Toward a language-theoretic foundation for planning and filtering

Fatemeh Zahra Saberifar, Shervin Ghasemlou,
Dylan A. Shell, and Jason M. O'Kane

July 6, 2018

Abstract

We address problems underlying the algorithmic question of automating the co-design of robot hardware in tandem with its apposite software. Specifically, we consider the impact that degradations of a robot's sensor and actuation suites may have on the ability of that robot to complete its tasks. We introduce a new formal structure that generalizes and consolidates a variety of well-known structures including many forms of plans, planning problems, and filters, into a single data structure called a procrustean graph, and give these graph structures semantics in terms of ideas based in formal language theory. We describe a collection of operations on procrustean graphs (both semantics-preserving and semantics-mutating), and show how a family of questions about the destructiveness of a change to the robot hardware can be answered by applying these operations. We also highlight the connections between this new approach and existing threads of research, including combinatorial filtering, Erdmann's strategy complexes, and hybrid automata.
Keywords: planning; combinatorial filter; design automation

1 Introduction

The process of designing effective autonomous robots—spanning the selection of sensors, actuators, and computational resources along with software to govern that hardware—is a messy endeavor. There appears to be little hope of fully automating this process, at least in the short term. There would, however, be significant value in *design tools* for roboticists that can manipulate partial or tentative designs, in interaction with a human co-designer. For example, one might imagine algorithms that answer questions about the relationship between a robot's sensors and actuators and that robot's ability to complete a given task.

A crucial requirement for this kind of automation is a general formal model that can describe, in a precise way, a robot's sensing and actuation capabilities in the context of its interaction with an environment. To that end, this paper lays theoretical groundwork for reasoning about sensors and actuators and their associated estimation and planning processes. The underlying goal is to strengthen the link between idealized models and practical—that is, imperfect, imprecise, and limited— realizations of those idealized models in actual, available hardware.

To motivate these questions more concretely, consider the pair of scenarios that follows.

Example 1.1. Your robot is stationed on a distant planet and, though fully operable initially, has recently encountered a problem. It appears that debris has become affixed to one of the sensors. Should operations be altered by taking more conservative paths around obstacles because the robot's position estimates now involve greater error than previously? Or has the mission been entirely compromised? Assuming that the debris cannot be dislodged, what tasks are still feasible?

Example 1.2. You lead an R&D team who have built and tested a successful prototype robot, which performs cosmetic services (e.g., manicures, pedicures, facials, hair-weaves, etc.) efficiently and safely. Then…disaster! You discover that the sensor provided to your factory in bulk (say S_1), differs from the device (S_0) supplied by the same manufacturer to the team who built and tested the prototype. A successful redesign of the robot might require answers to these kinds of questions: Can S_1 be used directly as a plug-and-play replacement for S_0? If not, can we adjust some software parameters to make it work? Which parameters and what should the adjustments be? If S_1 necessarily incurs a loss in performance, how can this be understood—perhaps only the hair-styling functionality is affected? Supposing we can procure S_0 at greater cost through another vendor, is this worth doing?

Underlying these scenarios is the problem of how to ascertain whether or not a particular modification to one's model of a robot is destructive for a given task. In this paper, we formalize this question, providing theoretical foundations as well as algorithms to address problems of this type. This can be seen visually in Figure 1.

This paper makes several new contributions.

1. We introduce, in Section 2, the notion of an *interaction language*, which models the interactions between a robot and its environment using the theory of formal languages. This approach unifies several previously distinct conceptual classes of object.

2. We contribute, in Section 3, a general representation called a *procrustean graph*[1] for interaction languages. This representation is constructive, in that

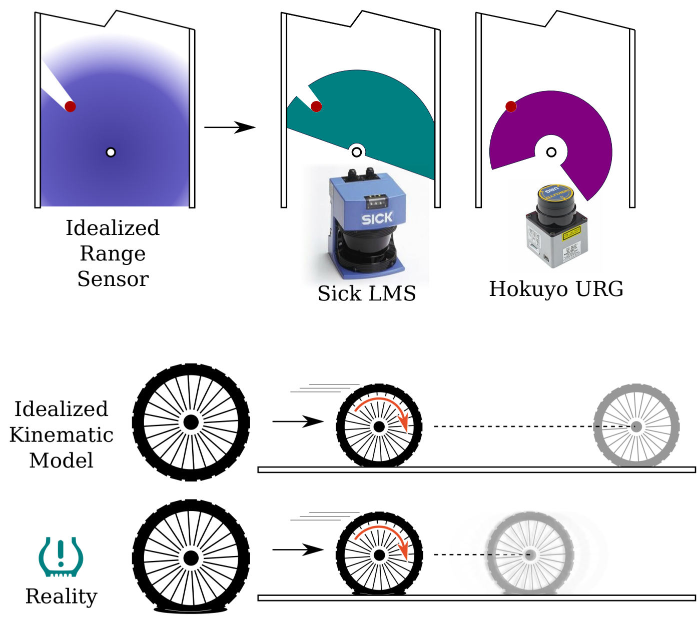

Figure 1: Given a specification of robot capabilities—encompassing both sensors (top) and actuators (bottom)—and a task, which changes to those sensors or actuators make the task infeasible? This question is fundamental because it addresses the link between idealized models and practical (imperfect, imprecise, and limited) realizations of those idealized models in hardware, and because its answers can can lead to insight about weakest robots that suffice for a given task.

it can be used to instantiate a data-structure from which various questions can be posed and addressed concretely.

3. We show, in Section 4, how to model degradations to sensing and action capabilities in this framework as *label maps*. We address the question of deciding whether a label map is *destructive*, in the sense of preventing the achievement of a previously-attainable goal, for both filtering and planning problems, in Sections 5 and 6 respectively. We also prove that the broader question of finding a non-destructive label map that is, in a certain sense, maximal, is NP-hard.

The dénouement of the paper includes a review of related work interleaved with a discussion of the outlook for continued progress (Section 7) and some concluding remarks (Section 8).

Preliminary versions of this work appeared in 2016 at RSS (Saberifar et al., 2016) and WAFR (Ghasemlou et al., 2016).

2 Actions, observations, and interaction languages

We begin with some basic definitions for modeling the interaction between an agent or robot and its environment. The robot executes *actions* drawn from a non-empty *action space* U; the environment yields *observations* drawn from a non-empty *observation space* Y. We assume that $\mathrm{U} \cap \mathrm{Y} = \varnothing$. Neither need necessarily be finite. Noting the duality between actions and observations—an observation can be viewed merely as an 'action by nature'—we treat actions and observations as specific subtypes of a more general class of events.

Definition 2.1 (event)**.** *An* event *is an action or an observation. The* event space *is* $\mathrm{E} \coloneqq \mathrm{U} \cup \mathrm{Y}$.

Definition 2.2 (event sequence)**.** *An* event sequence *over E is a finite sequence of events $e_1 \cdots e_m$ drawn from E. An event sequence is called* action-first *if* $e_1 \in \mathrm{U}$*, or* observation-first *if* $e_1 \in \mathrm{Y}$*. Likewise, an event sequence is called* action-terminal *if* $e_m \in \mathrm{U}$*, or* observation-terminal *if* $e_m \in \mathrm{Y}$.

Definition 2.3 (successor)**.** *For two event sequences s_1 and s_2 over the same event set* E*, we say that s_2 is a* successor *of s_1, if there exists some* $e \in \mathrm{E}$ *such that* $s_2 = s_1 e$.

[1]Named for Procrustes (Προκρούστες), son of Poseidon, who, according to myth, took the one-size-fits-all concept to extremes.

In what follows, we describe sets of event sequences using standard notation for regular expressions: Concatenation (represented implicitly using juxtaposition), alternation (using the binary $+$ operator), the empty sequence (the ε symbol), and the Kleene star (the unary $*$ operator).

Definition 2.4 (interaction language)**.** *An* interaction language L *over an event space* E *is a set of event sequences which is either*

1. *a subset of* $(\mathrm{UY})^*(\mathrm{U} + \{\varepsilon\})$*, or*
2. *a subset of* $(\mathrm{YU})^*(\mathrm{Y} + \{\varepsilon\})$*,*

and which is closed under prefix.

The intuition behind interaction languages is that they describe sequences of events which alternate between action and observation. The definition admits two distinct types of interaction languages: those whose members begin with actions (hereafter, *action-first languages*) and those whose members begin with observations (*observation-first languages*).

Interaction languages encode an interaction between an agent or robot (which selects actions) and its environment (which dictates the observations made by the robot). The definition is intentionally ecumenical in regard to the nature of that interaction, because we intend this definition to serve as a starting point for more specific structures which, once specific context and semantics are added, lead to special cases that represent particular (and familiar) objects involving planning, estimation, and the like.

The prefix-closure requirement in Definition 2.4 ensures that for all event sequences $e_1e_2\cdots e_m \in \mathrm{L}$, every subsequence $e_1\cdots e_k$ with $k < m$ is also in L. If a language expresses properties of some structured interaction, then some event sequences are excluded from that language. In such cases then the prefix condition captures the idea that part of the way through a sequence, or even in a sequence stopped short, anterior structure is present.

The examples that follow illustrate a few different kinds of interaction languages.

Example 2.1 (Filters)**.** Filtering, in very broad terms, refers to any process by which observations are processed to produce specified outputs. That is, a specification of a filter tells us, for any plausible history of observations that an agent might have made, what the correct output from the filter should be. Filters are, of course, objects of intense and sustained interest within the robotics community.

In light of Definition 2.4, we can describe a filter as an action-first interaction language L, in which the filter's outputs are modeled as actions emit_x for various outputs x.

