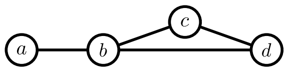
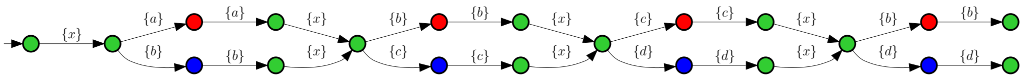


(a) An example instance of the 3-coloring problem.



(b) The sensori-computational device constructed from the graph coloring problem in Figure 6a.

Fig. 6: Reduction from an GRAPH-3C instance to an OVM instance. Strings traced on the device (in the lower figure) will map to sequences of outputs (here denoted $\{\text{red}, \text{green}, \text{blue}\}$) and their behavior indicates whether the graph in the top figure is 3-colorable.

states m_e^1 and m_e^2, two tail states t_e^1 and t_e^2. Then connect from h_e to middle state m_e^1 with label $\{v\}$, from h_e to m_e^2 with $\{w\}$—where v and w are the vertex names. Next, connect from m_e^1 to tail state t_e^1 with the $\{v\}$ label, and from m_e^2 to tail state t_e^2 with $\{w\}$. If e is not the last edge, then let e' denote the next edge to be processed, and connect from both tail states, t_e^1 and t_e^2, to the head state $h_{e'}$ with the arbitrarily introduced label $\{x\}$. When e is the first edge, create a separate initial state v_0 for the sensori-computational device, and connect from v_0 to h_e also with label $\{x\}$.

- All head and tail states, and also v_0, should output 0. The middle states m_e^i should output i.

Now, taking integer $n = 3$ along with sensori-computational device F, we have obtained an OVM instance. An example GRAPH-3C instance is given as shown in Figure 6a, and its constructed sensori-computational device is shown in Figure 6b.

Assume that G is 3-colorable. Let $k : V(G) \to \{0,1,2\}$ be a 3-coloring of G. Next, we will use k to create an observation variator (D, S_D), where $D = \{0,1,2\}$ and S_D is defined as follows: for all $v \in V(G)$, $\mathrm{S}_D(x, k(v)) = v$ and $\mathrm{S}_D(v, 0) = v$, $\mathrm{S}_D(v, 1) = x$. The only states with multiple outgoing edges are the head states h_e. Since $k(v) \neq k(w)$ for every edge (v,w), the observation variator will never create any non-determinism when applied at the head states. As a consequence, $\text{DELTA}_D(F)$ is deterministic, and output simulates F under the observation variator D.

Conversely, assume that there is an observation variator (D, S_D) such that $|D| = 3$ and there exists a sensori-computational device output simulating F under D. Let $D = \{d_0, d_1, d_2\}$. Then, we can construct a coloring function k accordingly: $k(v) = i$ if $\mathrm{S}_D(x, d_i) = v$, and $k(v) = 0$ for vertices that are not in the co-domain of S_D. Suppose k is not a valid 3-coloring of G. Then there must be two states v and w connected by an edge e but colored the same in G. As a consequence, $\text{DELTA}_D(F)$ must be non-deterministic after state h_e (going under the same $d \in D$ to both v and w, each with disjoint outputs) and fail to output simulate F, contradicting the assumption.

In summary, the graph G is colorable with 3 colors if and only if the sensori-computational device F has a derivative. Hence, we have reduced GRAPH-3C to OVM in polynomial time, and OVM is NP-hard. ■

D. *Aside: Trimming initial prefixes and absolute symbols*

Before moving on, a brief digression allays a potential niggling concern and serves as our first use of relation composition. Some readers might find it disconcerting that $\nabla|^{D}_{F}$'s definition includes the initial 'absolute' symbol y_0 for each sequence — this might be seen as a model for hybrid devices rather than sensors that report pure changes. One possible resolution is to introduce the new relation:

Definition 19 (Shave relation)**.** For a sensori-computational device F, the associated *shave relation* for $k \in \mathbb{N}$ is $\mathrm{S}_k \subseteq \mathscr{L}(F) \times Y(F)^*$ defined as follows:

0) $\varepsilon \, \mathrm{S}_k \, \varepsilon$, and

1) $y_0 y_1 \ldots y_m \, \mathrm{S}_k \, y_k y_{k+1} \ldots y_m$.

