is decidable and the complexity is non-elementary.

Proof. This is a consequence of Proposition 4.6 and Corollary 4.30. We may also argue directly as follows. Let $\mathcal{S} \in \mathsf{CPDS}(\mathfrak{A},\Sigma)$ be the system, let $\varphi \in \mathsf{MSO}(\mathfrak{A},\Sigma)$ be the specification and let $k > 0$. We consider as above the formulas $\Phi_{\mathcal{S}}, \Phi_{\mathsf{cbm}} \in \mathsf{MSO}(\mathfrak{A},\Sigma)$. Then, $L(\mathcal{S}) \cap \mathsf{CBM}^{k\text{-stw}} \subseteq L(\varphi)$ iff the formula $\Phi = \Phi_{\mathsf{valid}}^{k\text{-stt}} \wedge \widetilde{\Phi_{\mathsf{cbm}}}^k \wedge \widetilde{\Phi_{\mathcal{S}}}^k \wedge \widetilde{\neg\varphi}^k$ is *not* satisfiable over Λ^k-labeled binary trees.

We conclude as in the proof of Corollary 4.30. ∎

Exercise 4.33. Prove STW-INCLUSION and STW-UNIVERSALITY decidable. ◇

Recall that for an STT τ we write $[\![\tau]\!] = (G_\tau, \chi_\tau)$.

Not all k-STTs τ define graphs G_τ which are CBMs. In order to use the tree interpretation in STTs to efficiently solve problems on CBMs of bounded special tree-width, we will construct a tree automaton $\mathcal{A}^{k\text{-stw}}_{\mathsf{cbm}}$ which accepts k-STTs denoting CBMs (Proposition 4.38).

As a first warm-up, we construct a tree automaton checking that the graph associated with a STT is acyclic.

Proposition 4.34. There is a deterministic bottom-up tree automaton $\mathcal{A}^{k\text{-stw}}_{\mathsf{acyclic}}$ of size $2^{\mathcal{O}(k^2)}$ which accepts all binary trees τ such that τ is a k-STT and G_τ is acyclic.

Proof. A state of $\mathcal{A}^{k\text{-stw}}_{\mathsf{acyclic}}$ is a pair $(P, \prec)$ where $P \subseteq [k]$ and $\prec$ is a strict order on P. When reading an STT τ the automaton will reach the state $(P, \prec)$ satisfying the following two conditions:

(A_1) $P = \mathsf{dom}(\chi) \subseteq [k]$ is the set of active colors,
(A_2) $\prec$ is the restriction of the strict order $< = (\rightarrow \cup \rhd)^+$ to P.

The transitions are defined below (with $s = (P, \prec)$, $s' = (P', \prec')$, $s'' = (P'', \prec'')$).

$\perp \xrightarrow{(i,a,p)} s$	is a transition if $P = \{i\}$ and $\prec = \emptyset$.
$s' \xrightarrow{\mathsf{Add}^{\rightarrow}_{i,j}} s$	is a transition if $i, j \in P'$, $i \neq j$ and $\neg(j \prec' i)$. Then, $P = P'$ and $\prec = (\prec' \cup \{(i,j)\})^+$.
$s' \xrightarrow{\mathsf{Add}^{d}_{i,j}} s$	is a transition if $i, j \in P'$, $i \neq j$ and $\neg(j \prec' i)$. Then, $P = P'$ and $\prec = (\prec' \cup \{(i,j)\})^+$.
$s' \xrightarrow{\mathsf{Forget}_i} s$	is a transition if $i \in P'$. Then $P = P' \setminus \{i\}$ and $\prec = \prec' \cap (P \times P)$.
$s', s'' \xrightarrow{\oplus} s$	is a transition if $P' \cap P'' = \emptyset$ (active colors should be disjoint). Then, $P = P' \uplus P''$ and $\prec = \prec' \cup \prec''$.

We can easily check that if $\mathcal{A}^{k\text{-stw}}_{\mathsf{acyclic}}$ has a run on a binary tree τ then τ is a k-STT and G_τ is acyclic. The number of states of $\mathcal{A}^{k\text{-stw}}_{\mathsf{acyclic}}$ is at most 2^{k+k^2}. ∎

The second warm-up is a tree automaton checking the local conditions of edges in a CBM (see Definition 2.9).

Proposition 4.35. There is a deterministic bottom-up tree automaton $\mathcal{A}^{k\text{-stw}}_{\mathsf{edges}}$ of size $2^{\mathcal{O}(k|\mathsf{Procs}|)}$ which accepts all binary trees τ such that τ is a k-STT and G_τ satisfies the following conditions:

1. process edges are not branching and are between events of the same process,
2. data edges are disjoint and respect the Writer/Reader constraints of data structures.

Proof. A state of $\mathcal{A}^{k\text{-stw}}_{\mathsf{edges}}$ is a tuple $s = (P, \pi, \alpha, \beta, \gamma)$ where $P \subseteq [k]$, $\alpha, \beta, \gamma \subseteq P$ and $\pi \colon P \to \mathsf{Procs}$. When reading an STT τ with $[\![\tau]\!] = (G, \chi)$, the automaton will reach the state s satisfying the following two conditions:

(B_1) $P = \mathsf{dom}(\chi) \subseteq [k]$ is the set of active colors,
(B_2) $\pi \colon P \to \mathsf{Procs}$ gives the process of the colored event: $\pi(i) = \mathsf{pid}(\chi(i))$,
(B_3) if $e \to f$ in G then $\mathsf{pid}(e) = \mathsf{pid}(f)$ and for all $i \in P$ we have $i \in \alpha$ iff $\chi(i)$ is the source of some $\to$-edge, and $i \in \beta$ iff $\chi(i)$ is the target of some $\to$-edge,
(B_4) if $e \rhd^d f$ in G then $\mathsf{pid}(e) = \mathsf{Reader}(d)$, $\mathsf{pid}(f) = \mathsf{Writer}(d)$ and for all $i \in P$ we have $i \in \gamma$ iff $\chi(i)$ is the source or target of some $\rhd$-edge.

The transitions are defined below (with $s = (P, \pi, \alpha, \beta, \gamma)$, $s' = (P', \pi', \alpha', \beta', \gamma')$, $s'' = (P'', \pi'', \alpha'', \beta'', \gamma'')$).

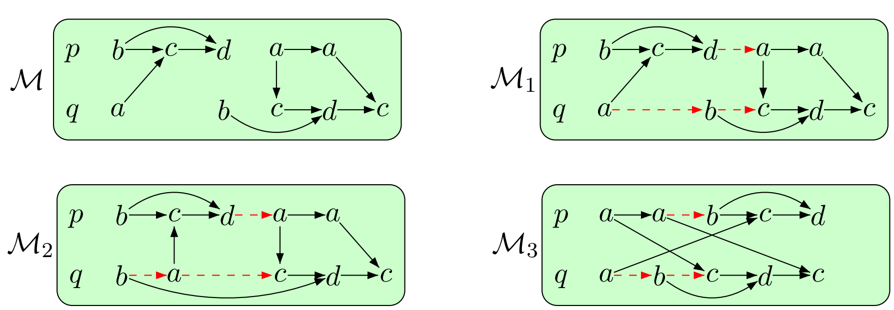
$\perp \xrightarrow{(i,a,p)} s$	is a transition if $P = \{i\}$, $\pi(i) = p$ and $\alpha = \beta = \gamma = \emptyset$.
$s' \xrightarrow{\mathsf{Add}^{\to}_{i,j}} s$	is a transition if $i, j \in P'$, $i \neq j$, $\pi'(i) = \pi'(j)$, $i \notin \alpha'$ and $j \notin \beta'$. Then, $P = P'$, $\pi = \pi'$, $\gamma = \gamma'$, $\alpha = \alpha' \cup \{i\}$ and $\beta = \beta' \cup \{j\}$.
$s' \xrightarrow{\mathsf{Add}^{d}_{i,j}} s$	is a transition if $i, j \in P'$, $i \neq j$, $i, j \notin \gamma'$, $\pi'(i) = \mathsf{Reader}(d)$ and $\pi'(j) = \mathsf{Writer}(d)$. Then, $P = P'$, $\pi = \pi'$, $\alpha = \alpha'$, $\beta = \beta'$ and $\gamma = \gamma' \cup \{i, j\}$.
$s' \xrightarrow{\mathsf{Forget}_i} s$	is a transition if $i \in P'$. Then $P = P' \setminus \{i\}$ and $\pi, \alpha, \beta, \gamma$ are the restrictions of $\pi', \alpha', \beta', \gamma'$ to P.
$s', s'' \xrightarrow{\oplus} s$	is a transition if $P' \cap P'' = \emptyset$ (active colors should be disjoint). Then, s is the disjoint union of s' and s''.

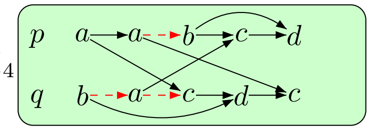
We can easily check that if $\mathcal{A}^{k\text{-stw}}_{\mathsf{edges}}$ has a run on a binary tree τ then τ is a k-STT and G_τ satisfies the local conditions of Definition 2.9. The number of states of $\mathcal{A}^{k\text{-stw}}_{\mathsf{edges}}$ is at most $2^{4k} \cdot |\mathsf{Procs}|^k$. ∎

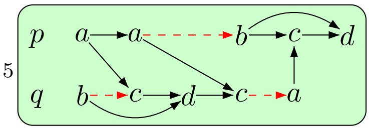
We turn now to the full automaton checking that an STT defines a CBM. We start with a definition. A *split*-CBM is a CBM in which behaviors of processes may be split in several factors.

Definition 4.36. *A graph $\mathcal{M} = (\mathcal{E}, \to, (\rhd^d)_{d \in \mathsf{DS}}, \mathsf{pid}, \lambda)$ is a* split-CBM *if it is possible to obtain a CBM $(\mathcal{E}, \to \uplus \dashrightarrow, (\rhd^d)_{d \in \mathsf{DS}}, \mathsf{pid}, \lambda)$ by adding some missing process edges $\dashrightarrow \in \mathcal{E}^2 \setminus \to$. A* factor *(or* block*) of $\mathcal{M}$ is a maximal sequence of events connected by process edges.* ◇

Example 4.37. The split-CBM $\mathcal{M}$ depicted below has 5 factors: 2 on process p and 3 on process q. It has two connected components. There are 5 ways to add the missing process edges in order to get a CBM: $\mathcal{M}_1, \ldots, \mathcal{M}_5$.



$\mathcal{M}_4$ 

$\mathcal{M}_5$ 

$\diamondsuit$

Proposition 4.38. There is a tree automaton $\mathcal{A}^{k\text{-}\mathsf{stw}}_{\mathsf{cbm}}$ of size $2^{\mathcal{O}(k^2|\mathfrak{A}|)}$ which accepts all binary trees τ such that τ is a k-STT and $[\![\tau]\!] = (G_\tau, \chi_\tau)$ is an uncolored CBM: G_τ is a CBM and $\mathsf{dom}(\chi_\tau) = \emptyset$.

Actually, the automaton $\mathcal{A}^{k\text{-}\mathsf{stw}}_{\mathsf{cbm}}$ admits a run (not necessarily accepting) on a binary tree τ iff τ is a k-STT and $[\![\tau]\!] = (G_\tau, \chi_\tau)$ where G_τ is a split-CBM.

Proof. For simplicity, we construct the automaton assuming that all data structures are bags. In case of stacks or queues, we could enforce the LIFO or FIFO properties with an intersection with a further automaton (see Exercises below).

The automaton $\mathcal{A}^{k\text{-}\mathsf{stw}}_{\mathsf{cbm}}$ is nondeterministic and computes an abstraction of the split-CBM defined by a given term. More precisely, when reading bottom-up a binary tree τ, the automaton $\mathcal{A}^{k\text{-}\mathsf{stw}}_{\mathsf{cbm}}$ checks that τ is a k-STT and reaches a state $s = (P, \pi, \alpha, \beta, \gamma, \prec, L, R)$ satisfying the following conditions with $[\![\tau]\!] = (G, \chi)$:

(I_1) $P = \mathsf{dom}(\chi) \subseteq [k]$ is the set of active colors,

(I_2) $\pi \colon P \to \mathsf{Procs}$ gives the associated processes: $\pi(i) = \mathsf{pid}(\chi(i))$ for all $i \in P$,

(I_3) $\alpha, \beta, \gamma \subseteq P$ are such that

- $i \in \alpha$ iff $\chi(i)$ is the source of a $\to$-edge in G.
- $i \in \beta$ iff $\chi(i)$ is the target of a $\to$-edge in G.
- $i \in \gamma$ iff $\chi(i)$ is the source or target of a $\rhd$-edge in G.

(I_4) $\prec \subseteq P^2$ is a strict partial order such that for all $i, j \in P$,

(a) $\chi(i)\ (\to \cup \rhd)^+\ \chi(j)$ in G implies $i \prec j$,

(b) $\pi(i) = \pi(j)$ and $i \neq j$ implies $i \prec j$ or $j \prec i$.
Hence, for each $p \in \mathsf{Procs}$, $\prec$ defines a total order on $\pi^{-1}(p) \subseteq P$.
We denote by $\lessdot_p$ the successor relation of this total order.

(c) If $i \lessdot_p j$ then $i \in \alpha$ iff $j \in \beta$,

(d) If $i \lessdot_p j \wedge i \in \alpha$ then $\chi(i) \to^+ \chi(j)$,

The partial order $\prec$ is guessed by $\mathcal{A}^{k\text{-}\mathsf{stw}}_{\mathsf{cbm}}$ so that it will eventually correspond to the order of the final CBM defined by the global term. So $\prec$ must be compatible with all $\to$ and $\rhd$ edges already added in the subterm τ (I_4a) and for each process the final ordering has been guessed already (I_4b), even though some $\to$-edges may still be missing in G.

Together with α and β, the partial order $\prec$ allows to locate the *holes* between consecutive factors of processes. Formally, the hole relation is defined by $i \rightsquigarrow j$ if $i \notin \alpha \wedge i \lessdot_p j$ for some $p \in \mathsf{Procs}$.

(I_5) $[\![\tau]\!] = (G, \chi)$ is a split-CBM with the additional process edges defined by $\chi(i) \dashrightarrow \chi(j)$ iff $i \rightsquigarrow j$.

$L, R \subseteq \mathsf{Procs}$ give the processes whose minimal/maximal event in the split-CBM $([\![\tau]\!], \dashrightarrow)$ is no more colored. More precisely, consider the sequence $w_1, \ldots, w_k$ of factors of some process $p \in \mathsf{Procs}$. Let $e_1, f_1, \ldots, e_k, f_k$ be the endpoints of these factors. We have $e_1 \rightarrow^* f_1 \dashrightarrow e_2 \rightarrow^* f_2 \dashrightarrow \cdots e_k \rightarrow^* f_k$. By definition of $\dashrightarrow$, the events $f_1, e_2, \ldots, e_k$ must be colored.
Now, we have $p \in L$ iff e_1 is not colored, and $p \in R$ iff f_k is not colored.

Notice that the number of states of $\mathcal{A}^{k\text{-stw}}_{\mathsf{cbm}}$ is at most $2^{4k} \cdot |\mathsf{Procs}|^k \cdot 2^{k^2} \cdot 2^{2|\mathsf{Procs}|}$.

The *transitions* of $\mathcal{A}^{k\text{-stw}}_{\mathsf{cbm}}$ are defined in Table 4.1. We check inductively that the invariants (I_1–I_5) are preserved by the transitions of $\mathcal{A}^{k\text{-stw}}_{\mathsf{cbm}}$. This is clear at the leaves. Let us go through the other cases.