Example 2.2 (Schoppers's universal plans)**.** For observable domains, a universal plan (Schoppers, 1987) is a specification of an appropriate action for each circumstance that an agent might find itself in. This kind of model can be expressed as an interaction language L in which each observation corresponds to a world state, and for each observation $y \in Y$, every event sequence ending in y has exactly one successor, and that these successors are all formed by appending a single unique action u. The intuition is that this unique u indicates the action that should be taken when the robot is in the state corresponding to y.

Example 2.3 (Erdmann-Mason-Goldberg-Taylor plans)**.** Several classic papers (Erdmann and Mason, 1988; Mason et al., 1988; Goldberg, 1993) find policies for manipulating objects in sensorless (or nearly sensorless) conditions. The problems are usually posed in terms of a polygonal description of a part; the solutions to such problems are sequences of actions. Such plans can be expressed as interaction languages containing all event sequences in which the actions (e.g., a squeeze-grasp or a tray tilt at a particular orientation) in each sequence guarantee a known final orientation of the part regardless of its unknown initial orientation. In the event sequences of the interaction language, these actions are interleaved with with a special η which constitutes the sole element in Y, acting as dummy 'no observation' token.

Example 2.4 (Counting amidst beams)**.** As another example, consider a system in which an unknown number of agents moves through a known network of rooms. Their movements are observed by discrete beam sensors that detect the passage of an agent from one room to another, but not the identity of that agent. Actions allow barriers between the rooms to be opened or closed. (Similar problems were addressed by Erickson et al. (2014) and Gierl et al. (2014).) The evolution of this kind of system can be modeled as an interaction language whose event sequences are those that correspond to valid traces of this system.

Figure 2 shows an example, in which a observations indicate an agent moving from r_1 to r_2, and b observations indicate an agent moving from r_2 to r_1. Some unknown number of agents begins in r_1, whereas r_2 is initially empty. Interestingly, even for this very simple case, the interaction language is not a regular language. To see this, note that the number of a observations must be no less than the number of b observations in any event sequence that occurs in this system—no agent can leave r_2 if that room is already empty—and no finite-state automaton can do this kind of 'counting' for arbitrarily many agents.

Next, we introduce some definitions for reasoning about relationships between pairs of interaction languages, in terms of the event sequences that are shared between them. In Section 6, we model both planning problems and plans themselves

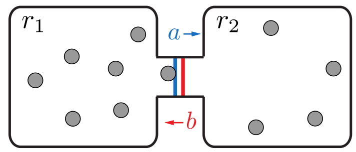

Figure 2: A simple environment with two regions and beam sensors in the corridor connecting them; a grey body is moving from region r_1 to region r_2 (drawn after Erickson et al. (2014), but simplified). For an unknown number of grey agents, all of which start in r_1. The interaction language for this system is not a regular language.

via interaction languages. The next definitions will be helpful for formalizing the relationships between those two languages.

At the simplest level, we recall the distinction between action-first and observation-first interaction languages.

Definition 2.5 (akin)**.** *Two interaction languages* $\mathrm{L_A}$ *and* $\mathrm{L_B}$*, both over the same set of events, are* akin *if they are both action-first languages, or they are both observation-first languages.*

We can also consider the set of event sequences shared between a pair of akin interaction languages.

Definition 2.6 (joint event sequence)**.** *Given two interaction languages* $\mathrm{L_A}$ *and* $\mathrm{L_B}$ *that are akin, an event sequence* s *is a* joint event sequence *if* $s \in \mathrm{L_A}$ *and* $s \in \mathrm{L_B}$.

As an aside, we note that the structure required of interaction languages is preserved when we consider only the set of joint event sequences for a pair of languages.

Theorem 2.1 (Joint event sequences form an interaction language)**.** *For any two interaction languages* L_A *and* L_B *that are akin, the set* $L_A \cap L_B$ *of their joint event sequences is itself an interaction language.*

Proof. Follows directly from Definitions 2.4 and 2.6. □

Of particular interest in the context of planning will be pairs of languages for which there is some bound on the longest joint event sequence. The next definition makes that intuition more precise.

Definition 2.7 (finite on)**.** *Given two akin event languages* $\mathrm{L_A}$ *and* $\mathrm{L_B}$*, if there exists an integer* k *that bounds the length of every joint event sequence of* $\mathrm{L_A}$ *and* $\mathrm{L_B}$*, we say* $\mathrm{L_A}$ *is* finite on $\mathrm{L_B}$.

Note that 'finite on' is a symmetric relation, though the way it is written does not immediately emphasize this fact. Some caution is likely warranted as the definitions have made a subtle departure from standard language theory. In particular, finite on does not require that the either of the interaction languages be finite sets, but only that there exist some finite bound on the lengths of their joint event sequences. In fact, the set of joint sequences may form an infinite set since U or Y need not be finite; Definition 2.7 requires only a bound on the length of the sequences.

Finally, we consider a notion of 'compatibility' between two interaction language.

Definition 2.8 (safety)**.** *Given two event languages* $\mathrm{L_A}$ *and* $\mathrm{L_B}$*, both akin to one another,* $\mathrm{L_A}$ *is* safe on $\mathrm{L_B}$ *if, for every joint event sequence* s*, the following holds:*

1. *if* s *is observation-terminal, then for every successor* s' *of* s*,*

$$s' \in \mathrm{L_A} \implies s' \in \mathrm{L_B};$$

2. *if* s *is action-terminal, then for every successor* s' *of* s*,*

$$s' \in \mathrm{L_B} \implies s' \in \mathrm{L_A}.$$

To understand the intuition, imagine a joint event sequence constructed one event at a time, with actions selected via the successors in L_A of the current event sequence and observations selected via the successors of L_B. When the next event should be an action, Definition 2.8 requires that L_B must be 'ready' (in the sense of containing at least one suitable event sequence) for any action that might be selected from the successor event sequences in L_A. Likewise, when the next event should be an observation, L_A must be ready for any observation that might be selected from the successor event sequences in L_B.

Though the symmetry in the definition is perhaps aesthetically pleasing, one should not be misled: safety of L_A on L_B does not imply that L_B is safe on L_A. Moreover, safety is not transitive. (Note that appearing on the left differs from appearing on the right, as the quantifiers shift.) However, safety is indeed is reflexive (L_A is always safe on L_A).

3 Procrustean graphs and set labels

3.1 Procrustean graphs

The languages and other definitions in the preceding section express the fact that interactions may possess structure. Though formal, they are not a directly useful construct for algorithmic manipulation nor for reasoning about causality in the aspects involved. To automate (or help automate) design-time processes, we introduce a new representation called a procrustean graph for a broad class of interaction languages, based on graphs with transitions labeled by sets.

Definition 3.1 (p-graph)**.** *A* procrustean graph (p-graph) *over event space* E *is a finite edge-labeled bipartite directed graph in which*

1. *the finite vertex set, of which each member is called a* state*, can be partitioned into two disjoint parts, called the* action vertices $\mathrm{V_u}$ *and the* observation vertices $\mathrm{V_y}$*, with* $\mathrm{V} = \mathrm{V_u} \cup \mathrm{V_y}$*,*

2. *each edge* e *originating at an action vertex is labeled with a set of actions* $\mathrm{U(e)} \subseteq \mathrm{U} \subset \mathrm{E}$ *and leads to an observation vertex,*

3. *each edge* e *originating at an observation vertex is labeled with a set of observations* $\mathrm{Y(e)} \subseteq \mathrm{Y} \subset \mathrm{E}$ *and leads to an action vertex, and*

4. *a non-empty set of states* $\mathrm{V_0}$ *are designated as* initial states*, which may be either exclusively action states (*$\mathrm{V_0} \subseteq \mathrm{V_u}$*) or exclusively observation states (*$\mathrm{V_0} \subseteq \mathrm{V_y}$*).*

A small example, intended to illustrate the basic intuition of the definition, follows.

Example 3.1 (wheels, walls, and wells)**.** Figure 3 show a p-graph that models a Roomba-like robot that uses single-bit wall and cliff sensors to navigate through an environment. Action states are shown as unshaded squares; observation states are shaded circles. Action labels are subsets of $[0, 500] \times [0, 500]$, of which each element specifies velocities for the robot's left and right drive wheels, expressed in mm/s. Observations are bit strings of length 2, in which the first bit is the output of the wall sensor, and the second bit is the output of the cliff sensor.

P-graphs bear a close relationship to interaction languages—they describe sets comprised of sequences of actions and observations that alternate. The intention is for p-graphs to serve as concrete data structures for representing interaction languages. This helps realize the paper's objective, which is to treat p-graphs, in a general sense, as first-class objects, suitable for manipulation by automated means.

Before formalizing the details of the connection between p-graphs and interaction languages, we make a minor detour to show that p-graphs are sufficiently rich to describe things that have been of broad interest to roboticists for a long time.