Intuitively, S_k trims away prefixes of length k. Then we can compose relations to give $\nabla|^{D}_{F} \fatsemi \mathrm{S}_1$, which removes the initial 'absolute' symbol y_0. Hence by analogy to Question 1, one might ask whether sensori-computational device F with variator (D, S_D) is $(\nabla|^{D}_{F} \fatsemi \mathrm{S}_1)$-simulatable? This can be answered most directly by modifying Algorithm 1, inserting a step between lines 30 and 31 which replaces the labels of edges departing v_0' with ε labels and performs an ε-closure thereafter. Then, state determinization (lines 31–40) will succeed if and only if the answer to the question is affirmative.

V. DATA ACQUISITION SEMANTICS: OBSERVATION SUB-SEQUENCES AND SUPER-SEQUENCES

It is worth closely scrutinizing the phrase 'event-based', a technical term used in multiple different ways. When people speak of event-based sensors or event sensors, they typically refer to a device capable of reporting changes. But in computing more generally, and in the sub-field of 'event processing' specifically, the phrase is used when a system is driven by elementary stimulus from without. Adopting that standpoint as an organizational principle in structuring software, one obtains event-based (or event-driven) software architectures. Such architectures will oftentimes impose synchronous operation. This contrasts with systems that must 'poll' to obtain information, and polling systems will usually do some sort of comparison or differencing between successive checks to detect a change. To de-conflate these slightly intertwined ideas, we must distinguish data acquisition from difference calculation. (Concretely: Definitions 9 & 10 dealt with the latter; Definitions 20 & 22 will deal with the former.)

The previous section determines if the set D of differences retains adequate information to preserve input–output behavior. All processing considered thus far preserves sequence lengths precisely, which may encode structured information. But such tight synchronous operation may be infeasible or an inconvenient mode of data acquisition for certain devices, and is inconsistent with, for example, event cameras. To emphasize this fact, recall Example 4. It showed, albeit only indirectly, how the lockstep flow of information from the world to the device affects whether D is adequate. In that example, the robot was allowed to turn 90° in a single step, whereas previously (in Example 3) it could do at most 45°. In practice, a 90° change in raw compass readings could happen for the robot in Example 3 if there was skew, or timing differences, or other non-idealities so that two actions occurred between sensor updates. One must be able to treat such occurrences, partly because they arise in practice, and partly because the implicit structure arising from synchronization is an artefact of the discrete-time model and is not something one wishes the event sensors to exploit. (The information baked in to time, especially as it is given privileged status, is often quite subtle, cf. [16].)

Thus, we next consider two other, practical ways in which sensors might generate the symbols that they send downstream. The first is that they may be *change-triggered* in that the sensori-computational device produces an output symbol only when a change has occurred. The second is for the element consuming the sensori-computational device's output to *poll* at some high frequency. Definitions that suffice to express each of these two modes, Definitions 20 and 22, are developed next.

(Note on notation: In what appears next, we consider sequences which may be from an observation set Y, or from a D of differences, or a combination, *etc.*; we use Σ as an arbitrary set to help emphasize this fact.)

Definition 20 (shrink)**.** Given $L \subseteq \Sigma^*$ and $\mathcal{N} \subseteq \Sigma$, then the $\mathcal{N}$-*shrink* is the single-valued, total function $\pi_{\mathcal{N}}$ defined recursively as follows:

$$\begin{aligned}\pi_{\mathcal{N}} : L &\to (\Sigma \setminus \mathcal{N})^*,\\ \varepsilon &\mapsto \varepsilon,\\ s_1 \dots s_m &\mapsto s_1 \dots s_m \text{ if } \forall j \in \{1,\dots,m\}, s_j \notin \mathcal{N},\\ s_1 \dots s_i \dots s_m &\mapsto \pi_{\mathcal{N}}(s_1 \dots s_{i-1} s_{i+1} \dots s_m) \text{ when } s_i \in \mathcal{N}.\end{aligned}$$