- $\mathsf{Add}^{\rightarrow}_{i,j}$. Clearly, ($\mathsf{I}_1$–$\mathsf{I}_3$) are preserved. Next, ($\mathsf{I}_4$a) is also preserved since the edge $\chi(i) \rightarrow \chi(j)$ is only added when $i \rightsquigarrow j$, which implies $i \prec j$. Items (I_4b–I_4d) are trivially preserved. Finally, (I_5) is also preserved since the effect of $\mathsf{Add}^{\rightarrow}_{i,j}$ is to turn the $\dashrightarrow$-edge from $\chi(i)$ to $\chi(j)$ into a $\rightarrow$-edge.

- $\mathsf{Add}^{d}_{i,j}$. As above, (I_1–I_4) are trivially preserved. For (I_5), notice that the effect of $\mathsf{Add}^{d}_{i,j}$ is to add a $\rhd^d$-edge from $\chi(i)$ to $\chi(j)$. The resulting graph is still a split-CBM since the transition is only allowed when $i \prec j$, $\pi(i) = \mathsf{Writer}(d)$, $\pi(j) = \mathsf{Reader}(d)$ and $i, j \notin \gamma$.

- Forget_i. We can also easily check that (I_1–I_5) are preserved. The only non-trivial cases are (I_4c) and (I_4d). Assume that $j \prec'_p k$. Either $j \prec_p k$ and we can conclude easily. Or $j \prec_p i \prec_p k$ and the conditions of the Forget_i-transition imply that $i \in \alpha$ and $i \in \beta$. By induction we obtain $k \in \beta$, $j \in \alpha$ and $\chi(j) \rightarrow^+ \chi(i) \rightarrow^+ \chi(k)$. Hence, ($\mathsf{I}_4$c) and ($\mathsf{I}_4$d) hold.

- $\oplus$ is the most difficult case. Assume that the transition $s', s'' \xrightarrow{\oplus} s$ is applied at the root of a term $\tau = \tau' \oplus \tau''$ and that the invariants (I_1–I_5) hold for (s', τ') and (s'', τ''). Since $P' \cap P'' = \emptyset$, (I_1) implies that $\mathsf{dom}(\chi_{\tau'})$ and $\mathsf{dom}(\chi_{\tau''})$ are disjoint and τ is a legal k-STT. It is easy to check that (I_1–I_3) hold for (s, τ).

  We turn now to (I_4). By definition of the $\oplus$-transition, $\prec$ is a strict partial order on P satisfying (I_4b). Now, $\chi(i)\ (\rightarrow \cup \rhd)^+\ \chi(j)$ in G_τ iff $\chi(i)\ (\rightarrow \cup \rhd)^+\ \chi(j)$ in $G_{\tau'}$ or in $G_{\tau''}$, which implies $i \prec' j$ or $i \prec'' j$, and finally $i \prec j$. Hence, (I_4a) holds. Assume that $i \prec_p j$. If $i \in \alpha$ or $j \in \beta$ then $i \prec'_p j$ or $i \prec''_p j$ by definition of the $\oplus$-transition. We deduce that $j \in \alpha \cap \beta$ and $\chi(i) \rightarrow^+ \chi(j)$. The other case is $i \notin \alpha$ and $j \notin \beta$. Hence, (I_4c) and (I_4d) hold. Hence, (I_4) holds for (s, τ).

  Next, we check that (I_5) holds for the pair (s, τ). Let $[\![\tau]\!] = (G, \chi)$ with $G = (\mathcal{E}, \rightarrow, (\rhd^d)_{d \in \mathsf{DS}}, \mathsf{pid}, \lambda)$. We have to prove that $[\![\tau]\!]$ is a split-CBM with additional process edges defined by $\dashrightarrow = \{(\chi(i), \chi(j)) \mid i, j \in P \wedge i \rightsquigarrow j\}$.

  Let $p \in \mathsf{Procs}$. The set of p-events is $\mathcal{E}_p = \mathcal{E}'_p \uplus \mathcal{E}''_p$. If $\mathcal{E}''_p = \emptyset$ then $\dashrightarrow$ coincide with $\dashrightarrow'$ on $\mathcal{E}_p = \mathcal{E}'_p$ and $(\mathcal{E}_p, \rightarrow \cup \dashrightarrow, \lambda)$ is indeed a word. The same holds

$\perp \xrightarrow{(i,a,p)} s$	is a transition if $P = \{i\}$, $\pi(i) = p$, $\alpha = \beta = \gamma = \emptyset$, $\prec\, = \emptyset$ and $L = R = \emptyset$.
$s \xrightarrow{\mathsf{Add}^{\rightarrow}_{i,j}} s'$	is a transition if $i, j \in P$ and $i \rightsquigarrow j$. Then, $P' = P$, $\pi' = \pi$, $\alpha' = \alpha \cup \{i\}$, $\beta' = \beta \cup \{j\}$, $\gamma' = \gamma$, $\prec' = \prec$, $L' = L$ and $R' = R$.
$s \xrightarrow{\mathsf{Add}^{d}_{i,j}} s'$	is a transition if $i, j \in P$, $i \prec j$, $\pi(i) = \mathsf{Writer}(d)$, $\pi(j) = \mathsf{Reader}(d)$, $i, j \notin \gamma$. Then, $P' = P$, $\pi' = \pi$, $\alpha' = \alpha$, $\beta' = \beta$, $\gamma' = \gamma \cup \{i, j\}$, $\prec' = \prec$, $L' = L$ and $R' = R$.
$s \xrightarrow{\mathsf{Forget}_i} s'$	is a transition if $i \in P$ and $i \in \alpha \vee (\pi(i) \notin R \wedge \forall k,\ \pi(k) = \pi(i) \implies k \preceq i)$ and $i \in \beta \vee (\pi(i) \notin L \wedge \forall k,\ \pi(k) = \pi(i) \implies i \preceq k)$. For color i to be forgotten, the corresponding event e should already be the source of a $\rightarrow$-edge ($i \in \alpha$) or it should be maximal on its process, and symmetrically, it should be the target of a $\rightarrow$-edge or it should be minimal on its process. Then, $P' = P \setminus \{i\}$ and $\pi', \alpha', \beta', \gamma', \prec'$ are the restrictions of $\pi, \alpha, \beta, \gamma, \prec$ to P', and $L' = L$ if $i \in \beta$ and $L' = L \cup \{\pi(i)\}$ otherwise, and $R' = R$ if $i \in \alpha$ and $R' = R \cup \{\pi(i)\}$ otherwise.
$\oplus$	$s', s'' \xrightarrow{\oplus} s$ is a transition if $P' \cap P'' = \emptyset$ (active colors should be disjoint), $L' \cap L'' = \emptyset$ (the minimal event of some process cannot belong to both subterms) and $R' \cap R'' = \emptyset$. Then, s is the disjoint union of s' and s'': $P = P' \uplus P''$, $\pi = \pi' \uplus \pi''$, $\alpha = \alpha' \uplus \alpha''$, $\beta = \beta' \uplus \beta''$, $\gamma = \gamma' \uplus \gamma''$, $L = L' \uplus L''$, $R = R' \uplus R''$ and $\prec$ is a (guessed) strict partial order on P satisfying (I_4b) and • additional ordering is guessed only between colored points of the left and right subterms: $\prec' = \prec \cap (P' \times P')$ and $\prec'' = \prec \cap (P'' \times P'')$, • the new sequence of factors on some process p is obtained by shuffling the sequences of factors of p of the subterms: for all $i, j \in P$ and $p \in \mathsf{Procs}$, if $i \in \alpha$ or $j \in \beta$ then ($i \lessdot_p j$ iff $i \lessdot'_p j$ or $i \lessdot''_p j$), • if for some process p, the minimal event of the global CBM occurs in the right subterm and its color has been forgotten ($p \in L''$), then we cannot insert a p-factor of the left subterm before the first p-factor of the right subterm (and similarly for the other cases): for all $i \in P'$, if $\pi(i) \in L''$ then $j \prec i$ for some $j \in P''$ with $\pi(j) = \pi(i)$, for all $i \in P''$, if $\pi(i) \in L'$ then $j \prec i$ for some $j \in P'$ with $\pi(j) = \pi(i)$, for all $i \in P'$, if $\pi(i) \in R''$ then $i \prec j$ for some $j \in P''$ with $\pi(j) = \pi(i)$, for all $i \in P''$, if $\pi(i) \in R'$ then $i \prec j$ for some $j \in P'$ with $\pi(j) = \pi(i)$.

Table 4.1: Transitions of $\mathcal{A}^{k\text{-stw}}_{\mathsf{cbm}}$ where $a \in \Sigma$, $p \in \mathsf{Procs}$, $d \in \mathsf{DS}$, $i, j \in [k]$, and states $s = (P, \pi, \alpha, \beta, \gamma, \prec, L, R)$, $s' = (P', \pi', \alpha', \beta', \gamma', \prec', L', R')$ and $s'' = (P'', \pi'', \alpha'', \beta'', \gamma'', \prec'', L'', R'')$.

when $\mathcal{E}'_p = \emptyset$. We assume now that $\mathcal{E}'_p \neq \emptyset \neq \mathcal{E}''_p$. We can prove that each p-factor, i.e., each $\rightarrow$-connected component of $\mathcal{E}_p$, has at least one endpoint colored. Since $\prec$ is a total order on $\pi^{-1}(p)$, it induces a total order on the factors: $w_1, \ldots, w_k$. Let $e_1, f_1, \ldots, e_k, f_k$ be the left and right endpoints of $w_1, \ldots, w_k$. Let $i_1, j_1, \ldots, i_k, j_k$ be the colors of $e_1, f_1, \ldots, e_k, f_k$. We have $j_\ell \prec_p i_{\ell+1}$ and $j_\ell \notin \alpha$ for $1 \leq \ell < k$. Therefore, $j_\ell \rightsquigarrow i_{\ell+1}$ and $f_\ell \dashrightarrow e_{\ell+1}$. We deduce that $(\mathcal{E}_p, \rightarrow \cup \dashrightarrow, \lambda)$ is the word $w_1 w_2 \cdots w_k$. Also, $p \in L'$ iff $e_1 \in \mathcal{E}'_p$ and e_1 is not colored. Similarly, $p \in L''$ iff $e_1 \in \mathcal{E}''_p$ and e_1 is not colored. Therefore, $p \in L = L' \cup L''$ iff e_1 is not colored. Similarly, $p \in P = R' \cup R''$ iff f_k is not colored.

We prove now that the relation $R = \rightarrow \cup \dashrightarrow \cup \rhd$ is acyclic. Notice that $e \rightarrow f$ in G_τ iff $e \rightarrow f$ in either $G_{\tau'}$ or $G_{\tau''}$. The same holds for $\rhd$. Moreover, the relation $\rightarrow \cup \rhd$ is acyclic both in $G_{\tau'}$ and in $G_{\tau''}$. Hence, if the relation R admits a cycle in G_τ, it must use some $\dashrightarrow$-edges:

$$e_1 \dashrightarrow f_1 (\rightarrow \cup \rhd)^* e_2 \dashrightarrow f_2 \cdots e_k \dashrightarrow f_k (\rightarrow \cup \rhd)^* e_1 \,.$$

By definition of $\dashrightarrow$ we have $e_1, f_1, \ldots, e_k, f_k \in \chi(P)$. Let $i_1, j_1, \ldots, i_k, j_k \in P$ such that $\chi(i_\ell) = e_\ell$ and $\chi(j_\ell) = f_\ell$ for $1 \leq \ell \leq k$. By definition of $\dashrightarrow$ and using (I_4a) we deduce that $i_1 \prec j_1 \preceq i_2 \prec j_2 \cdots i_k \prec j_k \preceq i_1$, a contradiction with $\prec$ acyclic.

This concludes the proof that (I_5) holds for (s, τ).

A state $s = (P, \pi, \alpha, \beta, \gamma, \prec, L, R)$ of $\mathcal{A}^{k\text{-stw}}_{\mathsf{cbm}}$ is accepting if $P = \emptyset$. It follows that if a binary tree τ is accepted by $\mathcal{A}^{k\text{-stw}}_{\mathsf{cbm}}$ then τ is a k-STT and $[\![\tau]\!] = (G, \chi)$ is a split-CBM with $\dashrightarrow$ defined as in (I_5). From the definition of accepting states, we deduce that $P = \emptyset$, and $\dashrightarrow = \emptyset$. Therefore, G is a CBM and $\mathsf{dom}(\chi) = P = \emptyset$.

There are some legal k-STTs denoting CBMs that are not accepted by the above automaton. For instance, the term

$$\tau = \mathsf{Add}^{\rightarrow}_{i,j}\, \mathsf{Add}^{\rightarrow}_{i,j}((i, a, p) \oplus (j, b, p))$$

is not accepted because the automaton prevents adding twice the same edge to the graph. To circumvent this restriction, the automaton may additionally store a relation $\rightarrow \subseteq P^2$ such that $i \rightarrow j$ iff $\chi(i) \rightarrow \chi(j)$. Then, a transition $\mathsf{Add}^{\rightarrow}_{i,j}$ is possible if either $i \rightarrow j$ or $i \rightsquigarrow j$.

Similarly, by keeping for each data-structure $d \in \mathsf{DS}$ a relation $\rhd^d \subseteq P^2$ such that $i \rhd^d j$ iff $\chi(i) \rhd^d \chi(j)$, the tree automaton may allow adding several times a same data-structure edge. ∎

Exercise 4.39. Prove that, if τ is a k-STT and $[\![\tau]\!] = (G_\tau, \chi_\tau)$ is such that G_τ is a CBM and $\mathsf{dom}(\chi_\tau) = \emptyset$ then τ is accepted by the tree automaton $\mathcal{A}^{k\text{-stw}}_{\mathsf{cbm}}$ constructed in the proof of Proposition 4.38.

Hint: The only non-determinism in the tree automaton $\mathcal{A}^{k\text{-stw}}_{\mathsf{cbm}}$ occurs during $\oplus$-transitions, when a strict partial order $\prec$ is guessed. When τ is a k-STT such that G_τ is a CBM, we can resolve the non-determinism of $\mathcal{A}^{k\text{-stw}}_{\mathsf{cbm}}$ by choosing the order induced by G_τ: if τ' is a subterm of τ then $G_{\tau'}$ is a subgraph of G_τ and the ordering $\prec$ guessed by $\mathcal{A}_{\mathsf{cbm}}^{k\text{-}\mathsf{stw}}$ at τ' should be $i \prec j$ iff $\chi_{\tau'}(i) < \chi_{\tau'}(j)$ in G_τ. $\diamondsuit$

Exercise 4.40. Modify the automaton constructed in the proof of Proposition 4.38 in order to check that data-structures $d \in \mathsf{Stacks}$ follow the LIFO policy and data-structures $d \in \mathsf{Queues}$ follow the FIFO policy.

Hint: For a data-structure $d \in \mathsf{DS}$, store a relation $R_d \subseteq (P \cup \mathsf{Procs})^2$ with the invariant defined below. For each event $e \in \mathcal{E}$, define $\zeta(e) = \mathsf{pid}(e)$ if there is no active color i such that $\chi(i) \rightarrow^* e$ and let $\zeta(e)$ be the maximal such i otherwise. The invariant is $R_d = \{(\zeta(e), \zeta(f)) \mid e \rhd^d f\}$. When taking a $\oplus$-transition, make sure that the policy of d is respected. $\diamondsuit$

Proposition 4.41. Given $\mathcal{S} \in \mathsf{CPDS}(\mathfrak{A}, \Sigma)$, we can construct a tree automaton $\mathcal{A}_{\mathcal{S}}^{k\text{-}\mathsf{stw}}$ of size $|\mathcal{S}|^{\mathcal{O}(k+|\mathsf{Procs}|)}$ such that for all k-STT $\tau \in L(\mathcal{A}_{\mathsf{cbm}}^{k\text{-}\mathsf{stw}})$, we have

$\tau \in L(\mathcal{A}_{\mathcal{S}}^{k\text{-}\mathsf{stw}})$ iff G_τ is accepted by the CPDS $\mathcal{S}$.