Example 3.2 (Combinatorial filters)**.** Recall from Example 2.1 that filtering problems can be cast in terms of interaction languages in which the filter outputs are

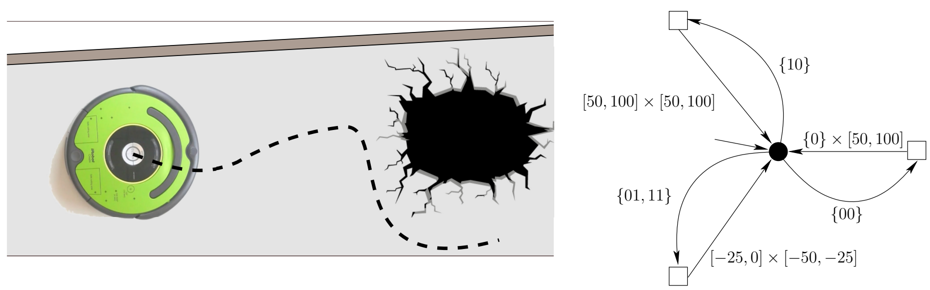

Figure 3: [left] A differential drive robot with sensors for obstacles, both positive (walls) and negative (holes). [right] An example p-graph that models behavior in which the robot follows a wall while avoiding negative obstacles. This graph, and those that follow, have solid circles to represent elements of V_u, and empty squares for V_y. The arcs are labeled with sets; those that leave the central vertex have two digits, the first digit is '1' iff the wall is detected by the IR sensor on the left-hand side; the second digit is '1' iff the downward pointing IR sensor detects a cliff. The actions, on the edges leaving squares, represent sets of left and right wheel velocities, respectively.

encoded as actions of the form emit_x. A particular class of filters that is well suited to representation as p-graphs are the *combinatorial filters*. As formalized by LaValle (2012), combinatorial filters are discrete expressions of estimation problems. More precisely, combinatorial filters are finite-state transition systems in which each state has a specific output associated with it. Such filters can be cast as p-graphs by having observations and observation transitions exactly as in the filter, but with action vertices having only a single out-edge that is labeled with a singleton set bearing the output (which, as in Example 2.1, we label emit_x for output each x). Figure 4 shows a canonical example in which the property of interest is whether two agents, in an annulus-shaped environment with three beam sensors, are apart or not.

Example 3.3 (P-graphs for universal plans)**.** The interaction languages for universal plans introduced in Example 2.2 can be cast as p-graphs in a straightforward way. The p-graph has a single observation vertex, with one uniquely-labeled out-edge corresponding to each world state, and one action state for each of the distinct available actions. See Figure 5.

Example 3.4 (Erdmann-Mason-Goldberg-Taylor plans)**.** Figure 6 shows an example of a sensorless manipulation plan, in the form described in Example 2.3, expressed as a p-graph. Of particular note is the fact that this plan exhibits an unexpected dimension of nondeterminism: at each step it indicates sets of allowable actions, rather than a single predetermined one. This degree of 'choice' in the actions appears in the interaction language as a large collection of individual event sequences, but is expressed compactly within the p-graph. Also of note is

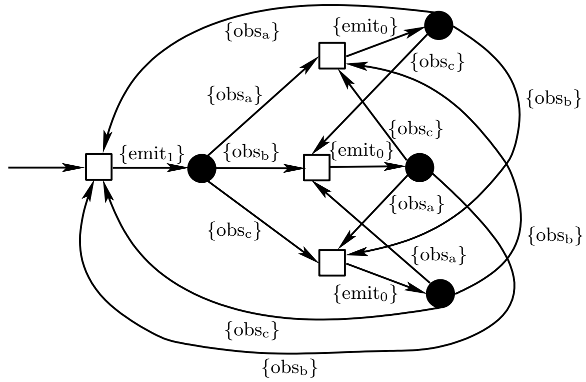

Figure 4: The 'agents together' filter devised by Tovar et al. (2014) expressed as a p-graph. The emit_0 action indicates that the agents are separated by a beam, and emit_1 indicates that the agents are together.

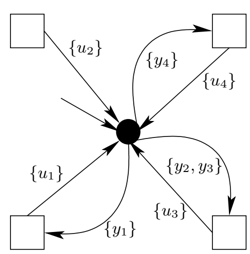

Figure 5: A universal plan expressed as a p-graph.

that, generally, the graphs of knowledge states searched to produce such plans are themselves p-graphs.

Example 3.5 (Nondeterministic graphs)**.** Recent work by Erdmann (2010; 2012) encodes planning problems using finite sets of states, along with nondeterministic actions represented as collections of edges 'tied' together into single actions. One might convert such a graph to a p-graph by replacing each group of action edges with an observation node, with an outgoing observation edge for each edge constituting the original action. Figure 7 shows an example.

The intent in these examples is to illustrate that p-graphs form a general class that unifies, in a relatively natural way, a number of different kinds of objects that have been studied over a long period of time. The particular constraints applied in each case impose certain kinds of structure that proved useful in the original context.

While graph and graph-like objects appear in the prior work in various guises, few have formalized the semantics of those objects by connecting them to the lan-

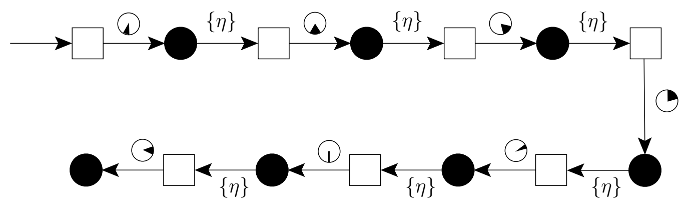

Figure 6: A plan for orienting an Allen wrench via tray tilting, expressed as a p-graph. Action edges are labeled with sets of azimuth angles for the tray. There is a single dummy observation, η. This plan is shown as Figure 2 in Erdmann and Mason (1988).

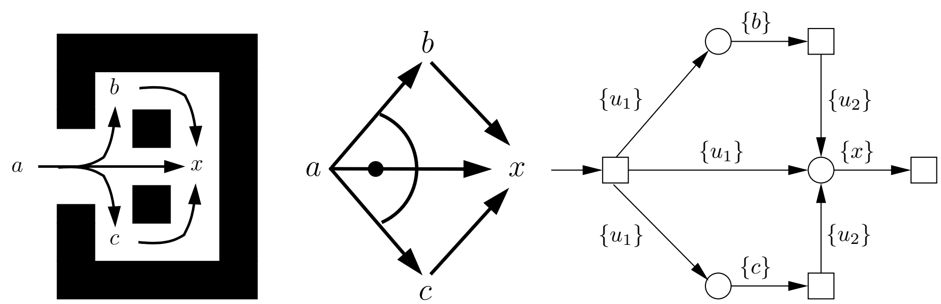

Figure 7: [left] A planning problem due to Erdmann (2012). [middle] The nondeterministic graph of this problem. [right] An equivalent p-graph.

guages they induce. The definitions we present next make precise the way in which a p-graph is an implicit definition of an interaction language.

Definition 3.2 (transitions to)**.** *For a given p-graph G and two states $v, w \in V(G)$, a sequence of events $e_1 \cdots e_k$* transitions in G from v to w *if there exists a sequence of states $v_1, \ldots, v_{k+1}$, such that*

1. *$v_1 = v$,*
2. *$v_{k+1} = w$, and*
3. *for each $i = 1, \ldots, k$, there exists an edge $v_k \xrightarrow{E_k} v_{k+1}$ for which $e_k \in E_k$.*

The states v and w need not be distinct; for every v, the empty sequence transitions in G from v to v. Longer cycles may result in non-empty sequences of states that start at some v and return.

Definition 3.3 (valid)**.** *For a given p-graph G and a state $v \in V(G)$, a sequence of events $e_1 \cdots e_k$ is* valid *from v if there exists some $w \in V(G)$ for which $e_1 \cdots e_k$ transitions from v to w.*

Observe that the empty sequence, ε, is valid from all states in any p-graph.

Definition 3.4 (execution)**.** *An* execution *on a p-graph G is a sequence of events valid from some start state in $V_0(G)$.*

The preceding definitions prescribe when a sequence is valid on a p-graph, placing few restrictions on the sets involved. There are several instances of choices recognizable as forms of non-determinism: (i) there may be multiple elements in V_0; (ii) from any $v \in V_u$ some action u may be an element in sets on multiple outgoing action edges; (iii) similarly, from any $w \in V_y$ some observation y may qualify for multiple outgoing observation edges.

We can now 'close the loop' between interaction languages and p-graphs.

Definition 3.5 (induced language)**.** *Given a p-graph G, its* induced language *is the set of all of its executions. It is denoted $\mathcal{L}(G)$.*

Theorem 3.1 (induced languages are interaction languages)**.** *For any p-graph G, the induced language $\mathcal{L}(G)$ is an interaction language.*

Proof. Because G is bipartite, with its states partitioned into action states and observation states, all of its executions alternate between actions and observations. Moreover, if $\mathrm{V_0} \subset \mathrm{V_u}$, then its non-empty executions begin with actions, matching the first regular expression in Definition 2.4. If $\mathrm{V_0} \subset \mathrm{V_y}$, then the non-empty executions if G begin with actions, matching the second regular expression in Definition 2.4.

It remains only to confirm that $\mathcal{L}(G)$ is prefix-closed. Consider some execution $s = e_1 e_2 \cdots e_m \in \mathrm{L}$, and some prefix of s, denoted $s' = e_1 \cdots e_k$ with $k < m$. We need to show that $s' \in \mathcal{L}(G)$. We know from Definition 3.3 that s transitions from some start state v to some final state w. Therefore, via Definition 3.2, we know that there exists sequence of states $v_1, \ldots, v_{m+1}$ in G, with $v_1 = v$, reached by followed edges labeled with events in s. But considering only $v_1, \ldots, v_{k+1}$, we see that s' is valid from v_1 as well. Therefore, $s' \in \mathcal{L}(G)$. □

This theorem establishes a tight relationship between interaction languages and p-graphs. Every p-graph induces an interaction language (though some interaction languages cannot be expressed as p-graphs with finitely many states, cf. Example 2.4). Thus, we can meaningfully apply terms defined for interaction languages to p-graphs as well: Given two p-graphs G_1 and G_2, one might refer to the set of joint executions (that is, joint event sequences) of G_1 and G_2. We might say that G_1 and G_2 are akin to one another, or that G_1 is finite on G_2, or that G_1 is safe on G_2. These kinds of statements should be read as referring to the interaction languages induced by the p-graphs.