The intuition is that the $\mathcal{N}$-shrink drops all elements in $\mathcal{N}$ from sequences. The mnemonic for $\mathcal{N}$ is 'neutral' and the idea is that we will apply the $\mathcal{N}$-shrink relation in order to model change-triggered sensors; we can do this by choosing, for the set $\mathcal{N}$, symbols that reflect no change in signal. This assumes some subset of D will represent this no-change condition. Shortly, Section VI will address choices for D that guarantee such a subset is present.

Question 2. For sensori-computational device F and a set $\mathcal{N} \subseteq Y(F)$, is F $\pi_{\mathcal{N}}$-simulatable?

Question 2 (Constructive). For device F and $\mathcal{N} \subseteq Y(F)$, give some sensori-computational device G such that $G \sim F \pmod{\pi_{\mathcal{N}}}$ if any exists, or indicate otherwise.

Example 8 (Wall sensor, revisited)**.** In Example 1 the iRobot Create's traditional wall sensor produces a stream of `0`'s and `1`'s depending on the intensity of the infrared reflection it obtains. We discussed how, under output variator $D_2 = \{\perp, \top\}$, a derivative sensori-computational device exists that produces a stream of $\perp$s and $\top$s, the former occurring when there is no change in the presence/absence of a wall, and latter when there is. When one examines this derivative device under the $\{\perp\}$-shrink relation, we are considering whether the desired output can be obtained merely on a sequence of $\top$s. If the output depends on a count of the number of $\top$s, like the even- and odd-numbered doorways, then this is possible. If it depends on a count of the number of $\perp$s, or the interleaving of $\top$s and $\perp$s then it can not.

If some derivative, F' say, is $\pi_{\{\perp\}}$-simulatable, then it can operate effectively even if it is notified only when the wall-presence condition changes. It is in this sense that such F's are change-triggered. □

Algorithm 2: Shrink Transform: $\textsc{Shrink}_{\mathcal{N}}(F) \rightarrowtail G$

```
Input : A sensori-computational device $F$, a set $\mathcal{N}$
Output: A deterministic sensori-computational device $G$ if
        $G$ output simulates $F$ modulo $\pi_{\mathcal{N}}$; otherwise,
        return 'No Solution'
1  Make $G$, a copy of $F$
2  for $e' \in G.edges()$ do // Label replacement
3      for $\ell \in e'.labels()$ do
4          if $\ell \in \mathcal{N}$ then
5              Replace $\ell$ with $\varepsilon$ on edge $e'$
6  Merge $\varepsilon$-closure($V_0(G)$) as a single state $v'_0$ in $G$
7  $q := Queue([v'_0])$
8  while $q \neq \varnothing$ do // State determinization
9      $v' := q.pop()$
10     for $\ell \in v'.OutgoingLabels()$ do
11         $W := \{w : V(G) \mid v' \xrightarrow{\ell} w\}$
12         $W' := \cup_{w \in W} \varepsilon$-closure($w$)
13         $X := \cap_{w' \in W'} c(w')$
14         if $X = \varnothing$ then
15             return 'No Solution'
16         if $|W'| > 1$ then
17             Create a new state $w''$ inheriting all outgoing
                 edges of $W'$ in $G$, add a new edge $v' \xrightarrow{\ell} w''$,
                 remove $\ell$ from $v'$ to $W$
18             $c(w'') := X$, add $w''$ to $q$
19         else if $W'$ is not visited then
20             Add the single $w' \in W'$ to $q$
21 return $G$
```

A constructive procedure for addressing Question 2 appears in Algorithm 2. It operates as follows: first, it changes all the transitions bearing labels in $\mathcal{N}$ to ε-transitions between lines 1–5. It then shrinks those ε-transitions by determinizing the structure between lines 6–20. By doing so, it captures all the sequences possible after the shrink relation. The following lemma shows correctness:

Lemma 21. Algorithm 2 gives a sensori-computational device that output simulates F modulo $\pi_{\mathcal{N}}$ if and only if there exists a solution for Question 2.