Proof. The automaton $\mathcal{A}_{\mathcal{S}}^{k\text{-}\mathsf{stw}}$ follows the characterization given by the formula $\Phi_{\mathcal{S}}$ of Theorem 3.4. We also use the notations introduced in the proof of this theorem.

A state of $\mathcal{A}_{\mathcal{S}}^{k\text{-}\mathsf{stw}}$ is a tuple $s = (P, \pi, \alpha, \beta, \delta, \sigma)$ where P, π, α, β are as in the proof of Proposition 4.38, δ stores the transitions that are guessed for the events associated with active colors, and σ stores the (partial) global final state. More precisely, when reading bottom-up a term $\tau \in L(\mathcal{A}_{\mathsf{cbm}}^{k\text{-}\mathsf{stw}})$, the tree automaton $\mathcal{A}_{\mathcal{S}}^{k\text{-}\mathsf{stw}}$ reaches a state s satisfying the following conditions with $[\![\tau]\!] = (G, \chi)$:

(J_1) $P = \mathsf{dom}(\chi) \subseteq [k]$ is the set of active colors,

(J_2) $\pi \colon P \to \mathsf{Procs}$ gives the associated processes: $\pi(i) = \mathsf{pid}(\chi(i))$ for all $i \in P$,

(J_3) $\alpha, \beta \subseteq P$ are such that

- $i \in \alpha$ iff $\chi(i)$ is the source of a $\rightarrow$-edge in G.
- $i \in \beta$ iff $\chi(i)$ is the target of a $\rightarrow$-edge in G.

(J_4) $\delta \colon P \to \mathsf{Trans}$ defines for each active color $i \in P$ the transition $\delta(i)$ guessed for the event $\chi(i)$.

(J_5) $\sigma \colon \mathsf{Procs} \to \mathsf{Locs}$ is a partial map which collects the global final state: when the color i of the maximal event of some process p is forgotten, we store the target state of $\delta(i)$ in $\sigma(p)$.

The number of states of $\mathcal{A}_{\mathcal{S}}^{k\text{-}\mathsf{stw}}$ is $|\mathcal{S}|^{\mathcal{O}(k+|\mathsf{Procs}|)}$.

The transitions of $\mathcal{A}_{\mathcal{S}}^{k\text{-}\mathsf{stw}}$ are given in Table 4.2.

A state s of $\mathcal{A}_{\mathcal{S}}^{k\text{-}\mathsf{stw}}$ is accepting if $P = \emptyset$ and $\bar{\sigma} \in F$ is an accepting global state of $\mathcal{S}$, where $\bar{\sigma}$ completes the partial global state σ with the initial location ℓ_{in} for processes having no events in the CBM: $\bar{\sigma}(p) = \sigma(p)$ if $p \in \mathsf{dom}(\sigma)$ and $\bar{\sigma}(p) = \ell_{\text{in}}$ otherwise. ∎

(i,a,p)	$\perp \xrightarrow{(i,a,p)} s$ is a transition if $P = \{i\}$, $\pi(i) = p$, $\alpha = \beta = \emptyset$, $\delta(i) \in \mathsf{Trans}(p,a)$ and $\mathsf{dom}(\sigma) = \emptyset$.
$\mathsf{Add}^{\rightarrow}_{i,j}$	$s \xrightarrow{\mathsf{Add}^{\rightarrow}_{i,j}} s'$ is a transition if $i,j \in P$ and $\mathsf{tgt}(\delta(i)) = \mathsf{src}(\delta(j))$. Then, $P' = P$, $\pi' = \pi$, $\delta' = \delta$, $\sigma' = \sigma$, $\alpha' = \alpha \cup \{i\}$, $\beta' = \beta \cup \{j\}$.
$\mathsf{Add}^{d}_{i,j}$	$s \xrightarrow{\mathsf{Add}^{d}_{i,j}} s'$ is a transition if $i,j \in P$ and for some $v \in \mathsf{Val}$ we have $\delta(i) \in \mathsf{Send}(d,v)$ and $\delta(j) \in \mathsf{Receive}(d,v)$. Then, $s' = s$.
Forget_i	$s \xrightarrow{\mathsf{Forget}_i} s'$ is a transition if $i \in P$ and $i \in \beta \vee \mathsf{src}(\delta(i)) = \ell_{\text{in}}$. Then, $P' = P \setminus \{i\}$ and $\pi', \alpha', \beta', \delta'$ are the restrictions of $\pi, \alpha, \beta, \delta$ to P', and $\sigma' = \sigma$ if $i \in \alpha$ and $\sigma' = \sigma[\pi(i) \mapsto \mathsf{tgt}(\delta(i))]$ otherwise.
$\oplus$	$s', s'' \xrightarrow{\oplus} s$ is a transition if $P' \cap P'' = \emptyset$ and $\mathsf{dom}(\sigma') \cap \mathsf{dom}(\sigma'') = \emptyset$ (this condition is actually always satisfied for terms accepted by $\mathcal{A}^{k\text{-stw}}_{\mathsf{cbm}}$). Then, s is the disjoint union of s' and s'': $P = P' \uplus P''$, $\pi = \pi' \uplus \pi''$, $\alpha = \alpha' \uplus \alpha''$, $\beta = \beta' \uplus \beta''$, $\delta = \delta' \uplus \delta''$ and $\sigma = \sigma' \uplus \sigma''$.

Table 4.2: Transitions of $\mathcal{A}^{k\text{-stw}}_{\mathcal{S}}$ where $a \in \Sigma$, $p \in \mathsf{Procs}$, $d \in \mathsf{DS}$, $i,j \in [k]$, $s = (P,\pi,\alpha,\beta,\delta,\sigma)$, $s' = (P',\pi',\alpha',\beta',\delta',\sigma')$ and $s'' = (P'',\pi'',\alpha'',\beta'',\delta'',\sigma'')$.

Corollary 4.42. The problem STW-NONEMPTINESS$(\mathfrak{A},\Sigma)$ is decidable in EXPTIME. The procedure is only polynomial in the size of the CPDS.

Proof. The problem reduces to checking nonemptiness of a tree automaton. Given $k > 0$ and $\mathcal{S} \in \mathsf{CPDS}(\mathfrak{A},\Sigma)$, we have $L(\mathcal{S}) \cap \mathsf{CBM}^{k\text{-stw}} \neq \emptyset$ iff $L(\mathcal{A}^{k\text{-stw}}_{\mathsf{cbm}} \cap \mathcal{A}^{k\text{-stw}}_{\mathcal{S}}) \neq \emptyset$. Indeed, given $\mathcal{M} \in L(\mathcal{S}) \cap \mathsf{CBM}^{k\text{-stw}}$, we find $\tau \in L(\mathcal{A}^{k\text{-stw}}_{\mathsf{cbm}})$ with $G_\tau = \mathcal{M}$ by Proposition 4.38. We obtain $\tau \in L(\mathcal{A}^{k\text{-stw}}_{\mathcal{S}})$ by Proposition 4.41. Conversely, if $\tau \in L(\mathcal{A}^{k\text{-stw}}_{\mathsf{cbm}} \cap \mathcal{A}^{k\text{-stw}}_{\mathcal{S}})$ then $G_\tau \in \mathsf{CBM}^{k\text{-stw}}$ by Proposition 4.38 and $G_\tau \in L(\mathcal{S})$ by Proposition 4.41. ∎

Proposition 4.43. Given $k > 0$ and an MSO formula $\varphi \in \mathsf{MSO}(\mathfrak{A},\Sigma)$, we can construct a tree automaton $\mathcal{A}^{k\text{-stw}}_{\varphi}$ such that for all k-STT τ we have

$$\tau \in L(\mathcal{A}^{k\text{-stw}}_{\varphi}) \qquad \text{iff} \qquad G_\tau \models \varphi\,.$$

Proof. From $\varphi \in \mathsf{MSO}(\mathfrak{A},\Sigma)$, we construct $\tilde{\varphi}^k \in \mathsf{MSO}(\Lambda^k, \downarrow_0, \downarrow_1)$ using Proposition 4.26. Using [TW68] we obtain an equivalent tree automaton $\mathcal{A}^{k\text{-stw}}_{\varphi}$. ∎

Corollary 4.44. The problem STW-MODELCHECKING$(\mathfrak{A},\Sigma)$ is decidable. The procedure is only polynomial in the size of the CPDS.

Proof. The problem reduces to checking emptiness of a tree automaton. Given $k > 0$, a CPDS $\mathcal{S}$, and a formula $\varphi \in \mathsf{MSO}(\mathfrak{A},\Sigma)$ we have $L(\mathcal{S}) \cap \mathsf{CBM}^{k\text{-stw}} \subseteq L(\varphi)$ iff $L(\mathcal{A}^{k\text{-stw}}_{\mathsf{cbm}} \cap \mathcal{A}^{k\text{-stw}}_{\mathcal{S}} \cap \mathcal{A}^{k\text{-stw}}_{\neg\varphi}) = \emptyset$. ∎

CHAPTER 5

Propositional Dynamic Logic

6th Lecture

5.1 Propositional Dynamic Logic

Syntax:

Let AP be a set of atomic propositions (node labels) and Γ be a set of atomic programs (edge labels). The syntax of $\mathsf{ICPDL}(\mathsf{AP}, \Gamma)$ is given by

$$\begin{aligned}
\Phi &::= \mathsf{E}\sigma \mid \Phi \vee \Phi \mid \neg\Phi \\
\sigma &::= p \mid \sigma \vee \sigma \mid \neg\sigma \mid \langle\pi\rangle\sigma \qquad\qquad \mid \mathsf{Loop}\langle\pi\rangle \\
\pi &::= \gamma \mid \mathsf{test}(\sigma) \mid \pi + \pi \mid \pi \cdot \pi \mid \pi^* \quad \mid \pi^{-1} \qquad\qquad \mid \pi \cap \pi
\end{aligned}$$

where $p \in \mathsf{AP}$, $\gamma \in \Gamma$. We call

- Φ a sentence,
- σ a *state formula* or *node formula*, and
- π a *program*, or *path formula.*

If intersection $\pi \cap \pi$ is not allowed, the fragment is PDL with loop and converse (LCPDL). If neither intersection nor loop are allowed, the fragment is PDL with converse (CPDL). If backward paths π^{-1} are not allowed the fragment is called PDL with intersection (IPDL). In simple PDL neither backwards paths nor intersections nor loops are allowed.

Semantics of ICPDL formulas:

Let $G = (V, (E_\gamma)_{\gamma\in\Gamma}, \lambda)$ be a $(2^{\mathsf{AP}}, \Gamma)$-labeled graph: $\lambda\colon V \to 2^{\mathsf{AP}}$ and $E_\gamma \subseteq V^2$.

We write $G \models \mathsf{E}\sigma$ if $[\![\sigma]\!]_G \neq \emptyset$.

The semantics $[\![\sigma]\!]_G \subseteq V$ of state fomulas is given below.

For $\Phi \in \mathsf{ICPDL}(\mathsf{AP}, \Gamma)$, we let $L(\Phi) := \{G \mid G \models \Phi\}$.

Semantics of state formulas:

The semantics of a state formula σ wrt. G is a set $[\![\sigma]\!]_G \subseteq V$, inductively defined below. We also write $G, e \models \sigma$ for $v \in [\![\sigma]\!]_G$.

$$
\begin{aligned}
[\![p]\!]_G &:= \{e \in V \mid p \in \lambda(e)\} \\
[\![\sigma_1 \vee \sigma_2]\!]_G &:= [\![\sigma_1]\!]_G \cup [\![\sigma_2]\!]_G \\
[\![\neg\sigma]\!]_G &:= V \setminus [\![\sigma]\!]_G \\
[\![\langle\pi\rangle\sigma]\!]_G &:= \{e \in V \mid \text{there is } f \in [\![\sigma]\!]_G \text{ such that } (e, f) \in [\![\pi]\!]_G\} \\
[\![\mathsf{Loop}\langle\pi\rangle]\!]_G &:= \{e \in V \mid (e, e) \in [\![\pi]\!]_G\}
\end{aligned}
$$

Semantics of path formulas:

The semantics of a path formula π wrt. G is a set $[\![\pi]\!]_G \subseteq V \times V$, inductively defined below. We also write $G, e, f \models \pi$ for $(e, f) \in [\![\pi]\!]_G$.

$$
\begin{aligned}
[\![\gamma]\!]_G &:= E_\gamma \\
[\![\mathsf{test}(\sigma)]\!]_G &:= \{(e, e) \mid e \in [\![\sigma]\!]_G\} \\
[\![\pi^{-1}]\!]_G &:= [\![\pi]\!]_G^{-1} = \{(f, e) \mid (e, f) \in [\![\pi]\!]_G\} \\
[\![\pi_1 + \pi_2]\!]_G &:= [\![\pi_1]\!]_G \cup [\![\pi_2]\!]_G \qquad [\![\pi_1 \cap \pi_2]\!]_G := [\![\pi_1]\!]_G \cap [\![\pi_2]\!]_G \\
[\![\pi_1 \cdot \pi_2]\!]_G &:= [\![\pi_1]\!]_G \circ [\![\pi_2]\!]_G \\
&= \{(e, g) \in V \times V \mid \exists f \in V : (e, f) \in [\![\pi_1]\!]_G \text{ and } (f, g) \in [\![\pi_2]\!]_G\} \\
[\![\pi^*]\!]_G &:= [\![\pi]\!]_G^* = \textstyle\bigcup_{n\in\mathbb{N}} [\![\pi]\!]_G^n
\end{aligned}
$$

One can show that ICPDL is no more expressive than MSO:

Exercise 5.1. Show that, for every $\mathsf{ICPDL}(\mathsf{AP}, \Gamma)$ sentence Φ, state formula σ and path formula π, we can construct *equivalent* $\mathsf{MSO}(\mathsf{AP}, \Gamma)$ formulas $\overline{\Phi}$, $\overline{\sigma}(x)$ and $\overline{\pi}(x, y)$ where $\overline{\Phi}$ is a sentence, $\mathsf{Free}(\overline{\sigma}) = \{x\}$ and $\mathsf{Free}(\overline{\pi}) = \{x, y\}$. $\diamondsuit$

Notice that MSO is strictly more powerful than ICPDL. Indeed, we cannot express that a graph is connected in ICPDL. Also, the modality U_2 defined below checks a path of even length and therefore cannot be expressed in FO.

Over message passing automata ($\mathsf{DS} = \mathsf{Queues}$) it was shown very recently (CONCUR 2018) that $\mathsf{FO}(<, \rightarrow, \rhd)$ has the same expressive power as the starfree fragment of LCPDL.

For CBMs over $(\mathfrak{A}, \Sigma)$, we use the set $\mathsf{AP} = \mathsf{Procs} \cup \Sigma$ of atomic propositions and the set $\Gamma = \{\rightarrow\} \cup \mathsf{DS}$ of atomic programs. We then also write $(\mathsf{ILC})\mathsf{PDL}(\mathfrak{A}, \Sigma)$.