Example 3.6 (Pentagonal world)**.** Figure 8 presents concrete realizations of several of the preceding definitions in a single scenario. A robot moves in a pentagonal

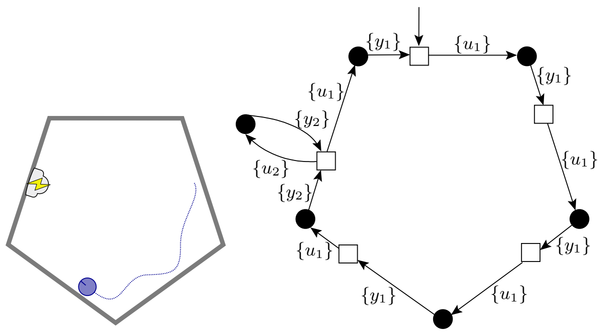

Figure 8: [left] A robot wanders around a pentagonal environment; the segment with the lightning-bolt contains a battery charger. [right] A p-graph model of this world.

environment. Information—at least at a certain level of abstraction—describing the structure of the environment, operation of the robot's sensors, its actuators, and their inter-relationships is represented in the p-graph associated with the scenario. The induced interaction language is

$$\mathcal{L}(G) = \mathrm{Pref}((u_1 y_1 u_1 y_1 u_1 y_1 u_1 y_2 (u_2 y_2)^* u_1 y_1)^*),$$

in which $\mathrm{Pref}(\cdot)$ denotes the prefix-closure of its language operand. Both filtering and planning questions can be posed as problems on this interaction language, as represented in this p-graph.

3.2 Labels

To keep the model amenable to direct algorithmic manipulation, we have required that a p-graph consist of only finitely many states. However, the labels for each edge, either $\mathrm{U}(e)$ or $\mathrm{Y}(e)$, are sets that need not be finite. This detail is important for modeling real systems. For example, robots typically have observation spaces which are large or infinite—including most nontrivial real sensor systems—in which it would be, at best, computationally intractable to list observations individually. The same can be said for actions too.

We can permit labels that describe infinite sets if some simple operations on the set algebra over U and Y are available. Any observation-originating edge e is labeled with the set $\mathrm{Y}(e)$, such that $\mathrm{Y}(e) \in 2^{\mathrm{Y}}$. The analogous relation holds for action-originating edges too. In what follows, we assume that both 2^{U} and 2^{Y} are equipped with the following six operations:[2]

[2]To save presenting distracting technicalities, we use 2^{U} for the set algebra over U, though the need for finite constructions usually means that the algebra is a proper subset of the powerset.

1–3. UNION, which accepts two labels and computes a new label representing their union, along with INTERSECTION and DIFFERENCE, which operate *mutatis mutandis* for the set intersection and set difference operations.

4. EMPTY, which accepts a label and returns TRUE if and only if the label represents the empty set.

5. CONTAINS, which accepts a label and an event, and decides whether that event is member of the set represented by that label.

6. REPRESENTATIVE, which accepts a non-empty label and returns an event contained in the set represented by that label.

Any data structure capable of answering these queries is suitable for representing the labels in the algorithms in this paper. Some examples follow.

Example 3.7. Suppose $\mathrm{U} = \mathbb{R}$ or $\mathrm{Y} = \mathbb{R}$. Since each label should represent a set of real numbers, one option is to let each represent a finite union of real intervals. The intervals may be bounded or unbounded. Each interval may also be open, closed, or half-closed. Figure 9 shows an example. To represent a label from this label space, we use a data structure with three parts:

1. A list of $n \in \mathbb{N}_0$ real number *endpoints* $e_1, \ldots, e_n \in \mathbb{R}$.

2. A list of $n+1$ boolean *interval flags* $f_1, \ldots, f_{n+1}$. The interpretation is that, for each $1 < j < n$, the real numbers between e_j and e_{j+1} are included in the set if and only if f_j is TRUE. At the extremes, real numbers less than e_1 are in the set when f_1 is TRUE, and likewise numbers greater than e_n are in the set when f_n is TRUE.

3. A list of n boolean *endpoint flags* $p_1, \ldots, p_n$, with the semantics that, for any $1 \leq j \leq n$, the real number e_j is in the label's observation set if and only if p_j is TRUE.

Note that any finite union of real intervals (including, for example, the empty set and the full real line, which have $n = 0$) can be expressed in this format.

The UNION, INTERSECTION, and DIFFERENCE operations can be implemented by performing a left-to-right sweep, adding endpoints and flags appropriately to the resulting label. The EMPTY method requires a simple check for any endpoint flags or interval flags that are TRUE. The CONTAINS check can be implemented by a binary search for the correct interval, followed by a check against the relevant flag. REPRESENTATIVE should return an element, either an endpoint or in the interior of an interval (in the general case, perhaps the midpoint between two endpoints) for which the corresponding flag is TRUE.

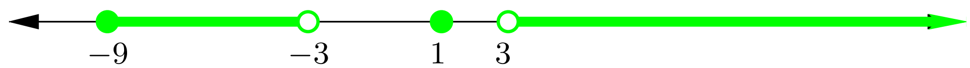

Figure 9: An interval label for the set $[-9,-3) \cup \{1\} \cup (3, \infty)$. The label data structure has 4 endpoints $(-9, -3, 1, 3)$, 5 interval flags (FALSE, TRUE, FALSE, FALSE, TRUE), and 4 endpoint flags (TRUE, FALSE, TRUE, FALSE).

Example 3.8. Labels that represent a finite number of events—as is the case for many simple sensors such beam detectors or bump sensors, or simple actuators with a discrete modes of operation—can be modeled by storing the elements explicitly in almost any container data structure, such as a balanced binary tree or a hash table.

Example 3.9. We expect that a common case will involve action or observation sets that are composed, via Cartesian product, from simpler sets. That is, we may generally have $\mathrm{X} = \mathrm{L}_1 \times \cdots \times \mathrm{L}_m$, in which each L_i is a set for which we have the requisite operations, and X is U or Y. In such a case, we can define a set algebra over X in which each label represents a union of Cartesian products of sub-labels, in the form $\bigcup_i \left(\ell_1^{(i)} \times \cdots \times \ell_m^{(i)}\right)$, $\ell_k^{(i)} \subseteq \mathrm{L}_k$, where $i \in \{1, \ldots, n\}$. Under this representation, a UNION between labels becomes a mere concatenation of Cartesian product lists. The INTERSECTION operation requires pairwise intersections between each of the constituent Cartesian products of each of the two labels:

$$\left[\bigcup_i \left(\ell_1^{(i)} \times \cdots \times \ell_m^{(i)}\right)\right] \cap \left[\bigcup_j \left(p_1^{(j)} \times \cdots \times p_m^{(j)}\right)\right] = \bigcup_i \bigcup_j \left((\ell_1^{(i)} \cap p_1^{(j)}) \times \cdots \times (\ell_n^{(i)} \cap p_m^{(j)})\right).$$

The DIFFERENCE operation is similar, but first requires a refinement of the labels (see Section 3.2.1 below) along each dimension.

Example 3.9 also illustrates that while it is natural to think of labels as the descriptions of sets borne on edges, such as either $\mathrm{U}(e)$ or $\mathrm{Y}(e)$ for some e, it is also meaningful to think of sets which are basic constituents from which to make up such labels. For this reason, in what follows we use the general notation of ℓ_i, which can describe either sets of actions or observations.

3.2.1 Label refinement

Several of the algorithms in subsequent sections rely on a subroutine to compute of a *refinement* of a set of labels. Specifically, in several places we need an

Algorithm 1: REFINELABELS($\ell_1, \ldots, \ell_n$)

```
Input  : A list of labels, ℓ1, ℓ2, ..., ℓn.
Output: A list of refined labels ℓ'1, ℓ'2, ..., ℓ'm.
// Construct the union of all the input labels
r ← ℓ1
for ℓ ∈ {ℓ2, ..., ℓn} do
    r ← UNION(r, {ℓ})
R ← {r}
// Refine each at at time
for ℓ ∈ {ℓ1, ..., ℓn} do
    R' ← ∅
    for r ∈ R do
        R'.append(INTERSECTION(r, ℓ)); // The part inside r...
        R'.append(DIFFERENCE(r, ℓ)); // ...and then the rest
    R ← R'
return R
```

algorithm that accepts as input an unordered set of labels $\ell_1, \ldots, \ell_n$, and produces as output an unordered set of labels $\ell'_1, \ldots, \ell'_m$, such that $\bigcup_i \ell_i = \bigcup_j \ell'_j$ and, for each $\ell' \in \{\ell'_1, \ldots, \ell'_m\}$ and each $x_1, x_2 \in \ell'$, we have

$$\big\{\ell \in \{\ell_1, \ldots, \ell_n\} \mid x_1 \in \ell\big\} = \big\{\ell \in \{\ell_1, \ldots, \ell_n\} \mid x_2 \in \ell\big\}.$$

The intuition is, given a set of labels, to compute a partition of the events spanned by those labels. This partition should be fine enough to separate the input labels from one another, in the sense that the set of corresponding input labels is constant across all events in each output label. Such a partition is valuable because it enables us to ‘drop down’ from the level of sets to the level of individual events, by selecting a REPRESENTATIVE from each of the output labels, without danger of missing any structure inherent to the input label set.

Algorithm 1 shows how one can perform this operation in a general way, for any labels that support the UNION, INTERSECTION, and DIFFERENCE operations. The algorithm starts with a single label representing the complete set of relevant events, and then refines that partition using each of the input labels.