Proof: $\Longrightarrow$: We show that if Algorithm 2 gives a sensori-computational device G, then G output simulates F modulo $\pi_{\mathcal{N}}$. First, we show that for every string $s \in \mathscr{L}(F)$, $\pi_{\mathcal{N}}(s) \in \mathscr{L}(G)$. (Because $\pi_{\mathcal{N}}$ is a total function, this ensures condition 1 in Definition 5 is met, and that $\mathscr{L}(G)$ covers the co-domain.) Let $s = s_1 s_2 \ldots s_n \in \mathscr{L}(F)$, and form $t = t_1 t_2 \ldots t_n$, where $t_i = s_i$ if $s_i \notin \mathcal{N}$, otherwise $t_i = \varepsilon$. Then t can be traced in the structure constructed from lines 1–5. According to Definition 20, $t = \pi_{\mathcal{N}}(s)$ as it has been obtained by dropping (or, equivalently replacing by ε) those symbols in $\mathcal{N}$. Since the ε-closure and determinimization steps preserve the language, the fact t can be traced in G means $\pi_{\mathcal{N}}(s) \in \mathscr{L}(G)$.

Second, we will show that for every string $s \in \mathscr{L}(F)$ and $t = \pi_{\mathcal{N}}(s) \in \mathscr{L}(G)$, we have $\mathscr{C}(F,s) \supseteq \mathscr{C}(G,t)$. To facilitate the analysis, we focus on the structure G_n that is constructed between lines 1–5, ('n' stands for a non-deterministic version of G). Whichever edges are crossed in tracing s on F, copies of those edges will be crossed in a tracing of t on G_n, and the state reached in G_n in this way must be a copy of the state reached by s in F. After state determinization (from G_n to G) in lines 6–20, the output of t in G must be a subset of $\mathscr{C}(F,s)$, i.e., $\mathscr{C}(G,t) \subseteq \mathscr{C}(F,s)$ because it takes the intersection of the outputs from the states that are non-deterministically reached (line 13).

$\Longleftarrow$: If there exists a solution G for Question 2, then we will show that Algorithm 2 will return a device. Suppose Algorithm 2 does not give a sensori-computational device. Then it returns a 'No Solution' at line 15. As a consequence, there exists a set of strings S which are mapped to the same image t via function $\pi_{\mathcal{N}}$ and $\cap_{s\in S}\mathscr{C}(F,s) = \varnothing$. As a consequence, there will be no appropriate output for t to satisfy for output simulation. Since G is a solution, $t \in \mathscr{L}(G)$. Any output chosen for t will violate output simulation, which contradicts with the assumption that G is a solution. ■

The essence of $\mathcal{N}$-shrink is that, via relation $\pi_{\mathcal{N}}$, it associates to a string all those strings we obtain by winnowing away symbols within $\mathcal{N}$. A second, related definition expands the set of strings by adding elements of $\mathcal{N}$. (Thinking, again, of changes within $\mathcal{N}$ as 'neutral'.)

Definition 22 (pump)**.** Given $L \subseteq \Sigma^*$ and $\mathcal{N} \subseteq \Sigma$, then the $\mathcal{N}$-*pump* is the relation $\mathrm{P}_{\mathcal{N}} \subseteq L \times \Sigma^*$ defined as follows: $\forall (s_1 \ldots s_m) \in L$,

0) $\varepsilon\, \mathrm{P}_{\mathcal{N}}\, \varepsilon$,
1) $(s_1 \ldots s_m)\, \mathrm{P}_{\mathcal{N}}\, (s_1 \ldots s_m)$,
2) $\forall b \in \mathcal{N}, k \in \{1, \ldots, \ell\}$,
$(s_1 \ldots s_m)\mathrm{P}_{\mathcal{N}}(t_1 \ldots t_\ell) \Longrightarrow (s_1 \ldots s_m)\mathrm{P}_{\mathcal{N}}(t_1 \ldots t_k\, b\, t_{k+1} \ldots t_\ell)$.