Example 5.2. Consider the following abbreviation/examples:

- $\mathsf{A}\sigma \;=\; \neg\mathsf{E}\neg\sigma$ $\qquad$ (ICPDL formula)
- $[\pi]\sigma \;=\; \neg\langle\pi\rangle\neg\sigma$ $\qquad$ (state formula)
- $\mathsf{true} \;=\; p \vee \neg p$ $\qquad$ (state formula)
- $\sigma_1 \,\mathsf{U}\, \sigma_2 \;=\; \langle(\mathsf{test}(\sigma_1) \cdot \rightarrow)^*\rangle\sigma_2$ $\qquad$ (state formula)
- $\sigma_1 \,\mathsf{U}_2\, \sigma_2 \;=\; \langle(\mathsf{test}(\sigma_1) \cdot \rightarrow \cdot\, \mathsf{test}(\sigma_1) \cdot \rightarrow)^*\rangle\sigma_2$ $\qquad$ (state formula)
- $\rhd = \sum_{d \in \mathsf{DS}} \rhd^d$ $\qquad$ (path formula)
- $\mathsf{write} = \langle\rhd\rangle\mathsf{true}$ and $\mathsf{read} = \langle\rhd^{-1}\rangle\mathsf{true}$ $\qquad$ (state formulas)
- $\Phi_1 = \mathsf{A}\big(a \Rightarrow \langle(\rightarrow + \rhd)^*\rangle b\big) \in \mathsf{PDL}(\mathfrak{A}, \Sigma)$
- $\mathsf{req\text{-}ack} = (\mathsf{test}(p_1) \cdot \rhd^{c_1} \cdot \rightarrow \cdot \rhd^{c_2}) \;\;+\;\; (\mathsf{test}(p_1) \cdot \rhd^{c_1} \cdot \rightarrow \cdot \rhd^{s} \cdot \rightarrow \cdot \rhd^{c_2})$
(path formula; cf. client-server system from previous chapter)
- $\Phi_2 = \mathsf{A}(a \Rightarrow [\mathsf{req\text{-}ack}]a) \;\wedge\; \mathsf{A}(b \Rightarrow [\mathsf{req\text{-}ack}]b) \quad \in \mathsf{PDL}(\mathfrak{A}, \Sigma)$
$\equiv \mathsf{A}([\mathsf{test}(a) \cdot \mathsf{req\text{-}ack}]a) \;\wedge\; \mathsf{A}([\mathsf{test}(b) \cdot \mathsf{req\text{-}ack}]b) \quad \in \mathsf{PDL}(\mathfrak{A}, \Sigma)$
$\equiv \forall x, y\Big(\mathit{req\text{-}ack}(x, y) \Rightarrow \big((a(x) \wedge a(y)) \vee (b(x) \wedge b(y))\big)\Big) \;\in \mathsf{MSO}(\mathfrak{A}, \Sigma)$

We have $L(\mathcal{S}_{\mathrm{cs}}) \not\subseteq L(\Phi_1)$ and $L(\mathcal{S}_{\mathrm{cs}}) \subseteq L(\Phi_2)$. $\Diamond$

Example 5.3. Let $\Gamma = \{\rightarrow\} \cup \mathsf{DS}$. Most conditions ensuring that a labeled graph $G = (\mathcal{E}, \rightarrow, (\rhd^d)_{d \in \mathsf{DS}}, \mathsf{pid}, \lambda)$ is in fact a CBM can be expressed in LCPDL. Missing is a formula stating that on each $\mathcal{E}_p$, $\rightarrow$ is the successor relation of a total order.

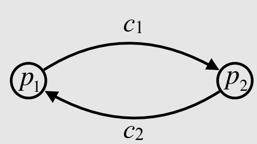
- $\mathsf{LABELS} = \mathsf{A}\Big(\bigvee_{p \in \mathsf{Procs}} \big(p \wedge \bigwedge_{p' \neq p} \neg p'\big) \wedge \bigvee_{a \in \Sigma} \big(a \wedge \bigwedge_{a' \neq a} \neg a'\big)\Big)$
- $\mathsf{ORDER} = \mathsf{A}\neg\mathsf{Loop}\langle(\rightarrow + \rhd)^+\rangle$
- $\mathsf{PROCESS} = \mathsf{A}\big(\mathsf{test}(\langle\rightarrow\rangle) \implies \bigvee_{p \in \mathsf{Procs}}(p \wedge \langle\rightarrow\rangle p)\big)$
- $\mathsf{WRITER} = \mathsf{A} \bigwedge_{d \in \mathsf{DS}} \mathsf{test}(\langle\rhd^d\rangle) \implies \mathsf{Writer}(d)$
- $\mathsf{READER} = \mathsf{A} \bigwedge_{d \in \mathsf{DS}} \mathsf{test}(\langle(\rhd^d)^{-1}\rangle) \implies \mathsf{Reader}(d)$
- $\mathsf{DISJOINT} = \neg\mathsf{E}\Big(\langle\rhd \cdot \rhd\rangle \vee \bigvee_{d \neq d'} \big(\langle\rhd^d\rangle \wedge \langle\rhd^{d'}\rangle\big) \vee \big(\langle(\rhd^d)^{-1}\rangle \wedge \langle(\rhd^{d'})^{-1}\rangle\big)\Big)$
$\wedge \neg\mathsf{E}\Big(\bigvee_{d \in \mathsf{DS}} \mathsf{Loop}\langle(\rhd^d \cdot \rightarrow^+ \cdot (\rhd^d)^{-1}) + ((\rhd^d)^{-1} \cdot \rightarrow^+ \cdot \rhd^d\rangle)\Big)$
- $\mathsf{FIFO} = \mathsf{A} \bigwedge_{c \in \mathsf{Queues}} \neg\mathsf{Loop}\langle\rhd^c \cdot \rightarrow^+ \cdot (\rhd^c)^{-1} \cdot \rightarrow^+\rangle$
- $\mathsf{LIFO} = \mathsf{A} \bigwedge_{s \in \mathsf{Stacks}} \Big(\mathsf{test}(\langle\rhd^s\rangle) \implies$
$\mathsf{Loop}\langle\rightarrow \cdot \big(\rhd^s \cdot \rightarrow + \mathsf{test}(\neg\langle\rhd^s + (\rhd^s)^{-1}\rangle) \cdot \rightarrow\big)^* \cdot (\rhd^s)^{-1}\rangle\Big)$ $\Diamond$

For an architecture $\mathfrak{A}$ and an alphabet Σ, consider the following problems:

(ILC)PDL-SATISFIABILITY($\mathfrak{A}, \Sigma$):	
Instance:	$\Phi \in (\mathsf{ILC})\mathsf{PDL}(\mathfrak{A}, \Sigma)$
Question:	$L(\Phi) \neq \emptyset$?

(ILC)PDL-MODELCHECKING($\mathfrak{A}, \Sigma$):	
Instance:	$\mathcal{S} \in \mathsf{CPDS}(\mathfrak{A}, \Sigma)$; $\Phi \in (\mathsf{ILC})\mathsf{PDL}(\mathfrak{A}, \Sigma)$
Question:	$L(\mathcal{S}) \subseteq L(\Phi)$?

Theorem 5.4. *Let $\mathfrak{A}$ be given as follows (and Σ be arbitrary):*



Then, all the abovementioned problems are undecidable.

By Theorem 5.42, we obtain the following positive result:

Theorem 5.5. *Suppose* $\mathsf{DS} = \mathsf{Bags}$. *Then, the problems*

PDL-SATISFIABILITY($\mathfrak{A}, \Sigma$) *and*

PDL-MODELCHECKING($\mathfrak{A}, \Sigma$)

are both decidable.

We have seen in Section 4.7 that a $(2^{\mathsf{AP}}, \Gamma)$-labeled graph G with special tree-width at most k can be interpreted in any k-STT τ such that $[\![\tau]\!] = (G, \chi)$. We prove now an analog of Proposition 4.26 for PDL.

We define $\mathsf{AP}^k = \mathsf{AP} \cup \{\oplus, \mathsf{color}_i, \mathsf{Add}^{\gamma}_{i,j}, \mathsf{Forget}_i \mid i, j \in [k], \gamma \in \Gamma\}$

Proposition 5.6 (PDL interpretation). For all sentences $\Phi \in \mathsf{ICPDL}(\mathsf{AP}, \Gamma)$ and all $k > 0$, we can construct a sentence $\widetilde{\Phi}^k \in \mathsf{ICPDL}(\mathsf{AP}^k, \downarrow_0, \downarrow_1)$ of size $\mathcal{O}(k^2|\Phi|)$ such that, for every valid k-STT τ with $[\![\tau]\!] = (G, \chi)$, we have

$$G \models \Phi \quad \text{iff} \quad \tau \models \widetilde{\Phi}^k \,.$$

Moreover, if $\Phi \in \mathsf{LCPDL}(\mathsf{AP}, \Gamma)$ then we can construct $\widetilde{\Phi}^k \in \mathsf{LCPDL}(\mathsf{AP}^k, \downarrow_0, \downarrow_1)$.

Proof. We prove by induction that for all $\mathsf{ICPDL}(\mathsf{AP}, \Gamma)$ sentences Φ, state formulas σ and path formulas π, and all $k > 0$, there are $\mathsf{ICPDL}(\mathsf{AP}^k, \downarrow_0, \downarrow_1)$ formulas $\widetilde{\Phi}^k$, $\widetilde{\sigma}^k$ and $\widetilde{\pi}^k$ such that, for all valid k-STT τ with $[\![\tau]\!] = (G, \chi)$, and all vertices e, f of G we have

$$\begin{aligned} G \models \Phi \quad &\text{iff} \quad \tau \models \widetilde{\Phi}^k \\ G, e \models \sigma \quad &\text{iff} \quad \tau, e \models \widetilde{\sigma}^k \\ G, e, f \models \pi \quad &\text{iff} \quad \tau, e, f \models \widetilde{\pi}^k \end{aligned}$$

The difficult case is to translate the edge relations. We define

$$\widetilde{\gamma}^k = \sum_{1 \le i,j \le k} \mathsf{test}_i \cdot (\mathsf{test}(\neg\,\mathsf{Forget}_i) \cdot \uparrow)^+ \cdot \mathsf{test}(\mathsf{Add}^{\gamma}_{i,j}) \cdot (\mathsf{test}(\neg\,\mathsf{Forget}_j) \cdot \downarrow)^+ \cdot \mathsf{test}_j \,.$$

where $\downarrow = \downarrow_0 + \downarrow_1$, $\uparrow = \downarrow^{-1}$ and $\mathsf{test}_i = \mathsf{test}(\mathsf{color}_i)$.

The formula $\widetilde{\gamma}^k$ is of size $\mathcal{O}(k^2)$. The other cases are trivial:

$$\begin{aligned}
\widetilde{\mathsf{E}\sigma}^k &= \mathsf{E}(\neg\langle\downarrow\rangle \wedge \widetilde{\sigma}^k) & \widetilde{\neg\Phi}^k &= \neg\,\widetilde{\Phi}^k & \widetilde{\Phi_1 \vee \Phi_2}^k &= \widetilde{\Phi_1}^k \vee \widetilde{\Phi_2}^k \\
\widetilde{p}^k &= p & \widetilde{\neg\sigma}^k &= \neg\,\widetilde{\sigma}^k & \widetilde{\sigma_1 \vee \sigma_2}^k &= \widetilde{\sigma_1}^k \vee \widetilde{\sigma_2}^k \\
\widetilde{X_\ell}^k &= X_\ell & \widetilde{\langle\pi\rangle\sigma}^k &= \langle\widetilde{\pi}^k\rangle\widetilde{\sigma}^k & \widetilde{\mathsf{Loop}\langle\pi\rangle}^k &= \mathsf{Loop}\langle\widetilde{\pi}^k\rangle \\
\widetilde{\mathsf{test}(\sigma)}^k &= \mathsf{test}(\widetilde{\sigma}^k) & \widetilde{\pi^{-1}}^k &= (\widetilde{\pi}^k)^{-1} & \widetilde{\pi_1 + \pi_2}^k &= \widetilde{\pi_1}^k + \widetilde{\pi_2}^k \\
\widetilde{\pi_1 \cdot \pi_2}^k &= \widetilde{\pi_1}^k \cdot \widetilde{\pi_2}^k & \widetilde{\pi^*}^k &= (\widetilde{\pi}^k)^* & \widetilde{\pi_1 \cap \pi_2}^k &= \widetilde{\pi_1}^k \cap \widetilde{\pi_2}^k
\end{aligned}$$

This concludes the proof. ∎

Exercise 5.7. Write a sentence $\Phi^k_{\mathsf{valid}} \in \mathsf{PDL}(\mathsf{AP}^k, \downarrow_0, \downarrow_1)$ stating that the binary tree is a valid k-STT. What is the size of Φ^k_{valid}? ◊

Theorem 5.8 (Göller, Lohrey, and Lutz [GLL09]). *For a given formula* $\varphi \in \mathsf{LCPDL}(\mathsf{AP}, \downarrow_0, \downarrow_1)$ *over binary trees, we can construct a tree automaton* $\mathcal{B}_\varphi$ *of size* $2^{\mathit{poly}(|\varphi|)}$ *such that, for every binary tree* τ*, we have*

$$\tau \models \varphi \quad \text{iff} \quad \tau \in L(\mathcal{B}_\varphi)\,.$$

Moreover, if $\varphi \in \mathsf{ICPDL}(\mathsf{AP}, \downarrow_0, \downarrow_1)$ *then we can construct an equivalent* $\mathcal{B}_\varphi$ *of double exponential size.*

Proof. We construct an *alternating two-way (tree) automata* (A2As) which is equivalent to φ. An A2A "walks" in a tree, similarly to a path formula from CPDL. In addition, it can spawn several copies of an automaton, which *all* have to accept the input. This spawning is dual to non-deterministic choice, hence the name *alternating*. If φ is an LCPDL formula, then the A2A is of size polynomial in φ.

Next, we construct a non-deterministic tree automaton equivalent to the A2A associated with φ. This induces an exponential blow-up. Hence, the resulting automaton is of size $2^{\mathsf{poly}(|\varphi|)}$.

If φ is in ICPDL then the construction of the corresponding A2A is exponential, resulting in $\mathcal{B}_\varphi$ of double exponential size. ∎

Corollary 5.9. Given $\varphi \in \mathsf{LCPDL}(\mathsf{AP}, \Gamma)$ and $k > 0$, we can construct a tree automaton $\mathcal{A}_\varphi^{k\text{-}\mathsf{stw}}$ of size $2^{\mathsf{poly}(k,|\varphi|)}$ such that, for every valid k-STT τ with $[\![\tau]\!] = (G, \chi)$, we have

$$G \models \varphi \quad \text{iff} \quad \tau \in L(\mathcal{A}_\varphi^{k\text{-}\mathsf{stw}})\,.$$

Moreover, if $\varphi \in \mathsf{ICPDL}(\mathsf{AP}, \Gamma)$ then we can construct an equivalent $\mathcal{A}_\varphi^{k\text{-}\mathsf{stw}}$ of double exponential size.

Proof. Given $\varphi \in \mathsf{LCPDL}(\mathsf{AP}, \Gamma)$ and $k > 0$, we construct using Proposition 5.6 the corresponding formula $\widetilde{\varphi}^k$ of size $\mathcal{O}(k^2|\varphi|)$. Then, we apply Theorem 5.8 to construct the automaton $\mathcal{A}_\varphi^{k\text{-}\mathsf{stw}} = \mathcal{B}_{\widetilde{\varphi}^k}$ of size $2^{\mathsf{poly}(k,|\varphi|)}$.