We note, however, that for certain kinds of labels, such as the interval labels introduced in Example 3.7, it may be practical and efficient to implement this operation directly, utilizing the internal details of the label data structure, rather than this generalized approach. For the interval label space introduced in Example 3.7, the label refinement operation can be implemented directly. Figure 10 shows an example of the computation. The approach is to form a combined, sorted list of all of the endpoints for each of the input labels, and then form the refined output

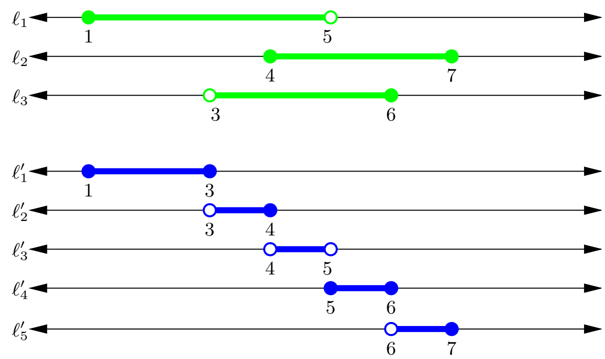

Figure 10: An example of label refinement with interval labels. The three input labels, representing the overlapping intervals $[1, 5)$, $[4, 7]$, and $(3, 6]$, are refined into six disjoint output labels $[1, 3]$, $(3, 4]$, $(4, 5)$, $[5, 6]$, and $(6, 7)$.

labels using a left-to-right sweep, starting a new label each time the set input labels touched by the sweep line changes.

3.3 Basic operations on p-graphs

Next, we examine operations on p-graphs. Of particular interest is the question of how these operations affect the induced interaction language: some will mutate the language; others will preserve it.

We give an example of a constructive operation which produces a new p-graph with a new interaction language, exploiting initial state nondeterminism.

Definition 3.6 (union of p-graphs)**.** *The* union *of two p-graphs U and W, each akin to the other, denoted by $U \uplus W$, is the p-graph constructed by including both sets of vertices, both sets of edges, and with initial states equal to $V_0(U) \cup V_0(W)$.*

The intuition is to form a graph that allows, via the nondeterministic selection of the start state, executions that belong to either U or W.

Theorem 3.2. *For p-graphs P and Q: $\mathcal{L}(P) \cup \mathcal{L}(Q) = \mathcal{L}(P \uplus Q)$*

Proof. Follows directly from Definitions 3.5 and 3.6. □

In general, the sets labeling two edges departing a vertex of a p-graph need not be disjoint, allowing multiple 'next' states to be indicated for the same event. It can be useful to distinguish p-graphs where this circumstance arises from those

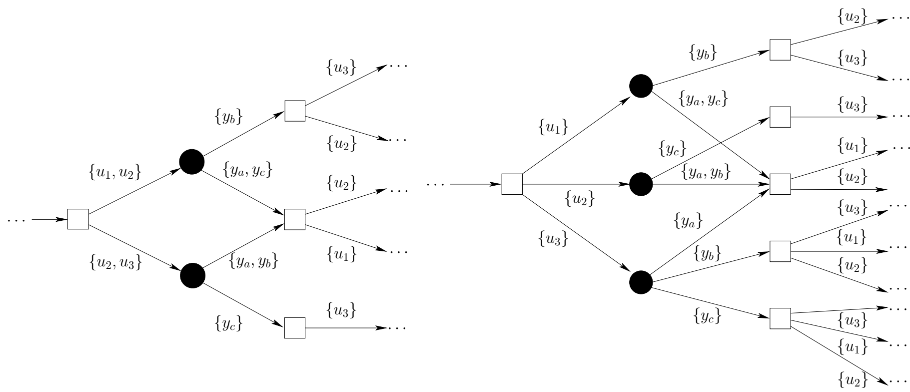

Figure 11: A fragment of a p-graph on the left is processed into the state-determined presentation on the right with the concomitant increase in number of states.

where it is absent. Our next definition formalizes this, while also highlighting that multiple p-graphs can induce the same interaction language.

Definition 3.7 (state-determined)**.** *A p-graph P is in a* state-determined presentation *if $|V_0(P)| = 1$ and from every action vertex $u \in V_u$, the edges $e^1_u, e^2_u, \ldots, e^n_u$ originating at u bear disjoint labels: $U(e^i_u) \cap U(e^j_u) = \varnothing, i \neq j$, and from every observation vertex $y \in V_y$, the edges $e^1_y, e^2_y, \ldots, e^m_y$ originating at y bear disjoint labels: $Y(e^i_y) \cap Y(e^j_y) = \varnothing, i \neq j$.*

The intuition is that in a p-graph in a state-determined presentation it is easy to determine whether an event sequence is an execution: one starts at the unique initial state and always has an unambiguous edge to follow. We note, however, that the p-graph with a state-determined presentation for some set of executions need not be unique.

Given any p-graph it is possible to construct a new p-graph that has the same set of executions on it, but which is in a state-determined presentation. Algorithm 2 shows how to convert an arbitrary p-graph into state-determined presentation. The basic idea is a forward search that performs a powerset construction on the input p-graph. We begin by constructing a single state to represent the "superposition" of all initial states, and push that onto a empty queue. While the queue has elements, remove a vertex and examine the edges leaving the set of vertices associated with it in the original input p-graph. The labels on those edges are refined by constructing a partition of the set spanned by the union of the labels in a way that the subsequent sets of states in the input p-graph is clear. Edges are formed with the refined sets connecting to their target vertices, constructing new ones as necessary, and placing these in the queue. This requires the use of Algorithm 1 to ensure that the edges in the new filter are drawn correctly. Figure 11 gives a simple example of the process for part of a p-graph. Though this shows a moderate increase in size, in

Algorithm 2: TOSTATEDETERMINEDPRESENTATION(G)

```
Input  : A p-graph G with vertex set V and starting set V_0.
Output: An equivalent state-determined p-graph G' with W and W_0, respectively.

Initialize W, W_0, as empty
Corresp[·] = ∅  // Construct an empty map to associate vertices between p-graphs
Add v'_0 to W and W_0  /* Construct an initial the vertex in G, preserving action-
                          originating or observation-originating type of elements in V_0   */
Corresp[v'_0] ← V_0  // Associate v'_0 with all v_0 ∈ V_0 in G
Initialize queue Q ← W_0
while Q not empty do
    s' ← Q.pop
    // Refine each label and determine which states each refinement maps to:
    L ← All outgoing edge labels of Corresp[s']
    L' ← REFINELABELS(L)  // cf. Algorithm 1
    Lab[·] = ∅  // Construct an empty map to associate refined labels to states
    for ℓ' ∈ L' do
        Determine the set of states reached by tracing event REPRESENTATIVE(ℓ')
        from each Corresp[s'], adding them to Lab[ ℓ' ]
    // Produce new states as needed:
    for s ∈ Lab[ℓ] for some ℓ do
        if t ∈ W, where t corresponds with s then
            Add s' -ℓ-> t   // Add transition on ℓ to G'
        else
            Create new state t corresponding to s  // Type should correspond too
            Q.push(t)    // Add to queue to be processed
            Add s' -ℓ-> t  // Add transition on ℓ to G'
return G'
```

general, following the procedure above may produce a p-graph as output that has an exponentially larger set of states than the input.

4 Label maps

We express modification of capabilities through maps that mutate the labels attached to the edges of a p-graph.

Definition 4.1 (action, observation, and label maps)**.** *An* action map *is a function* $h_u : \mathrm{U} \to 2^{\mathrm{U}'} \setminus \{\varnothing\}$ *mapping from an action space* U *to a non-empty set of actions in a different action space* U'*. Likewise, an* observation map *is a function* $h_y : \mathrm{Y} \to 2^{\mathrm{Y}'} \setminus \{\varnothing\}$ *mapping from an observation space* Y *to a non-empty set of observations in a different observation space* Y'*. A* label map *combines an action map* h_u *and a sensor map* h_y*:*

$$h(a) = \begin{cases} h_u(a) & \text{if } a \in \mathrm{U} \\ h_y(a) & \text{if } a \in \mathrm{Y} \end{cases}.$$

It is useful to extend this notion, so we do this immediately.

Definition 4.2 (label maps on sets and p-graphs)**.** *Given a label map* h*, its* extension to sets *is a function that applies the map to a set of labels:*

$$h(E) = \bigcup_{e \in \mathrm{E}} h(e).$$

The extension to p-graphs *is a function that mutates p-graphs by replacing each edge label* E *with* $h(\mathrm{E})$*. We will write* $h(P)$ *for application of* h *to p-graph* P*.*

Example 4.1 (label maps on intervals)**.** Representation of the action or observation spaces that are $\mathbb{R}$ via unions of intervals, as detailed in Example 3.7, lends itself to definition of label maps. To represent a label map on such an event space, we might, for example, take bounding polynomials $p_1(x)$ and $p_2(x)$, and define

$$h(x) = \{x' \mid p_1(x) \leq x' \leq p_2(x)\}.$$

Given a finite-union-of-intervals label $\ell \subset \mathbb{R}$, we can evaluate this kind of h by decomposing h into monotone sections, selecting the minimal and maximal values of p_1 and p_2 within that range, and computing the union of the results across all of the monotone sections. Figure 12 shows an example.

Label maps allow one to express weakening of capabilities as follows. If multiple elements in the domain of $h(\cdot)$ map to sets that are not disjoint, this expresses a conflation of two elements that formerly were distinct.

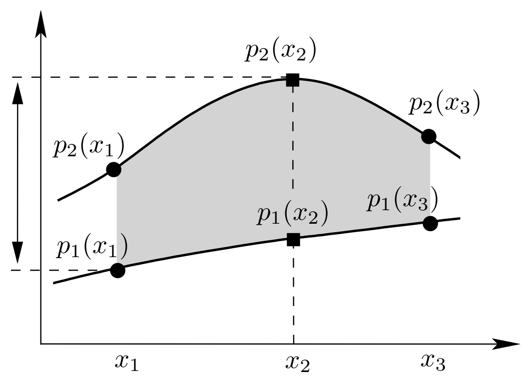

Figure 12: A label map from $\mathbb{R}$ to $2^{\mathbb{R}}$ may be described by functions p_1 and p_2 as lower and upper bounds, respectively. The marked vertical interval, spanning $p_1(x_1)$ to $p_2(x_2)$ illustrates the image of h across the monotone segment from x_1 to x_2. Values for other monotone segments would be computed similarly.