The intuition is that, to any string, the $\mathcal{N}$-pump associates all those strings with extra elements from $\mathcal{N}$ sandwiched between any two symbols, or at the very end. Notice that it does not place elements of $\mathcal{N}$ at the very beginning of the string.

Question 3. For sensori-computational device F and a set $\mathcal{N} \subseteq Y(F)$, is F $\mathrm{P}_{\mathcal{N}}$-simulatable?

Question 3 (**Constructive**)**.** For device F and $\mathcal{N} \subseteq Y(F)$, give a device G such that $G \sim F \pmod{\mathrm{P}_{\mathcal{N}}}$ if any exists, or indicate otherwise.

Example 9 (The 45° Minispot, again)**.** Reconsider Example 3, with a derivative compass device for the observation variator $D_3 = \{-, \emptyset, +\}$. Whenever some downstream consumer of the change-in-bearing information queries, an element of D_3 is produced. If it polls fast enough, we expect that it would contain a large number of $\emptyset$ values. Doubling the rate would (roughly) double the quantity of $\emptyset$ values. At high frequencies, there would be long sequences of $\emptyset$s and those computations on the input stream that are invariant to the rate of sampling would be $\mathrm{P}_{\{\emptyset\}}$-simulatable. □

Algorithm 3: Pump Transform: $\textsc{Pump}_{\mathcal{N}}(F) \rightarrowtail G$

```
Input  : A sensori-computational device $F$, a set $\mathcal{N}$
Output: A deterministic sensori-computational device $G$ if
        $G$ output simulates $F$ modulo $\mathrm{P}_{\mathcal{N}}$; otherwise,
        return 'No Solution'
1  Make $G$, a copy of $F$
2  Add a vertex $v_{\text{new}}$ and set $c(v_{\text{new}}) = c(v_0)$. Add edges from
     $v_{\text{new}}$ pointing to the destination of edges that depart
     $v_0 \in V_0(G)$
3  Update $G$'s initial vertex: $V_0(G) := \{v_{\text{new}}\}$.
4  for $v \in V(G) \setminus \{v_{new}\}$ do // Add self loops
5  |  Add a self loop at $v$ bearing labels $\mathcal{N}$
6  $q := Queue([v_{\text{new}}])$
7  while $q \neq \varnothing$ do // State determinization
8  |  $v := q.pop()$
9  |  for $\ell \in v.OutgoingLabels()$ do
10 |  |  $W := \{w : V(G) \mid v \xrightarrow{\ell} w\}$, $X := \cap_{w\in W} c(w)$
11 |  |  if $X = \varnothing$ then
12 |  |  |  return 'No Solution'
13 |  |  if $|W| > 1$ then
14 |  |  |  Create a new state $w'$ inheriting all outgoing
   |  |  |    edges of $W$ in $G$, add a new edge $v \xrightarrow{\ell} w$,
   |  |  |    remove $\ell$ from $v$ to $W$
15 |  |  |  $c(w') := X$, add $w'$ to $q$
16 |  |  else if $W$ is not visited then
17 |  |  |  Add the single $w \in W$ to $q$
18 return $G$
```

A procedure for answering Question 3 constructively appears in Algorithm 3. Its operation is as follows: first, it creates an initial state, reached (uniquely) by the ε string (line 3). Then, in lines 4–5, it adds self loops bearing labels to be pumped at all states (except the newly created one). Finally, it checks whether the resulting structure output simulates the input, and determinizes the structure between line 6–17. By doing so, it creates a deterministic structure to pump the elements in $\mathcal{N}$ using self loops. The following lemma addresses correctness.

Lemma 23. Algorithm 3 gives a sensori-computational device that output simulates F modulo $\mathrm{P}_{\mathcal{N}}$ if and only if there exists a solution for Question 3.

Proof: $\Longrightarrow$: We show that if Algorithm 3 gives a sensori-computational device G, then G output simulates F mod-