5.4 ICPDL model checking

In the previous sections, we showed that model checking CPDSs against MSO formulas is decidable when we restrict to behaviors of bounded special tree-width. However, the transformation of an MSO formula into a tree automaton is inherently non-elementary, and this non-elementary lower bound is in fact inherited by the model-checking problem. We will, therefore, turn to the logic ICPDL and consider the following problem, for a given architecture $\mathfrak{A}$ (and alphabet Σ):

STW-(ILC)PDL-MODELCHECKING($\mathfrak{A}, \Sigma$):	
Instance:	$\mathcal{S} \in \mathsf{CPDS}(\mathfrak{A}, \Sigma)$; $\Phi \in (\mathsf{ILC})\mathsf{PDL}(\mathfrak{A}, \Sigma)$; $k > 0$
Question:	$L(\mathcal{S}) \cap \mathsf{CBM}^{k\text{-}\mathsf{stw}} \subseteq L(\Phi)$?

Here, we suppose that k is given in unary.

Decidability of this problem follows from decidability of MSO and the statement of Exercise 5.1, saying that every ICPDL formula can be (effectively) translated into an MSO sentence. Unfortunately, this does not give us an elementary upper bound. Instead, we will use Corollary 5.9 in order to obtain the following result.

Theorem 5.10. *The problem* STW-LCPDL-MODELCHECKING($\mathfrak{A}, \Sigma$) *is in* EXPTIME *(when the bound k on the special tree-width is encoded in unary).*

The problem STW-ICPDL-MODELCHECKING($\mathfrak{A}, \Sigma$) *is solvable in doubly exponential time.*

This result was proved in [CGNK14] using split-width instead of special tree-width.

Proof. From Corollary 5.9 and Proposition 4.41 we construct the tree automata $\mathcal{A}_{\neg\Phi}^{k\text{-}\mathsf{stw}}$ and $\mathcal{A}_{\mathcal{S}}^{k\text{-}\mathsf{stw}}$. Recall also that we have a tree automaton $\mathcal{A}_{\mathsf{cbm}}^{k\text{-}\mathsf{stw}}$ from Proposition 4.38. We obtain

$$L(\mathcal{S}) \cap \mathsf{CBM}^{k\text{-}\mathsf{stw}} \subseteq L(\Phi) \text{ iff } L(\mathcal{A}_{\mathsf{cbm}}^{k\text{-}\mathsf{stw}} \cap \mathcal{A}_{\mathcal{S}}^{k\text{-}\mathsf{stw}} \cap \mathcal{A}_{\neg\Phi}^{k\text{-}\mathsf{stw}}) = \emptyset$$

and we conclude since emptiness of NTAs can be checked in polynomial time. ∎

5.5 Concrete Underapproximation Classes and Special Tree-Width

So far, we considered the following (decidable) version of the model-checking problem for CPDSs: Given a CPDS $\mathcal{S}$, a sentence φ in MSO or ICPDL, and $k > 0$, do we have

$$L(\mathcal{S}) \cap \mathsf{CBM}^{k\text{-}\mathsf{stw}} \subseteq L(\varphi)\,?$$

We will now study some other, more "concrete" families $\mathcal{C} = (\mathcal{C}_k)_{k \geq 0}$ that are

- monotone ($\mathcal{C}_k \subseteq \mathcal{C}_{k+1}$ for all $k \geq 0$),
- complete ($\bigcup_{k \geq 0} \mathcal{C}_k = \mathsf{CBM}$), and
- decidable (the usual decision problems are decidable when the domain of CBMs is restricted to $\mathcal{C}_k$).

In particular, the following model-checking problems for $\mathcal{C}$ should be decidable:

$\mathcal{C}$-MSO-ModelChecking$(\mathfrak{A}, \Sigma)$:	
Instance:	$\mathcal{S} \in \mathsf{CPDS}(\mathfrak{A}, \Sigma)$; $\varphi \in \mathsf{MSO}(\mathfrak{A}, \Sigma)$; $k \geq 0$
Question:	$L(\mathcal{S}) \cap \mathcal{C}_k \subseteq L(\varphi)\,?$

$\mathcal{C}$-(ILC)PDL-ModelChecking$(\mathfrak{A}, \Sigma)$:	
Instance:	$\mathcal{S} \in \mathsf{CPDS}(\mathfrak{A}, \Sigma)$; $\varphi \in (\mathrm{ILC})\mathrm{PDL}(\mathfrak{A}, \Sigma)$; $k \geq 0$
Question:	$L(\mathcal{S}) \cap \mathcal{C}_k \subseteq L(\varphi)\,?$

Next, we argue that, to show decidability of the above problem, we can make use of the previous results on special tree-width.

Consider a family $\mathcal{C} = (\mathcal{C}_k)_{k \geq 0}$ such that the following hold, for all $k \geq 0$:

(1) there is $k' \geq 0$ such that $\mathcal{C}_k \subseteq \mathsf{CBM}^{k'\text{-}\mathsf{stw}}$ (and k' is "easily" computable),

(2) one of the following is true:

  (a) there is $\mathcal{S}_k \in \mathsf{CPDS}$ such that $L(\mathcal{S}_k) = \mathcal{C}_k$.

  (b) there is $\varphi_k \in \mathsf{LCPDL}$ or ICPDL such that $L(\varphi_k) \cap \mathsf{CBM} = \mathcal{C}_k$, or

  (c) there is $\varphi_k \in \mathsf{MSO}$ such that $L(\varphi_k) = \mathcal{C}_k$, or

Then, we have

$$\{\tau \mid \tau \in k'\text{-}\mathsf{STT} \text{ and } G_\tau \in \mathcal{C}_k\} = L(\mathcal{A}^{k'\text{-}\mathsf{stw}}_{\mathsf{cbm}} \cap \mathcal{A}^{k'\text{-}\mathsf{stw}}_{\mathcal{S}_k}) = L(\mathcal{A}^{k'\text{-}\mathsf{stw}}_{\mathsf{cbm}} \cap \mathcal{A}^{k'\text{-}\mathsf{stw}}_{\varphi_k})$$

We deduce that the MSO or ICPDL model-checking problems are decidable due to the following equivalences:

$$\begin{aligned} L(\mathcal{S}) \cap \mathcal{C}_k \subseteq L(\varphi) \quad &\text{iff} \quad L(\mathcal{A}^{k'\text{-stw}}_{\mathsf{cbm}} \cap \mathcal{A}^{k'\text{-stw}}_{\mathcal{S}_k} \cap \mathcal{A}^{k'\text{-stw}}_{\mathcal{S}} \cap \mathcal{A}^{k'\text{-stw}}_{\neg\varphi}) = \emptyset \\ &\text{iff} \quad L(\mathcal{A}^{k'\text{-stw}}_{\mathsf{cbm}} \cap \mathcal{A}^{k'\text{-stw}}_{\varphi_k} \cap \mathcal{A}^{k'\text{-stw}}_{\mathcal{S}} \cap \mathcal{A}^{k'\text{-stw}}_{\neg\varphi}) = \emptyset \end{aligned}$$

Depending on the size of $\mathcal{S}_k \in \mathsf{CPDS}$ or $\varphi_k \in$ (ILC)PDL we even get an upper-bound of ExpTime or 2ExpTime for the complexity of (ILC)PDL-model-checking.

5.5.1 Context-Bounded MNWs

In the following, we assume that the architecture $\mathfrak{A}$ satisfies $|\mathsf{Procs}| = 1$ and $\mathsf{DS} = \mathsf{Stacks}$. Actually, many underapproximation classes have been defined for this setting of *multiply nested words* (MNWs).

In the first class that we consider, we restrict the number of contexts. In each context, only one stack can be accessed.

Definition 5.11. *Let $\mathcal{M} = (a_1 \ldots a_n, (\rhd^d)_{d \in \mathsf{DS}}) \in \mathsf{CBM}$. Note that $\mathcal{E} = \{1, \ldots, n\}$. A* context *of $\mathcal{M}$ is a possibly empty interval $I = \{e, e+1, \ldots, f\}$, for some $e, f \in \mathcal{E}$, such that, for all $d, d' \in \mathsf{DS}$, $(i,j) \in \rhd^d$, and $(i',j') \in \rhd^{d'}$, the following holds:*

$$\left.\begin{array}{rl} & I \cap \{i,j\} \neq \emptyset \\ \wedge & I \cap \{i',j'\} \neq \emptyset \end{array}\right\} \quad \Rightarrow \quad d = d'$$

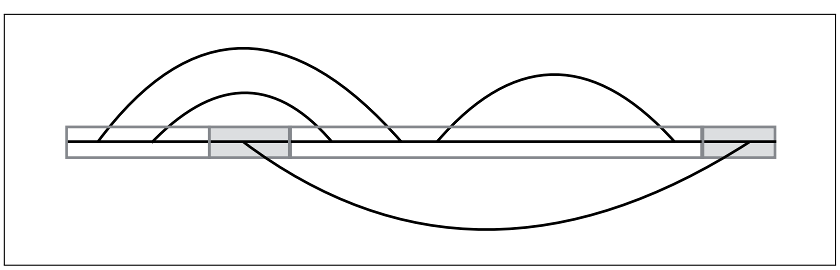
$\Diamond$

Definition 5.12 ([QR05]). *For $k \geq 0$, we call $\mathcal{M}$* k-context-bounded *if there are contexts $I_1, \ldots, I_k$ of $\mathcal{M}$ such that $\mathcal{E} = I_1 \cup \ldots \cup I_k$.* $\Diamond$

Theorem 5.13 ([QR05]). *Non-emptiness (reachability) of multipushdown systems restricted to k-context-bounded is decidable in NP.*

The set of k-context-bounded MNWs (over the fixed architecture) is denoted by $\mathsf{Context}_k$. Moreover, we let $\mathsf{Context} = (\mathsf{Context}_k)_{k \geq 0}$.

Example 5.14. Consider the MNW below, over two stacks and a singleton set Σ (so that we omit its letters).



The curved edges above the horizontal line stand for one of the stacks, the curved edge below it represents the other stack. The MNW is 4-context-bounded, but not 3-context-bounded. $\Diamond$

Lemma 5.15. We have $\mathsf{Context}_1 \subseteq \mathsf{CBM}^{3\text{-}\mathsf{stw}}$ and $\mathsf{Context}_k \subseteq \mathsf{CBM}^{(2k-1)\text{-}\mathsf{stw}}$ for $k \geq 2$, .

Proof. See Section 4.5. ∎

Lemma 5.16. For all $k > 0$, there is $\Phi^k_{\mathsf{context}} \in \mathsf{CPDL}$ of size $\mathcal{O}(k|\mathsf{DS}|^2)$ such that $L(\Phi^k_{\mathsf{context}}) = \mathsf{Context}_k$.

Proof. We introduce some macros. For $d \in \mathsf{DS}$, let $\mathsf{RW}_d = \langle \rhd^d + (\rhd^d)^{-1} \rangle$ be the state formula characterizing events accessing the data structure d. Next, the path formula $\mathsf{onlyRW}_d = (\mathsf{test}(\neg \bigvee_{d' \neq d} \mathsf{RW}_{d'}) \cdot \rightarrow)^* \cdot \mathsf{test}(\neg \bigvee_{d' \neq d} \mathsf{RW}_{d'})$ spans a context (not necessarily maximal) accessing only the data-structure d. Finally, we define

$$\Phi^k_{\mathsf{context}} = \mathsf{E}\Big\langle \mathsf{test}(\neg\langle \rightarrow^{-1} \rangle) \cdot \Big(\sum_{d \in \mathsf{DS}} \mathsf{onlyRW}_d \cdot \rightarrow \Big)^{<k} \cdot \sum_{d \in \mathsf{DS}} \mathsf{onlyRW}_d \cdot \mathsf{test}(\neg\langle \rightarrow \rangle) \Big\rangle$$

Notice the first and last tests ensuring that the path starts on the first event and ends on the last event. ∎

Corollary 5.17. For any architecture $\mathfrak{A}$ such that $|\mathsf{Procs}| = 1$ and $\mathsf{DS} = \mathsf{Stacks}$, the Context-ModelChecking($\mathfrak{A}$) problem is decidable for MSO or ICPDL specifications. The problem is in ExpTime for LCPDL and in 2ExpTime for ICPDL.

Proof. Let $k' = \max(3, 2k-1)$. The Context-ModelChecking($\mathfrak{A}$) problem for a CPDS $\mathcal{S}$ and a specification φ reduces to the emptiness problem for the tree automaton $\mathcal{A}^{k'\text{-}\mathsf{stw}}_{\mathsf{cbm}} \cap \mathcal{A}^{k'\text{-}\mathsf{stw}}_{\Phi^k_{\mathsf{context}}} \cap \mathcal{A}^{k'\text{-}\mathsf{stw}}_{\mathcal{S}} \cap \mathcal{A}^{k'\text{-}\mathsf{stw}}_{\neg\varphi}$. ∎

We can also directly construct a CPDS for the language $\mathsf{Context}_k$:

Lemma 5.18. For all $k > 0$, there is $\mathcal{S}_k \in \mathsf{CPDS}$ such that $L(\mathcal{S}_k) = \mathsf{Context}_k$.

Proof. The idea is simple: The set of locations being $\{\ell_{\text{in}}\} \cup (\mathsf{DS} \times \{1, \ldots, k\})$ and Val a singleton set, one keeps track of the current data structure and context number. The CPDS stays in state ℓ_{in} while reading a prefix of internal events. ∎

Lemma 5.19. For all $k \geq 0$, there is $\varphi_k \in \mathsf{MSO}$ such that $L(\varphi_k) = \mathsf{Context}_k$.

Proof. We define a formula $\mathit{cont}_k(x, y)$ that says that, assuming $x \leq y$, the events x and y are in the scope of at most k contexts. It says that there are no $k+1$ events between x and y that are in distinct contexts:

$$\mathit{cont}_k(x,y) \;=\; \neg\exists x_1, \ldots, x_{k+1} \left(\begin{array}{l} \quad x \leq x_1 < \ldots < x_{k+1} \leq y \\ \wedge \displaystyle\bigwedge_{1 \leq i \leq k} \bigvee_{d \neq d'} \mathit{stack}_d(x_i) \wedge \mathit{stack}_{d'}(x_{i+1}) \end{array} \right)$$

where $\mathit{stack}_d(x_i) = \exists z\,(x_i \rhd^d z \;\vee\; z \rhd^d x_i)$. With this, we set

$$\varphi_k \;=\; \forall x \forall y\; \mathit{cont}_k(x,y)\,.$$

∎

5.5.2 Phase-Bounded MNWs

There is another well established notion for MNWs, which relaxes the notion of a context:

Definition 5.20 ([LMP07]). *Let* $\mathcal{M} = (a_1 \dots a_n, (\rhd^d)_{d \in \mathsf{DS}}) \in \mathsf{CBM}(\mathfrak{A}, \Sigma)$. *A* phase *of* $\mathcal{M}$ *is a set* $I = \{e, e+1, \dots, f\}$, *for some* $e, f \in \mathcal{E}$, *such that, for all* $d, d' \in \mathsf{DS}$, $(i, j) \in \rhd^d$, *and* $(i', j') \in \rhd^{d'}$, *the following holds:*

$$\left.\begin{array}{rl} & I \cap \{j\} \neq \emptyset \\ \wedge & I \cap \{j'\} \neq \emptyset \end{array}\right\} \quad \Rightarrow \quad d = d' \qquad \diamondsuit$$

Theorem 5.21 ([LMP07]). *Non-emptiness (reachability) of multipushdown systems restricted to k-phase-bounded is decidable in* 2ExpTime.

The special tree-width of *k-phase-bounded* MNWs (i.e., those MNWs that can be split into at most k phases) is at most $k' = 2^{2k}$ [CGNK12].