1. When they are observations, this directly models a sensor undergoing a reduction in fidelity since the sensor loses the ability to distinguish elements.

2. When they are actions, this models circumstances where uncertainty increases because a single action can now potentially produce multiple outcomes, and the precise outcome is unknown until after its execution.

Further, when the image of element E is a set with multiple constituents, this also expresses the fact that planning becomes more challenging.

3. For observations, it means that several observations may result from the same state and, as observations are non-deterministic, this increases the onus for joint-executions to maintain safety (for example, plans must account for more choices).

4. For actions, while there is a seemingly larger choice of actions, this increase does not represent an increase in control authority because several actions behave identically.

In both action and observation instances, the map may become detrimental when the outputs of $h(\mathrm{E})$ intersect for multiple Es and thus 'bleed' into each other. Broadly, one would expect that this is more likely when the output sets from $h(\cdot)$ are larger.

The next two sections address questions of how to reason about this sort of destructiveness for filtering and planning problems, respectively.

5 Destructiveness in filters

The management of uncertainty via integration of sensor readings has been a central theme in robotics research for decades. It is, thus, worth examining how p-graphs might be specialized to express structures suitable for such operations. Earlier, Example 2.1 (along with Example 3.2 presented thereafter) showed how both interaction languages, generally, and p-graphs, in particular, can describe estimation processes in the form of filters. The word filter is most familiar as a term used to describe practical estimation components of robots and their controller software. The filters treated in this section subsume those, representing a larger class, providing a broad, abstract theoretical treatment of algorithmic processes that aggregate information.

A p-graph over event space E is useful as a filter if the elements of U, the actions, are interpreted as merely publishing or emitting information. The idea is that filters influence an agent's representation of state, rather than altering the underlying state of the world itself. This act of interpretation gives p-graphs for filtering special significance to the agent that beholds them. (The idea that some *interpretation* must be provided to apply a p-graph to some circumstance is an important recurring theme in this work.) In this section, we will use the word output, rather than action, to emphasize the interpretation of a p-graph as a filter; occasionally we also just use the word filter to refer to such a p-graph.

When a p-graph that is being used as a filter has some edge e bearing a non-singleton set $\mathrm{U}(\mathrm{e})$, or when such a graph has an action edge with multiple out-edges, the resulting language includes choices for the information to be emitted. That choice is made arbitrarily, so whoever the consumer of the outputs might be (perhaps it is a controller or a planner), it must be able to operate with any in the set. Of course, if the p-graph is to be a faithful estimator this will constrain the p-graph's $\mathrm{U}(\mathrm{e})$ sets. One expects that a p-graph acting as a filter would produce information that is sound given the stream of inputs seen; such a filter can produce multiple outputs so long as the information from the filter need not be 'tight.' In the argot used to describe probabilistic filters, this corresponds, roughly, to fact that many possible filters may satisfy an unbiasedness criterion, though relatively few that satisfy only that constraint are actually good estimators.

The previous discussion notwithstanding, it is far more usual to imagine a unique output being produced in response to a particular history of events. We find it useful to be precise about the p-graphs which, for any action vertex, do not have any non-determinism on the output produced:

Definition 5.1. *A p-graph G is* single-outputting *if, for all $v \in \mathrm{V_u}$ reached by an execution, there is at most one edge e originating at v, and it bears a set* $\mathrm{U}(\mathrm{e})$ *with*

$|\mathrm{U}(\mathrm{e})| = 1$.

Though much previous discussion has emphasized the ability to use labels that describe infinite sets, the following establishes that this is not needed for dealing with the outputs of filters if they are single-outputting. There is still, however, significant value in use of infinite sets of observations in these cases.

Theorem 5.1 (finiteness of single outputting p-graphs)**.** *For any single-outputting p-graph G, there is a single-outputting p-graph G' that is equivalent in the sense that $\mathcal{L}(G) = \mathcal{L}(G')$, but where G' is defined over an event space with finite* U.

Proof. When G does not have a finite U, one constructs G' by copying G and simply restricting the set of actions for G' to be the union of the $\mathrm{U}(\mathrm{e})$ for edges originating at action vertices in the executions. This U is finite for there are finitely many edges and each $\mathrm{U}(\mathrm{e})$ contributes no more than one element. □

Algorithm 3: TOSINGLEOUTPUTTINGPRESENTATION(F)

Input : A p-graph F over an event space E with finite U.
Output: An equivalent filter F' that is single-outputting.

Copy $\mathrm{V_y}$ to F'
for every pair $v_o, v_{o'} \in \mathrm{V_y}$, where $v_o \xrightarrow{Y(v_0, v_a)} v_a \xrightarrow{U(v_a, v_{o'})} v_{o'}$ **do**
  for $i \in \mathrm{U}(v_a, v_{o'})$ **do**
    Add action vertex v_i to $\mathrm{V_u}$ for F'
    Add edge $v_o \xrightarrow{Y(v_0, v_a)} v_i$ to F'
    Add edge $v_i \xrightarrow{\{i\}} v_{o'}$ to F'
if F is an action-first p-graph **then**
  for every $v_1 \in \mathrm{V_0}$, where $v_1 \xrightarrow{U(v_1, v_o)} v_o$ **do**
    for $i \in \mathrm{U}(v_1, v_o)$ **do**
      Add action vertex v_i to $\mathrm{V_0}$ and $\mathrm{V_u}$ for F'
      Add edge $v_i \xrightarrow{\{i\}} v_o$ to F'
else
  Copy $\mathrm{V_0}$ from F to F'.
return F'

Moreover, there is a sort of converse that is true too. If the set U is finite, then having at most one singleton output set at each vertex, while a seemingly significant constraint, does not limit the expressivity of such filters. Every finite action-space p-graph has an equivalent single-outputting presentation.

One can convert an arbitrary finite action-space p-graph to an equivalent single-outputting p-graph by making duplicates of each action vertex, one for each output

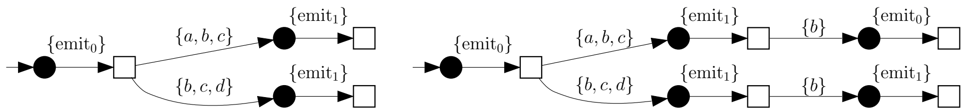

Figure 13: [left] A p-graph representation of a filter that, despite some labels not being disjoint, is a deterministic filter. [right] A p-graph that is not deterministic. To see this, note that the event sequence 'emit_0 b emit_0 b' has two distinct successors: 'emit_0' and 'emit_1'.

in the original, and making the sole transitions from those new vertices carry a single output. Algorithm 3 gives the procedure in detail.

There is a pattern worth noting here. Definition 3.7 details a particular structure that certain p-graphs possess, and Algorithm 2 then shows how any p-graph can transformed into a p-graph with that structure. For finite output sets, Definition 5.1 gives a particular structure that certain p-graphs possess, and Algorithm 3 shows how one can be transformed into a form with that structure. Both operations transform p-graphs whilst preserving the interaction languages they induce, which is why we call them presentations. There is more: Definitions 3.7 and 5.1 describe two distinct ways of presenting a p-graph, each of which places some restrictions on the kind of nondeterminism directly expressed in the p-graph, but they are, in a certain sense, duals of one another. Algorithm 3 may, in splitting edges with non-singleton labeled action sets, introduce some observation labels which overlap or multiple initial states. Algorithm 2 may, in eliminating overlapping labels on edges incident the same observation vertex (and also, in eliminating multiple initial states), produce a result which has action-states that have multiple edges departing it.

We claim that a certain class of p-graphs, however, can be represented in a way that is simultaneously single-outputting and state-determined.[3] We call these filters *deterministic*.

Definition 5.2. *A p-graph F is a* deterministic filter *if every observation-terminal sequence has at most one successor in $\mathcal{L}(F)$.*

In the sense we have defined here, the property of being a deterministic filter is a property of the p-graph's function, rather than a property of its representation; it is thus fitting that this notion can be expressed in terms of the induced interaction language. Note that the determinism does not mean that each observation-terminal sequence arrives at a unique state, but only that each observation sequence, if it yields any output, yields a single, determined output. (Figure 13 helps clarify this

[3]Proof of the claim that the class of p-graph defined in Definition 5.2 actually is this class appears in Theorem 5.4 below; first we elaborate on the version of determinism we define and, in particular, its relation to the classical notion.

distinction by illustrating the difference.) This second idea is closely tied to the usual sense of the word deterministic in classic automata theory: the notion that sequences of observations (note, solely, observations) drive the transitions of the automata, including dictating the precise state reached. This form is also important because the filters we described in Examples 2.1 and 3.2 are typically deterministic in this more traditional sense—no arbitrary choices need be made during their execution, observation inputs command the behavior. We define this class as follows:

Definition 5.3. *A single-outputting p-graph F is a* practicable filter *if the validity of every sequence $e_1 \cdots e_k \in \mathcal{L}(F)$ is the consequence of precisely one sequence of state transitions in F.*

They are named practicable because such filters are directly amenable to implementation. This, no doubt, goes some way to explaining why the filters that have appeared in the robotics literature are of this form.