Lemma 5.22. For all $k > 0$, there is $\Phi^k_{\mathsf{phase}} \in \mathsf{CPDL}$ of size $\mathcal{O}(k|\mathsf{DS}|^2)$ such that $L(\Phi^k_{\mathsf{phase}}) = \mathsf{Phase}_k$.

Proof. The formula is obtained from $\Phi^k_{\mathsf{context}}$ by replacing onlyRW_d with

$$\mathsf{onlyR}_d = (\mathsf{test}(\neg \textstyle\bigvee_{d' \neq d} \langle (\rhd^{d'})^{-1} \rangle) \cdot \rightarrow)^* \cdot \mathsf{test}(\neg \textstyle\bigvee_{d' \neq d} \langle (\rhd^{d'})^{-1} \rangle)$$

which spans a phase reading only the data-structure d. We define

$$\Phi^k_{\mathsf{phase}} = \mathsf{E}\Big\langle \mathsf{test}(\neg\langle \rightarrow^{-1} \rangle) \cdot \Big(\sum_{d \in \mathsf{DS}} \mathsf{onlyR}_d \cdot \rightarrow \Big)^{<k} \cdot \sum_{d \in \mathsf{DS}} \mathsf{onlyR}_d \cdot \mathsf{test}(\neg\langle \rightarrow \rangle) \Big\rangle$$

Notice the first and last tests ensuring that the path starts on the first event and ends on the last event. ∎

Corollary 5.23. For any architecture $\mathfrak{A}$ such that $|\mathsf{Procs}| = 1$ and $\mathsf{DS} = \mathsf{Stacks}$, the Phase-ModelChecking($\mathfrak{A}$) problem is decidable for MSO or ICPDL specifications. The problem is in 2ExpTime for ICPDL.

5.5.3 Scope-Bounded Nested Words

Next, we define a restriction that captures more behaviors than pure contexts. We continue to assume $|\mathsf{Procs}| = 1$ and $\mathsf{DS} = \mathsf{Stacks}$.

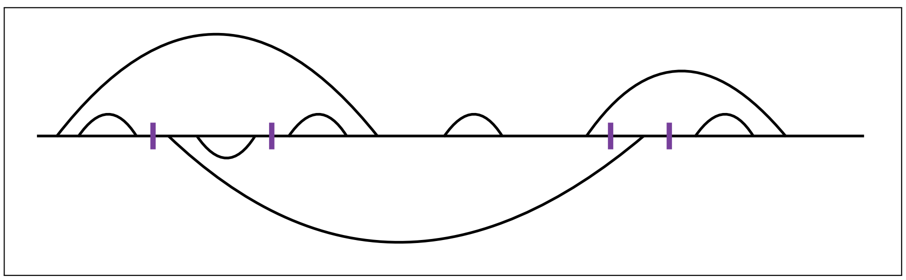
Definition 5.24 ([LN11]). *For $k \geq 0$, we call an MNW $\mathcal{M}$ k-*scope-bounded *if, for all $(e, f) \in \rhd$, there are contexts $I_1, \ldots, I_k$ of $\mathcal{M}$ such that*

$$\{e, e+1, \ldots, f\} = I_1 \cup \ldots \cup I_k \,. \qquad \diamondsuit$$

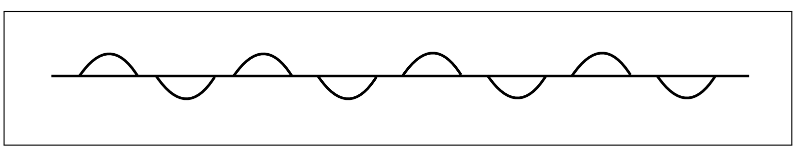
Theorem 5.25 ([LN11]). *Non-emptiness (reachability) of multipushdown systems restricted to k-scope-bounded is decidable in* PSPACE.

The set of k-scope-bounded MNWs is denoted by Scope_k. Moreover, we let $\mathsf{Scope} = (\mathsf{Scope}_k)_{k \geq 0}$.

Example 5.26. The figure below illustrates a CBM $\mathcal{M}$ with $\mathcal{M} \in \mathsf{Scope}_3$. Note that $\mathcal{M} \in \mathsf{Context}_5 \setminus \mathsf{Context}_4$.



Next, consider the set L of CBMs with an arbitrary number of alternating write-read edges, following the pattern below:



Then, $L \subseteq \mathsf{Scope}_1$ but $L \not\subseteq \mathsf{Context}_k$ for all $k \geq 0$. $\diamondsuit$

Lemma 5.27. We have $\mathsf{Scope}_1 \subseteq \mathsf{CBM}^{\text{3-stw}}$ and $\mathsf{Scope}_k \subseteq \mathsf{CBM}^{(2k-1)\text{-stw}}$ for $k \geq 2$.

Proof. If $k = 1$, then a CBM is the concatenation of (singly) nested words. So, suppose $k \geq 2$. Again, we show that Eve has a winning strategy in the split-game, using at most $2k$ colors. To illustrate her strategy, we consider Figure 5.1, depicting the CBM from Example 5.26. We omit internal events, which are easy to handle.

- Consider the *leftmost* write (i.e., a push). We split the process edge just behind the corresponding read (i.e., a pop). Moreover, we divide the induced prefix into its contexts. Since the CBM is k-scope-bounded, this process requires at most k split-edges (1), i.e., at most $2k$ colors.

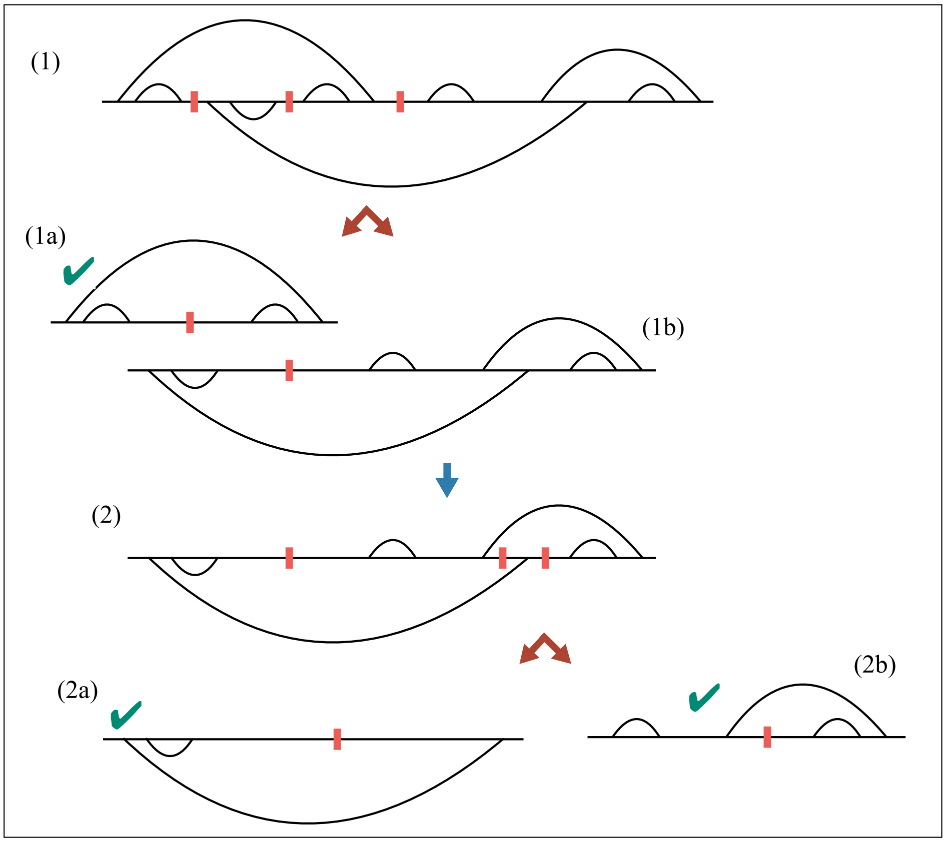


Figure 5.1: Split-game for scope-bounded MNWs

- One resulting component is a nested word (1a) with at most $2k-3$ colors and where the last point is colored. The nested word can be decomposed using three more colors, i.e., using a total of at most $2k$ colors.
- We proceed with the other remaining component (1b), which possibly has already some split-edges. Again, we look at the first call and its receive f, place a split-edge behind f, and divide the corresponding prefix into its contexts. Though the prefix may already contain split-edges, we do not have more than k split-edges after the division phase (2). We proceed like in the 2nd item.

Altogether, Eve wins the split-game with at most $2k$ colors. ∎

Lemma 5.28. For all $k > 0$, there is $\Phi^k_{\mathsf{scope}} \in \mathsf{LCPDL}$ of size $\mathcal{O}(k|\mathsf{DS}|^2)$ such that $L(\Phi^k_{\mathsf{scope}}) = \mathsf{Scope}_k$.

Proof. Recall that the state formula $\mathsf{RW}_d = \langle \rhd^d + (\rhd^d)^{-1} \rangle$ characterizes events accessing the data structure $d \in \mathsf{DS}$. We define

$$\Phi^k_{\mathsf{scope}} = \neg \mathsf{ELoop}\Big\langle \Big(\sum_{d \neq d'} \mathsf{test}(\mathsf{RW}_d) \cdot \rightarrow^+ \cdot\, \mathsf{test}(\mathsf{RW}_{d'}) \Big)^k \cdot \rhd^{-1} \Big\rangle$$

Notice that the path formula $\mathsf{test}(\mathsf{RW}_d) \cdot \rightarrow^+ \cdot \mathsf{test}(\mathsf{RW}_{d'})$ with $d \neq d'$ ensures a change of context. ■

Corollary 5.29. For any architecture $\mathfrak{A}$ such that $|\mathsf{Procs}| = 1$ and $\mathsf{DS} = \mathsf{Stacks}$, the Scope-ModelChecking($\mathfrak{A}$) problem is decidable for MSO or ICPDL specifications. The problem is in ExpTime for LCPDL and in 2ExpTime for ICPDL.

Lemma 5.30. For all $k \geq 0$, there is $\varphi_k \in \mathsf{MSO}$ such that $L(\varphi_k) = \mathsf{Scope}_k$.

Proof. According to the definition of k-scope-bounded words, it is enough to set

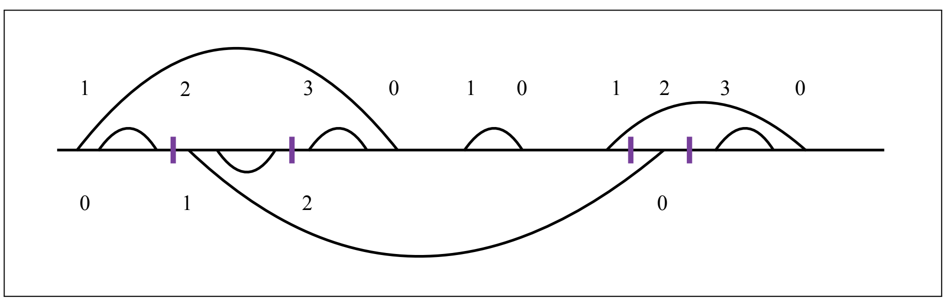
$$\varphi_k \; = \; \forall x \forall y \, (x \rhd y \;\Rightarrow\; \mathit{cont}_k(x,y))\,.$$

■

Again, one can also directly construct a CPDS:

Lemma 5.31. For all $k \geq 0$, there is $\mathcal{S}_k \in \mathsf{CPDS}$ such that $L(\mathcal{S}_k) = \mathsf{Scope}_k$.

Proof. The idea is to employ a counter from 1 to k for *each* stack $d \in \mathsf{DS}$, and to count the number of contexts in the scope of every *outermost* write-read edge of d. Thus, we can do with set of locations $\{\ell_{\mathrm{in}}, \ell_{\mathrm{pref}}\} \cup (\mathsf{DS} \times \{0, 1, \ldots, k\}^{\mathsf{DS}})$ and $\mathsf{Val} = \{\mathrm{o}, \mathrm{i}\}$, where a pushed value o signals an *outermost* nesting edge, and i an inner nesting edge. The following figure shows how the two counters, one for each stack, work:



Note that the number of locations is exponential in $|\mathsf{DS}|$. ■

5.5.4 Existentially Bounded MSCs

7th Lecture

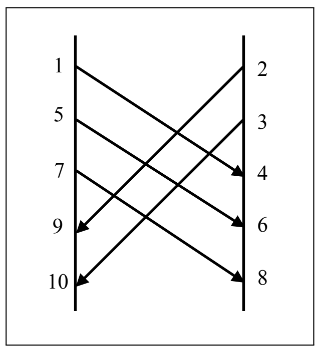
We consider existentially bounded CBMs. The following definition is slightly different from the notion that we have already seen. In fact, we now define a *local* variant of existential bounds.

We assume that there are no local events and that Σ is a singleton set (and can therefore be omitted).

Definition 5.32. *A* CBM $\mathcal{M}$ *is* $\exists k$-bounded *if it admits some linearization* $<_{\mathsf{lin}}$ *such that, at any time, there are no more than* k *messages in* each *data structure: for all* $g \in \mathcal{E}$ *and all* $d \in \mathsf{DS}$*,* $|\{(e,f) \in \rhd^d \mid e \leq_{\mathsf{lin}} g <_{\mathsf{lin}} f\}| \leq k$*.* $\diamondsuit$

The set of $\exists k$-bounded CBMs (over the given architecture) is denoted by $\mathsf{CBM}_{\exists k}$. Moreover, we let $\mathsf{CBM}_{\exists} = (\mathsf{CBM}_{\exists k})_{k \geq 0}$.

Example 5.33. Consider the MSC $\mathcal{M}$ below:



The suggested linearization shows that $\mathcal{M}$ is $\exists 2$-bounded. In fact, $\mathcal{M} \in \mathsf{CBM}_{\exists 2} \setminus \mathsf{CBM}_{\exists 1}$. $\diamondsuit$

We have already seen (Exercise 4.22 that $\exists k$-bounded CBMs have bounded special tree-width, though the bound on special tree-width has to take into account that we defined a local variant of existential bounds:

Lemma 5.34. For all $k \geq 0$, we have $\mathsf{CBM}_{\exists k} \subseteq \mathsf{CBM}^{(k|\mathsf{DS}|+|\mathsf{Procs}|)\text{-}\mathsf{stw}}$.

Proof. Let us quickly recall the proof by means of Figure 5.2.

- Eve's strategy is to choose a linearization and to cut the first $k|\mathsf{DS}|+1 = 5$ events according to that linearization (1). For this, she uses $k|\mathsf{DS}|+|\mathsf{Procs}|+1$ colors and she removes the $\to$-edges touching the first $k|\mathsf{DS}|+1$ events.
- Then, there is at least one isolated message edge, which in this case is $(1,4)$.
- In the remaining component, we again cut some more events until we isolated $k|\mathsf{DS}|+1$ of them (2), and so on.