Naturally, there is a connection between these practicable filters and the preceding notion of determinism, captured by these two lemmas:

Lemma 5.2. *Every practicable filter is a deterministic filter.*

Proof. Every observation-terminal sequence in the induced language traces a single trajectory through the filter's states and so arrives at precisely one state. It is single-outputting so there is at most one edge from that vertex and, hence, at most one successor. □

Lemma 5.3. *For every deterministic filter F there exists a practicable filter F' with $\mathcal{L}(F) = \mathcal{L}(F')$.*

Proof. One constructs F' by taking sets of vertices in F as vertices for F'; the start state of F' is V_0. One adds edges in F' by exploring each of the (finite) sequence prefixes that arrive at vertices in F, and labeling the transitions that are made. Tracing a prefix string on F may result in a branch: two edges leaving an observation vertex may have labels which overlap (though this cannot happen with action vertices). At such choice points both choices should be taken, which is why the states in F' are subsets of vertices of F. An edge in F' originating from action vertex labeled, say, $\{v_i, v_j\}$ (being associated with both action vertex v_i and action vertex v_j in F), only ever bears one label because the edge originating from v_i and from v_j in F must produce the same output, otherwise otherwise F would not be a deterministic filter. □

Having established that deterministic filters may be of practical importance because they can, ultimately, be turned into practicable filters, next we establish a relationship between the set of deterministic filters and the presentations (state-determined and single-outputting) introduced earlier.

Theorem 5.4. *Any state-determined p-graph is deterministic if and only if it is single-outputting.*

Proof. Forward direction: If F is a deterministic state-determined filter which is not single-outputting, there must be some sequence arriving at a vertex in $\mathrm{V_u}$ which, either has more than one departing edge, or it must possess an edge with a label containing at least two elements. If it has more than one departing edge, the labels cannot overlap because F is state-determined. But either multiple edges with distinct labels or an edge bearing a label with multiple elements imply multiple successors, contradicting the requirement that F be deterministic.
The other direction: Following the procedure described in Lemma 5.3 with a single-outputting and state-determined filter never leads to any choices. Therefore, only singleton subsets of 2^{V} are involved. As a result, any observation terminal sequence can yield at most one output. □

In the next section we use this result.

5.1 Ascertaining destructiveness of observation maps on filters

Since p-graphs are capable of representing filters, the next question is how they might enable a roboticist to evaluate tentative designs and to better understand solution space trade-offs. A class of interesting design-time questions arises when one considers how modifications to a given robot's capabilities alter the estimation efforts that the robot must undertake. In the specific context of filtering, consider the following illustrative examples of how maps might come into play when we apply them to observation sets.

Example 5.1. Your robot is equipped with a camera, and triplets of red–green–blue values within an array comprise Y. Now imagine that rose-tinted lenses are placed over the camera. Applied pixel-wise, $\mathrm{h_{rose}} : \langle r, g, b\rangle \mapsto \{\langle r, 0, 0\rangle\}$. Certain scenes that produce distinct inputs, $y_1 \neq y_2$, may now be indistinguishable under the transformation, as when two scenes differ only in elements of the spectrum filtered out by the lenses, and $\mathrm{h_{rose}}(y_1) = \mathrm{h_{rose}}(y_2)$.

Example 5.2. Observation maps need not only reduce the set. Suppose your sensor incurs cross-talk due to poor cable routing and cheap shielding. Where formerly a given circumstance would produce an observation y_i, this might be modeled with an observation map $y_i \mapsto \{y_i, y_i', y_i''\}$. It may be that $y' \in \mathrm{Y}$, or it might be some heretofore unseen class of signal. What we are interested in is whether this cross-talk is destructive or not. The answer to this depends on whether some other y_j exists where $y'_i \in \mathrm{h}(y_j)$. Even existence of such a y_j is insufficient, as y_i and y_j might occur in every pre-image together.

Next, we formalize the notion of a destructive observation map for filters.

Definition 5.4 (filter equivalence)**.** *Given two p-graphs for filtering, F over event space $\mathrm{E} = \mathrm{U} \cup \mathrm{Y}$ and F' over event space $\mathrm{E}' = \mathrm{U} \cup \mathrm{Y}'$, and an observation map $\mathrm{h_y} : \mathrm{Y} \to 2^{\mathrm{Y}'} \setminus \{\varnothing\}$ mapping from the observation space of F to sets of observations of F', we say that* F is equivalent to F' modulo $\mathrm{h_y}$*, denoted*

$$F \geqq F' \mod \mathrm{h_y},$$

if, for every observation-terminal sequence $e_1e_2e_3 \cdots e_m$ in $\mathcal{L}(F)$, we have that $\{u \mid e_1e_2e_3 \cdots e_m u \in \mathcal{L}(F)\} =$

$$\left\{ v \;\middle|\; f_1f_2f_3 \cdots f_m v \in \mathcal{L}(F'), \underset{i}{\forall} (e_i \in \mathrm{Y} \implies f_i \in \mathrm{h_y}(e_i)) \right\}.$$

Note that we eschew the traditional equivalence symbol '$\equiv$' for this relation because it is not symmetric: $F \geqq F' \mod \mathrm{h_y}$ does not necessarily imply $F' \geqq F \mod \mathrm{h_y}$.

To understand the preceding condition, observe that, on F, the sequence $s = e_1e_2 \cdots e_m$ produces an output that is an element of $\{u \mid su \in \mathcal{L}(F)\}$, the set on the left-hand side. Paying attention to only the observations that comprise s, which are every other element of the sequence, each of these e_i result in a set under $\mathrm{h_y}$. We consider all sequences that have observations such that every observation at position i in the sequence, which we denote f_i, is from the set $\mathrm{h_y}(e_i)$. Using all such sequences, we ask whether F' produces outputs that match F on s.

(Definition 5.4 has a simpler presentation if we assume that the filters involved have a finite action-space, in which case there exists a k, for which we are permitted to write the set on the left as $\{u_1, u_2, \ldots, u_k\}$. In what appears above we have not assumed that the action-space is finite, nor even denumerable.)

The intuition behind the definition is that if F', given observations mutated by $\mathrm{h_y}$, exhibits the same behavior that F exhibits when given those same observations, but unmutated, then any difference between F and F' is merely in the change in manifestation of the observations that was induced by $\mathrm{h_y}$ and the underlying structure is the same. In contrast, if the two filters can generate different outputs under these conditions, then there must be some other explanation for those differences. This suggestion motivates the idea of a nondestructive observation map.

Definition 5.5 (non-destructive)**.** *Given a p-graph F and an observation map* $\mathrm{h_y} : \mathrm{Y} \rightarrow 2^{\mathrm{Y}'} \setminus \{\varnothing\}$*, we say that* $\mathrm{h_y}$ *is* non-destructive *if*

$$F \geqq \mathrm{h_y}(F) \mod \mathrm{h_y}.$$

Informally, a nondestructive observation map is one that preserves enough structure that the filter still works after applying it, as long as the labels are updated accordingly. A destructive observation map is one that creates enough ambiguity (initially expressed in the resulting p-graph by states with out-edges whose labels overlap) that the correct outputs can no longer be determined solely by the observations.

Example 5.3. Suppose $\mathrm{h_y}$ is an injective map so that if $\mathrm{h_y}(y) = \mathrm{h_y}(z)$, then $y = z$. Because this kind of map does not introduce the possibility of conflating any two observations, it is clear that $\mathrm{h_y}$ is non-destructive. In the particular case of interval labels (recall Example 3.7), this implies that any sensor map that is a strictly-increasing or strictly-decreasing —including, for example, affine maps— is non-destructive. Contrapositively, we can also conclude that every destructive sensor map is non-injective.

Restricting ourselves to finite action-spaces, we can now pose the problem posed by the examples in Section 1 precisely and address it algorithmically. Given a p-graph F and an observation map $\mathrm{h_y} : \mathrm{Y} \rightarrow 2^{\mathrm{Y}'} \setminus \{\varnothing\}$, we wish to determine if $\mathrm{h_y}$ is non-destructive on F or not. We check explicitly whether $F \geqq \mathrm{h_y}(F) \mod \mathrm{h_y}$. Algorithm 4 shows how to perform this check. After converting to state-determined normal form, if necessary, the algorithm uses a forward search over pairs of states, one from each p-graph, that are reachable by some event sequence. For each such pair, if they are action vertices we verify that the outputs specified by the p-graphs are the same. For full generality, we show the algorithm for arbitrary pairs of p-graphs, not just for an F and its $\mathrm{h_y}(F)$.

5.1.1 Deterministic filters

Now suppose we have a deterministic filter F and an observation map $\mathrm{h_y}$, and wish to ascertain whether $\mathrm{h_y}$ is non-destructive on F—a special case that should be quite common, since those which are directly implementable, *viz.* practicable filters, are deterministic filters. For these filters we can use Algorithm 2 along with Theorem 5.4 to determine whether $\mathrm{h_y}$ is destructive. The intuition is to compute $\mathrm{h_y}(F)$, then convert that mapped filter to a state-determined presentation and check whether the result is also single-outputting. Checking whether a filter output from Algorithm 2 is single-outputting is especially straightforward because it only

Algorithm 4: EQUIVALENCEMODULOMAP$(F_1, F_2, \mathrm{h_y})$

Input : Two finite action-space p-graphs F_1 and F_2, and observation map $\mathrm{h_y} : \mathrm{Y} \to 2^{\mathrm{Y}'} \setminus \{\varnothing\}$.