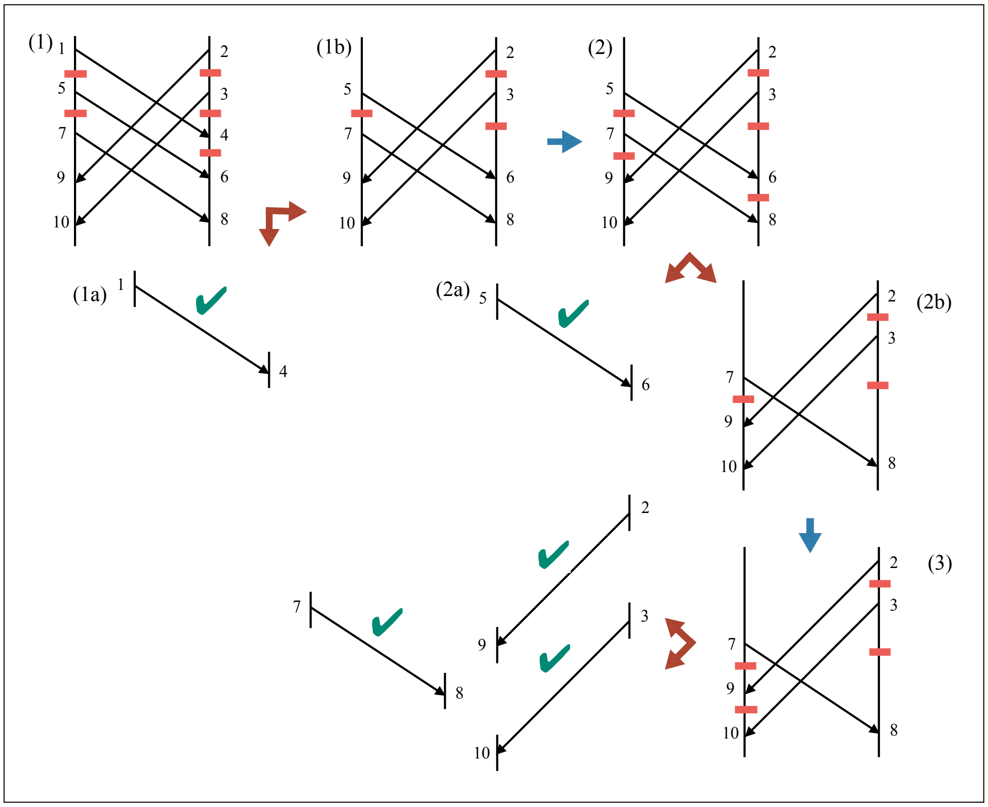


Figure 5.2: Split-game for existentially bounded MSCs

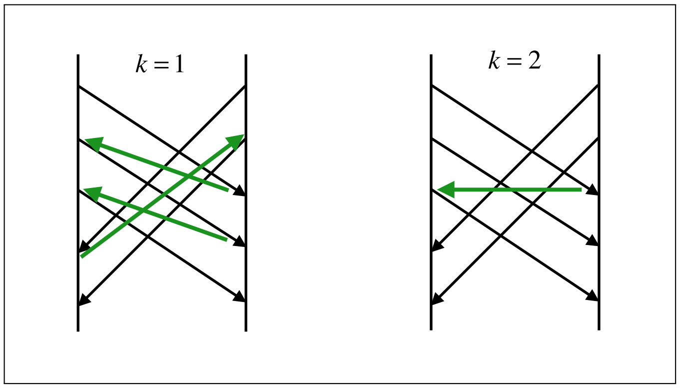
Thus, Eve wins the decomposition game with at most $k|\mathsf{DS}| + |\mathsf{Procs}| + 1$ colors.■

Lemma 5.35. For all $k > 0$, there is $\Phi^k_{\exists\mathsf{B}} \in \mathsf{LCPDL}$ of size $\mathcal{O}(k|\mathsf{DS}|)$ such that $L(\Phi^k_{\exists\mathsf{B}}) \cap \mathsf{CBM} = \mathsf{CBM}_{\exists k}$.

Proof. For an CBM $\mathcal{M}$, consider the binary relation $\rightsquigarrow_k \subseteq \mathcal{E} \times \mathcal{E}$ that connects events (f, e) where, for some data structure $d \in \mathsf{DS}$ and some $i \geq 1$:

- e is the $(i + k)$-th write on d
- f is the i-th read from d

The relation $\rightsquigarrow_k$ is illustrated below for the cases $k = 1$ (cyclic $\Rightarrow$ not $\exists$1-bounded) and $k = 2$ (acyclic $\Rightarrow$ $\exists$2-bounded).



Claim 5.36 ([LM04] for the special case $\mathsf{DS} = \mathsf{Queues}$).

$$\mathcal{M} \in \mathsf{CBM}_{\exists k} \text{ iff } < \cup \rightsquigarrow_k \text{ is acyclic.}$$

$\Leftarrow$ Take $<_{\mathsf{lin}}$ any linearization of $< \cup \rightsquigarrow_k$. We check that $<_{\mathsf{lin}}$ is k-bounded.

$\Rightarrow$ Let $<_{\mathsf{lin}}$ be a k-bounded linearization of $<$. We check that $f \rightsquigarrow e$ implies $f <_{\mathsf{lin}} e$.

The binary relation $\rightsquigarrow_k$ can be formalized by a path formula from CPDL:

$$\rightsquigarrow_k = \sum_{d \in \mathsf{DS}} (\rhd^d)^{-1} \cdot \Big(\rightarrow \cdot \, (\mathsf{test}(\neg\langle \rhd^d \rangle) \cdot \rightarrow)^* \cdot \mathsf{test}(\langle \rhd^d \rangle) \Big)^k$$

Acyclicity relies on the loop predicate: $\Phi^k_{\exists \mathsf{B}} = \neg \mathsf{ELoop}\langle (\rightarrow + \rhd + \rightsquigarrow_k)^+ \rangle$. ∎

Corollary 5.37. For any architecture $\mathfrak{A}$, the $\mathsf{CBM}_\exists$-ModelChecking($\mathfrak{A}$) problem is decidable for MSO or ICPDL specifications. The problem is in ExpTime for LCPDL and in 2ExpTime for ICPDL.

Lemma 5.38. For all $k \geq 0$, there is $\varphi_k \in \mathsf{MSO}$ such that $L(\varphi_k) = \mathsf{CBM}_{\exists k}$.

For message passing systems, there is a CPDS defining $\mathsf{CBM}_{\exists k}$, too, but the proof is much harder (and omitted here):

Theorem 5.39 ([GKM06]). *For any architecture $\mathfrak{A}$ such that $\mathsf{DS} = \mathsf{Queues}$, and for all $k \geq 0$, there is $\mathcal{S}_k \in \mathsf{CPDS}$ such that $L(\mathcal{S}_k) = \mathsf{CBM}_{\exists k}$.*

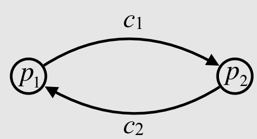
5.6 Synthesis from ICPDL to CPDS

The synthesis problem is to construct an implementation $\mathcal{S}$ from a given specification Φ. Here we are interested in specifications given in ICPDL and distributed implementations, i.e., CPDSs.

In the following, given $\Phi \in \mathsf{ICPDL}(\mathfrak{A}, \Sigma)$, we let $L_{\mathsf{cbm}}(\Phi) := L(\Phi) \cap \mathsf{CBM}(\mathfrak{A}, \Sigma)$.

Unfortunately, IPDL and LCPDL are too expressive to be translatable into CPDSs:

Theorem 5.40. *Suppose $\Sigma = \{a, b, c\}$ and let $\mathfrak{A}$ be given as follows:*



There is $\Phi \in \mathsf{IPDL}(\mathfrak{A}, \Sigma)$ such that $L(\mathcal{S}) \neq L_{\mathsf{cbm}}(\Phi)$, for all $\mathcal{S} \in \mathsf{CPDS}(\mathfrak{A}, \Sigma)$.

There is $\Phi \in \mathsf{LCPDL}(\mathfrak{A}, \Sigma)$ such that $L(\mathcal{S}) \neq L_{\mathsf{cbm}}(\Phi)$, for all $\mathcal{S} \in \mathsf{CPDS}(\mathfrak{A}, \Sigma)$.

Exercise 5.41. Prove Theorem 5.40 using the idea in the proof of Theorem 3.5. $\diamondsuit$

The exact relation between CPDL and CPDS is unknown. However, every PDL formula can be translated into a CPDS (the special case $\mathsf{DS} = \mathsf{Queues}$ was considered in [BKM10]):

Theorem 5.42 ([AGNK14]). *For every $\Phi \in \mathsf{PDL}(\mathfrak{A}, \Sigma)$, there is $\mathcal{S} \in \mathsf{CPDS}(\mathfrak{A}, \Sigma)$ such that $L(\mathcal{S}) = L_{\mathsf{cbm}}(\Phi)$.*

Proof. For a state formula σ, we construct, inductively,

$$\mathcal{S}_\sigma = (\mathsf{Locs}, \mathsf{Val}, (\rightarrow_p)_{p \in \mathsf{Procs}}, \ell_{\mathrm{in}}, \mathsf{Fin}) \in \mathsf{CPDS}(\mathfrak{A}, \Sigma)$$

together with a mapping $\gamma_\sigma \colon \mathsf{Trans} \to \{0, 1\}$ such that

- $L(\mathcal{S}_\sigma) = \mathsf{CBM}(\mathfrak{A}, \Sigma)$ and,
- for all $\mathcal{M} \in \mathsf{CBM}(\mathfrak{A}, \Sigma)$, all accepting runs ρ of $\mathcal{S}_\sigma$ on $\mathcal{M}$, and all events e of $\mathcal{M}$, we have
$$e \in [\![\sigma]\!]_{\mathcal{M}} \;\text{ iff }\; \gamma_\sigma(\rho(e)) = 1\,.$$

Here $\mathsf{Trans} = \biguplus_{p \in \mathsf{Procs}} \rightarrow_p$ and a run is a map $\rho \colon \mathcal{E} \to \mathsf{Trans}$.

The pair $(\mathcal{S}_\sigma, \gamma_\sigma)$ is a *transducer* from the input alphabet Σ to the output alphabet $\{0, 1\}$.

CPDS $\mathcal{S}_a$ for $a \in \Sigma$:

We let $\mathcal{S}_a = (\mathsf{Locs}, \mathsf{Val}, (\rightarrow_p)_{p \in \mathsf{Procs}}, \ell_{\text{in}}, \mathsf{Fin})$ where

- $\mathsf{Locs} = \{1\} \quad \mathsf{Val} = \{v\} \quad \ell_{\text{in}} = 1 \quad \mathsf{Fin} = \{1\}^{\mathsf{Procs}}$,
- transitions and output function ($b \neq a$, $p \in \mathsf{Procs}$ and $d \in \mathsf{DS}$):
  - $\gamma_a(1 \xrightarrow{a}_p 1) = \gamma_a(1 \xrightarrow{a,d!v}_{\mathsf{Writer}(d)} 1) = \gamma_a(1 \xrightarrow{a,d?v}_{\mathsf{Reader}(d)} 1) = 1;$
  - $\gamma_a(1 \xrightarrow{b}_p 1) = \gamma_a(1 \xrightarrow{b,d!v}_{\mathsf{Writer}(d)} 1) = \gamma_a(1 \xrightarrow{b,d?v}_{\mathsf{Reader}(d)} 1) = 0.$

CPDS $\mathcal{S}_p$ for $p \in \mathsf{Procs}$:

As above, $\mathcal{S}_p = (\mathsf{Locs}, \mathsf{Val}, (\rightarrow_q)_{q \in \mathsf{Procs}}, \ell_{\text{in}}, \mathsf{Fin})$ is the universal CPDS with one state. The output function is modified:

$$\gamma_p(t) = \begin{cases} 1 & \text{if } t \in \rightarrow_p \\ 0 & \text{otherwise.} \end{cases}$$

CPDS $\mathcal{S}_{\neg\sigma}$:

Suppose $\mathcal{S}_\sigma = (\mathsf{Locs}, \mathsf{Val}, (\rightarrow_p)_{p \in \mathsf{Procs}}, \ell_{\text{in}}, \mathsf{Fin})$ with associated mapping $\gamma_\sigma \colon \mathsf{Trans} \to \{0,1\}$. We set $\mathcal{S}_{\neg\sigma} = \mathcal{S}_\sigma$ and let $\gamma_{\neg\sigma}(t) = 1 - \gamma_\sigma(t)$ for all $t \in \mathsf{Trans}$.

CPDS $\mathcal{S}_{\sigma_1 \vee \sigma_2}$:

Suppose $\mathcal{S}_{\sigma_i} = (\mathsf{Locs}_i, \mathsf{Val}_i, (\rightarrow^i_p)_{p \in \mathsf{Procs}}, \ell^i_{\text{in}}, \mathsf{Fin}_i)$, for $i \in \{1,2\}$.

We construct the product $\mathcal{S}_{\sigma_1} \times \mathcal{S}_{\sigma_2} = (\mathsf{Locs}, \mathsf{Val}, (\rightarrow_p)_{p \in \mathsf{Procs}}, \ell_{\text{in}}, \mathsf{Fin})$ as usual:

- $\mathsf{Locs} = \mathsf{Locs}_1 \times \mathsf{Locs}_2$
- $\mathsf{Val} = \mathsf{Val}_1 \times \mathsf{Val}_2$
- $\ell_{\text{in}} = (\ell^1_{\text{in}}, \ell^2_{\text{in}})$
- $\mathsf{Fin} = \{((\ell^p_1, \ell^p_2))_{p \in \mathsf{Procs}} \in \mathsf{Locs}^{\mathsf{Procs}} \mid (\ell^p_i)_{p \in \mathsf{Procs}} \in \mathsf{Fin}_i \text{ for all } i \in \{1,2\}\}$
- transitions:
  - $t_1 \times t_2 = (\ell_1, \ell_2) \xrightarrow{a}_p (\ell'_1, \ell'_2) \quad \text{if} \quad t_i = \ell_i \xrightarrow{a}{}^i_p \ell'_i$ for all $i \in \{1,2\}$
  - $t_1 \times t_2 = (\ell_1, \ell_2) \xrightarrow{a,d!(v_1,v_2)}_p (\ell'_1, \ell'_2) \quad \text{if} \quad t_i = \ell_i \xrightarrow{a,d!v_i}{}^i_p \ell'_i$ for all $i \in \{1,2\}$
  - $t_1 \times t_2 = (\ell_1, \ell_2) \xrightarrow{a,d?(v_1,v_2)}_p (\ell'_1, \ell'_2) \quad \text{if} \quad t_i = \ell_i \xrightarrow{a,d?v_i}{}^i_p \ell'_i$ for $i \in \{1,2\}$

Finally, we let $\gamma_{\sigma_1 \vee \sigma_2}(t_1 \times t_2) = \max\{\gamma_{\sigma_1}(t_1), \gamma_{\sigma_2}(t_2)\}$.

More generally, transducers are closed under product: given $(\mathcal{S}_1, \gamma_1)$ and $(\mathcal{S}_2, \gamma_2)$, we construct $\mathcal{S} = \mathcal{S}_1 \times \mathcal{S}_2$ as above and we let $\gamma(t_1 \times t_2) = (\gamma_1(t_1), \gamma_2(t_2))$.

Transducers are also closed under composition.

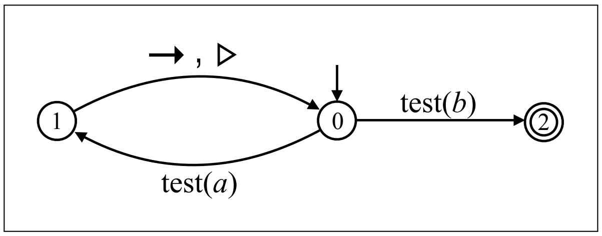
Let us turn to the case of formulas $\langle\pi\rangle\sigma$.

$\langle\pi\rangle\sigma \equiv \langle\pi\cdot\mathsf{test}(\sigma)\rangle\mathsf{true}$. Hence, we may assume that if $\langle\pi\rangle\sigma$ appears as a subformula then σ is true. Furthermore, we simply denote it by $\langle\pi\rangle$ (which means $\langle\pi\rangle\mathsf{true}$).

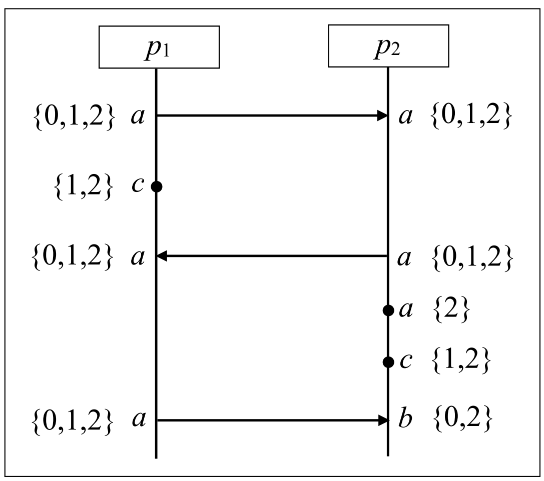
Example 5.43. Let us illustrate the idea by means of an example. Consider the PDL path formula

$$\pi \;=\; \big(\mathsf{test}(a)\cdot(\rightarrow + \rhd)\big)^* \cdot \mathsf{test}(b)\,.$$

We translate π into a finite automaton $\mathcal{B}$ over the alphabet $\{\rhd, \rightarrow, \mathsf{test}(a), \mathsf{test}(b)\}$ as follows:



The CPDS $\mathcal{S}_{\langle\pi\rangle}$ will now label each event e of a CBM with the set of states from which one "can reach a final state of $\mathcal{B}$", starting from event e. This computation proceeds backward starting from the maximal events wrt. $<$ (in the example, the only b-labeled one):



This can indeed be achieved by a CPDS. To do so, the CPDS has to "inspect", at each event e, the states at the immediate $\rightarrow$- and $\rhd^d$-successors of e. In particular, at a write event, it will have to guess what will be the state at the corresponding read event. Finally, an event satisfies $\langle\pi\rangle$ iff the initial state 0 is contained in the labeling. $\diamondsuit$

CPDS $\mathcal{S}_{\langle\pi\rangle}$:

Let $\text{Tests}(\pi) = \{\mathsf{test}(\sigma_1), \ldots, \mathsf{test}(\sigma_n)\}$ be the set of tests appearing in π.

Now, π can be seen as a regular expression over the alphabet

$$\Omega = \text{Tests}(\pi) \cup \{\rhd^d \mid d \in \mathsf{DS}\} \cup \{\rightarrow\}\,.$$

Let $\mathcal{B} = (S, \delta, \iota, F)$ be a finite automaton over Ω for π, i.e., such that $L(\mathcal{B}) = L(\pi) \subseteq \Omega^*$. Note that we can assume $|S| = |\pi|$. Given $s \in S$, we set $\mathcal{B}_s = (S, \delta, s, F)$, i.e., $\mathcal{B}_s$ is essentially $\mathcal{B}$, but with new initial state s.

Let π_s be a rational expression over Ω that is equivalent to $\mathcal{B}_s$ (in particular, $\pi_\iota = \pi$).

For a CBM $\mathcal{M} = ((w_p)_{p\in\mathsf{Procs}}, (\rhd^d)_{d\in\mathsf{DS}})$, we want to compute with a transducer the map $\nu : \mathcal{E} \to 2^S$ such that

$$\nu(e) = \{s \in S \mid e \in [\![\langle\pi_s\rangle]\!]_{\mathcal{M}}\}\,.$$

Let $\nu^+(e) = \begin{cases} \nu(f) & \text{if } e \rightarrow f \\ \emptyset & \text{if } e \text{ is maximal on its process.} \end{cases}$

Let $H(e) = \{\mathsf{test}(\sigma_i) \mid e \in [\![\sigma_i]\!]_{\mathcal{M}}\} \subseteq \Omega$.

For $K \subseteq \Omega^*$ and $T \subseteq S$, let

$$\delta^{-1}(K, T) = \{s \in S \mid \delta(s, w) \in T \text{ for some } w \in K\}\,.$$

Lemma 5.44. (i) If e is not a write event, then

$$\nu(e) = \delta^{-1}(H(e)^*, F \cup \delta^{-1}(\rightarrow, \nu^+(e)))\,.$$

(ii) If $e \rhd^d f$, then

$$\nu(e) = \delta^{-1}(H(e)^*, F \cup \delta^{-1}(\rightarrow, \nu^+(e)) \cup \delta^{-1}(\rhd^d, \nu(f)))\,.$$

Exercise 5.45. Prove Lemma 5.44. ◇

Remark 5.46. If e is maximal wrt. $<$, then $\nu(e) = \delta^{-1}(H(e)^*, F)$ can be computed directly.

We are looking for a transducer $\mathcal{S}_\nu$, i.e., a CPDS with output, which computes ν.

Problem: The computation of ν goes *backward*, whereas a CPDS run goes *forward*.

Solution: Guess ν nondeterministically and check afterwards whether the guess was correct.

We define $\mathcal{S}_\nu = \underbrace{(\mathcal{S}_{\sigma_1} \times \cdots \times \mathcal{S}_{\sigma_n})}_{\substack{\text{computes } H \\ \text{as input for } \mathcal{B}}} \cdot \mathcal{B}$. By induction, the $(\mathcal{S}_{\sigma_i})_{1\leq i\leq n}$ are given.

The product transducer $\mathcal{S} = \mathcal{S}_{\sigma_1} \times \cdots \times \mathcal{S}_{\sigma_n}$ outputs letters in $\mathbb{B}^n = \{0,1\}^n$.

Let $\rho = \rho_1 \times \cdots \times \rho_n$ be an accepting run of $\mathcal{S}$ on a CBM $\mathcal{M}$.

For all events $e \in \mathcal{E}$, we have $\gamma(\rho(e)) = (\gamma_1(\rho_1(e)), \ldots, \gamma_n(\rho_n(e)))$.

For $G = (G_1, \ldots, G_n) \in \mathbb{B}^n$, define $\overline{G} = \{\mathsf{test}(\sigma_i) \mid G_i = 1\}$.

By induction hypothesis, we have $\gamma_i(\rho_i(e)) = 1$ iff $\mathcal{M}, e \models \sigma_i$.

Therefore, $\overline{\gamma(\rho(e))} = H(e)$.

The transducer $\mathcal{B}$ has input alphabet $\mathbb{B}^n$ and output alphabet 2^S. Since we use future modalities, the transducer $\mathcal{B}$ must be non-determinitic. But we construct a transducer which is backward deterministic and backward complete. Since it is backward deterministic, it has a unique global final state, but it uses a set of global initial states. This is a generalization compared with single local initial states, but can be simulated with nondeterminism.

We define $\mathcal{B} = (\mathsf{Locs}, \mathsf{Val}, (\rightarrow_p)_{p \in \mathsf{Procs}}, \mathsf{Init}, \mathsf{Fin})$ as follows:

- $\mathsf{Locs} = 2^S$. A set $U \in \mathsf{Locs}$ represents $\nu(e)$.
- $\mathsf{Val} = 2^S$. Here, $W \in \mathsf{Val}$ represents $\nu(f)$ when $e \triangleright^d f$.
- $\mathsf{Init} = (2^S)^{\mathsf{Procs}}$.
- $\mathsf{Fin} = \{\emptyset\}^{\mathsf{Procs}} = \{(\emptyset, \ldots, \emptyset)\}$.

Now, we turn to the transitions.

$$
\begin{array}{lll}
U \xrightarrow{G}_p V & \text{if } U = \delta^{-1}(G^*, F \cup \delta^{-1}(\rightarrow, V)) & \\
U \xrightarrow{G,d!W}_p V & \text{if } U = \delta^{-1}(G^*, F \cup \delta^{-1}(\rightarrow, V) \cup \delta^{-1}(\triangleright^d, W)) & (p = \mathsf{Writer}(d)) \\
U \xrightarrow{G,d?U}_p V & \text{if } U = \delta^{-1}(G^*, F \cup \delta^{-1}(\rightarrow, V)) & (p = \mathsf{Reader}(d))
\end{array}
$$

Finally, we set $\gamma_{\mathcal{B}}(t) = \mathsf{src}(t)$.

Define $\mathcal{S}_\nu$ as the composition $(\mathcal{S}_1 \times \cdots \times \mathcal{S}_n) \cdot \mathcal{B}$.

Lemma 5.47. Let $\mathcal{M}$ be a CBM and let $\rho = \rho_1 \times \cdots \times \rho_n \times \rho_{\mathcal{B}}$ be an accepting run of $\mathcal{S}_\nu$ on $\mathcal{M}$. Then, for all events $e \in \mathcal{E}$, we have

$$\gamma_\nu(\rho(e)) = \gamma_{\mathcal{B}}(\rho_{\mathcal{B}}(e)) = \nu(e)\,.$$

Proof. The proof is by induction on the partial order $<$ of $\mathcal{M}$, starting from the maximal events. It is based on Lemma 5.44. ∎

Finally, we obtain $\mathcal{S}_{\langle\pi\rangle}$ as the composition $\mathcal{S}_\nu \cdot \mathcal{C}$ where $\mathcal{C}$ is the universal transducer with one state which transforms an input letter $T \subseteq S$ into the boolean value

$$\begin{cases} 1 & \text{if } \iota \in T \\ 0 & \text{otherwise.} \end{cases}$$

It remains to define automata for PDL formulas $\Phi \in \mathsf{PDL}(\mathfrak{A}, \Sigma)$. Without loss of generality, we assume that Φ is a positive boolean combination of formulas of the form $\mathsf{E}\sigma$ or $\mathsf{A}\sigma$. Disjunction and conjunction are easy to handle, since CPDSs are closed under union and intersection (exercise).

For the case $\mathsf{A}\sigma$, suppose we already have $\mathcal{S}_\sigma = (\mathsf{Locs}, \mathsf{Val}, (\rightarrow_p)_{p \in \mathsf{Procs}}, \ell_{\mathrm{in}}, \mathsf{Fin})$ with associated mapping $\gamma_\sigma \colon \mathsf{Trans} \to \{0,1\}$. Then, the CPDS for $\mathsf{A}\sigma$ is simply the restriction of $\mathcal{S}$ to those transitions t such that $\gamma_\sigma(t) = 1$.

The case $\mathsf{E}\sigma$ is left as an exercise. ∎

Exercise 5.48. Consider the following extension of $\mathsf{PDL}(\mathfrak{A}, \Sigma)$, which we call $\mathsf{PDL}^{-1}(\mathfrak{A}, \Sigma)$:

$$\begin{aligned}
\Phi &::= \mathsf{E}\sigma \mid \Phi \vee \Phi \mid \neg\Phi \\
\sigma &::= p \mid a \mid \sigma \vee \sigma \mid \neg\sigma \mid \langle\pi\rangle\sigma \mid \langle\pi^{-1}\rangle\sigma \\
\pi &::= \rhd^d \mid \rightarrow \mid \mathsf{test}(\sigma) \mid \pi + \pi \mid \pi \cdot \pi \mid \pi^*
\end{aligned}$$

where $p \in \mathsf{Procs}$, $d \in \mathsf{DS}$ and $a \in \Sigma$. Show that Theorem 5.42 even holds for the logic $\mathsf{PDL}^{-1}(\mathfrak{A}, \Sigma)$:
For every $\Phi \in \mathsf{PDL}^{-1}(\mathfrak{A}, \Sigma)$, there is $\mathcal{S} \in \mathsf{CPDS}(\mathfrak{A}, \Sigma)$ such that $L(\mathcal{S}) = L_{\mathsf{cbm}}(\Phi)$.$\diamondsuit$

Remark 5.49. Since PDL is closed under complementation (negation), while CPDSs are not (for certain architectures), we obtain, as a corollary, that CPDSs are strictly more expressive than PDL.

Bibliography

[AG14] C. Aiswarya and P. Gastin. Reasoning about distributed systems: WYSIWYG (invited talk). In *Proceedings of the 34th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'14)*, Leibniz International Proceedings in Informatics. Leibniz-Zentrum für Informatik, 2014. To appear.

[AGNK14] C. Aiswarya, P. Gastin, and K. Narayan Kumar. Verifying Communicating Multi-pushdown Systems. Technical report, Inria, 2014.

[AM09] R. Alur and P. Madhusudan. Adding nesting structure to words. *J. ACM*, 56(3):1–43, 2009.

[BKM10] B. Bollig, D. Kuske, and I. Meinecke. Propositional dynamic logic for message-passing systems. *Logical Methods in Computer Science*, 6(3:16), September 2010.

[BL06] B. Bollig and M. Leucker. Message-passing automata are expressively equivalent to EMSO logic. *Theor. Comput. Sci.*, 358(2-3):150–172, 2006.

[Büc60] J. Büchi. Weak second-order arithmetic and finite automata. *Z. Math. Logik, Grundlag. Math.*, 5:66–62, 1960.

[CE12] B. Courcelle and J. Engelfriet. *Graph Structure and Monadic Second-Order Logic - A Language-Theoretic Approach*, volume 138 of *Encyclopedia of mathematics and its applications*. Cambridge University Press, 2012.

[CGNK12] A. Cyriac, P. Gastin, and K. Narayan Kumar. MSO decidability of multi-pushdown systems via split-width. In *Proceedings of CONCUR 2012*, volume 7454 of *Lecture Notes in Computer Science*, pages 547–561. Springer, 2012.

[CGNK14] A. Cyriac, P. Gastin, and K. Narayan Kumar. Verifying communicating multi-pushdown systems via split-width. In *Proceedings of the 12th International Symposium on Automated Technology for Verification and Analysis (ATVA'14)*, volume 8837 of *Lecture Notes in Computer Science*. Springer, 2014. To appear.

[Cou10] Bruno Courcelle. Special tree-width and the verification of monadic second-order graph pr operties. In *FSTTCS*, volume 8 of *LIPIcs*, pages 13–29, 2010.

[Elg61] C. C. Elgot. Decision problems of finite automata design and related arithmetics. *Trans. Amer. Math. Soc.*, 98:21–52, 1961.

[GKM06] B. Genest, D. Kuske, and A. Muscholl. A Kleene theorem and model checking algorithms for existentially bounded communicating automata. *Information and Computation*, 204(6):920–956, 2006.

[GLL09] S. Göller, M. Lohrey, and C. Lutz. PDL with intersection and converse: satisfiability and infinite-state model checking. *Journal of Symbolic Logic*, 74(1):279–314, 2009.

[LM04] M. Lohrey and A. Muscholl. Bounded MSC communication. *Information and Computation*, 189(2):160 – 181, 2004.

[LMP07] S. La Torre, P. Madhusudan, and G. Parlato. A robust class of context-sensitive languages. In *LICS'07*, pages 161–170. IEEE Computer Society Press, 2007.

[LN11] S. La Torre and M. Napoli. Reachability of multistack pushdown systems with scope-bounded matching relations. In *CONCUR'11*, volume 6901 of *LNCS*, pages 203–218. Springer, 2011.

[QR05] S. Qadeer and J. Rehof. Context-bounded model checking of concurrent software. In *TACAS'05*, volume 3440 of *LNCS*, pages 93–107. Springer, 2005.

[Tho96] W. Thomas. Elements of an automata theory over partial orders. In *Proc. of POMIV'96*, volume 29 of *DIMACS*. AMS, 1996.

[Tra62] B. A. Trakhtenbrot. Finite automata and monadic second order logic. *Siberian Math. J*, 3:103–131, 1962. In Russian; English translation in *Amer. Math. Soc. Transl.* 59, 1966, 23–55.

[TW68] J. W. Thatcher and J. B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2(1):57–81, 1968.