Output: True iff $F_1 \geqq F_2 \mod \mathrm{h_y}$

if F_1 *and* F_2 *are not akin* **then**
  return False
Convert F_1 and F_2 to state determined presentation if needed.
/* Basic idea: conduct a forward search, computing the finite-set of observations needed to make all potential transitions along the way. */

Initialize queue $Q \leftarrow \mathrm{V}_0^{(\mathrm{F}_1)} \times \mathrm{V}_0^{(\mathrm{F}_2)}$
while Q *is not empty* **do**
  $(s_1, s_2) \leftarrow Q$.pop
  if s_1 *and* s_2 *are observation vertices* **then**
    $\mathrm{Y}_1 \leftarrow$ REFINELABELS(labels leaving s_1)
    $\mathrm{Y}_2 \leftarrow$ REFINELABELS(labels leaving s_2)
    $\mathrm{Y}_2' \leftarrow$ { pre-image of each element of Y_2 under $\mathrm{h_y}$}
    $\mathrm{L} \leftarrow$ REPRESENTATIVES$(\mathrm{Y}_1 \cup \mathrm{Y}_2')$
    /* L has a partition of the observation space which is just fine enough to exercise each filter. */
    for $\ell \in \mathrm{L}$ **do**
      $s_1' \leftarrow$ state that F_1 transitions to on ℓ
      $s_2' \leftarrow$ state that F_2 transitions to on $\mathrm{h_y}(\ell)$
      Q.push$((s_1', s_2'))$ // To be processed
  else
    // Both are action vertices
    $\mathrm{O}_1 \leftarrow$ UNION(labels leaving s_1)
    $\mathrm{O}_2 \leftarrow$ UNION(labels leaving s_2)
    // Equality of label sets computed using DIFFERENCE and EMPTY
    if $\mathrm{O}_1 \neq \mathrm{O}_2$ **then**
      return False // Output sets are not equal
    $\mathrm{U}_1 \leftarrow$ REFINELABELS(labels leaving s_1)
    $\mathrm{U}_2 \leftarrow$ REFINELABELS(labels leaving s_2)
    $\mathrm{L} \leftarrow$ REPRESENTATIVES$(\mathrm{U}_1 \cup \mathrm{U}_2)$
    for $\ell \in \mathrm{L}$ **do**
      $s_1' \leftarrow$ state that F_1 transitions to on ℓ
      $s_2' \leftarrow$ state that F_2 transitions to on ℓ
      Q.push$((s_1', s_2'))$ // To be processed
return True

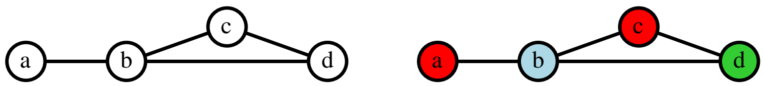

Figure 14: [left] An example instance of the 3-coloring problem. [right] A coloring of that graph using three colors.

outputs reachable vertices, thus, one simply checks that each action vertex has at most one singleton labeled edge departing it. Algorithm 5 gives the overall test for destructiveness, which is strikingly simple.

Algorithm 5: OBSERVATIONMAPDESTRUCTIVENESSTEST$(F, \mathrm{h_y})$

Input : A deterministic filter F and an observation map $\mathrm{h_y}$.
Output: TRUE iff $\mathrm{h_y}$ is non-destructive on F.

$G \leftarrow$ TOSTATEDETERMINEDPRESENTATION$(\mathrm{h_y}(F))$
return ISSINGLEOUTPUTTING(G)

5.2 Hardness

The preceding treatment of observation maps raises the question of why it is of interest to consider a variety of maps. Instead, why not simply find the observation map that is, in some sense, the 'most aggressive' nondestructive map for a given filter? In this section, we present a hardness result establishing that, unless $P = NP$, no efficient algorithm can find the nondestructive sensor map of minimal image size for a given filter, even approximately. Specifically, we consider the following decision problem:

Definition 5.6 (sensor minimization)**.** *The* sensor minimization decision problem *is: Given a p-graph F and integer n, return* TRUE *if there exists a set* K *and an observation map* $\mathrm{h_y} : \mathrm{Y} \to 2^{\mathrm{K}} \setminus \{\varnothing\}$*, nondestructive for F, with $|\mathrm{K}| \leq n$, and* FALSE *otherwise.*

Theorem 5.5. *The sensor minimization decision problem is NP-hard.*

Proof. Reduction from the graph 3-coloring problem GRAPH-3C, which is known to be NP-complete (Garey and Johnson, 1979). Given an instance G of GRAPH-3C, we construct an instance of the sensor minimization decision problem, building an action-first p-graph F as follows: Use one observation in Y for each vertex of G, so that $\mathrm{Y} = \mathrm{V}(G)$. For the set of actions, U, select $\{\mathrm{emit}_0, \mathrm{emit}_1, \mathrm{emit}_2\}$. Assign an

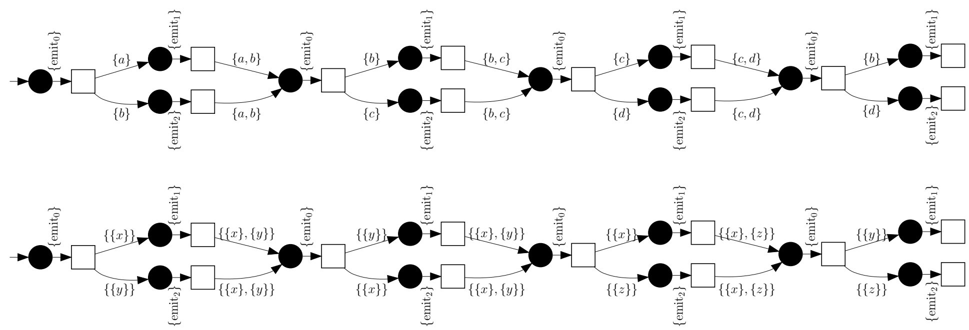

Figure 15: [top] The p-graph expressing a filter constructed from the graph coloring problem shown in Figure 14. [bottom] The result of applying an observation map under which $a \mapsto \{x\}$, $b \mapsto \{y\}$, $c \mapsto \{x\}$, and $d \mapsto \{z\}$. This mapped filter is equivalent to the original filter, modulo this map. Because this filter has a non-destructive map with image of size 3, the graph in Figure 14 can be colored with 3 colors.

arbitrary but fixed ordering to the edges $\mathrm{E}(G)$. For each edge $e \in \mathrm{E}(G)$ connecting nodes v and w: (1) insert three action vertices $i_e^\bullet$, $s_e^\bullet$, and $t_e^\bullet$, into $\mathrm{V_u}$; (2) insert three observation vertices $i_e^\square$, $s_e^\square$, and $t_e^\square$, into $\mathrm{V_y}$. Here, the names mnemonically indicate 'initial,' 'source node,' and 'target node.' Continue to build the p-graph G by adding an edge from $i_e^\bullet$ to $i_e^\square$ labeled with output $\{\mathrm{emit}_0\}$, adding an edge from $s_e^\bullet$ to $s_e^\square$ labeled with output $\{\mathrm{emit}_1\}$, and adding an edge from $t_e^\bullet$ to $t_e^\square$ labeled with output $\{\mathrm{emit}_2\}$. Add an edge labeled with the observation set $\{v\}$ from $i_e^\square$ to $s_e^\bullet$. Likewise, add an edge labeled with the observation set $\{w\}$ from $i_e^\square$ to $t_e^\bullet$. Unless e is the final edge in the ordering, let e' denote the next G-edge in the arbitrary ordering, and add edges to F labeled $\{v, w\}$ from $s_e^\square$ to $i_{e'}^\bullet$ and from $t_e^\square$ to $i_{e'}^\bullet$. For the first edge $e \in \mathrm{E}(G)$ in the ordering, designate $i_e^\bullet$ as the single initial node V_0. Select $n = 3$.

Figure 14 shows an example instance of GRAPH-3C, and Figure 15 shows the corresponding filter. The construction takes time linear in the size of G. It remains to show that G is 3-colorable if and only if F has a nondestructive observation map $\mathrm{h_y} : \mathrm{Y} \to 2^{\mathrm{K}}$ with $|\mathrm{K}| \le 3$.

Assume that G is 3-colorable. Let $\mathrm{c} : V(G) \to \{0, 1, 2\}$ be a 3-coloring of G. Since $\mathrm{Y} = \mathrm{V}(G)$, we construct an observation map for F using c as follows. We let $\mathrm{K} = \{0, 1, 2\}$ and map only to singleton subsets: $\mathrm{h_y}(y) \mapsto \{c(y)\}$.

Note that $\mathrm{h_y}(F)$ is state-determined; the only states with multiple out-edges are the $i_e^\square$ states, and since c is a coloring of G, the two observations labeling these edges in F must map to different sets under $\mathrm{h_y}$. Therefore, $\mathrm{h_y}$ is a nondestructive sensor map for F, and $|\mathrm{K}| = 3$.

For the other direction, assume F has a nondestructive observation map $\mathrm{h_y}$:

$\mathrm{Y} \to 2^{\mathrm{K}} \setminus \{\varnothing\}$ with $|\mathrm{K}| = 3$. Then, there must also exist a nondestructive map $\mathrm{h}^{\mathrm{s}}_{\mathrm{y}}$ mapping to singletons from K, i.e., $\{\{k\} \mid k \in \mathrm{K}\}$ since mapping to sets with more than one element only loses information. From h_{y} we can construct an $\mathrm{h}^{\mathrm{s}}_{\mathrm{y}}$ by making some arbitrary choice from items in the image set.

We argue that this $\mathrm{h}^{\mathrm{s}}_{\mathrm{y}}$ forms a valid 3-coloring of G. Suppose, to the contrary, that $\mathrm{h}^{\mathrm{s}}_{\mathrm{y}}$ is not a valid 3-coloring of G. Then there must exist some edge $e \in \mathrm{E}(G)$, connecting two nodes v and w, such that $\mathrm{h}^{\mathrm{s}}_{\mathrm{y}}(v) = \mathrm{h}^{\mathrm{s}}_{\mathrm{y}}(w)$. But in that case, in $h^s_y(F)$, and hence $h_y(F)$, from the node $i^{\square}_e$ there are two out-edges, both intersecting labels, leading to differently-colored states, namely $s^{\bullet}_e$ and $t^{\bullet}_e$. But $s^{\bullet}_e$ results in $\{\mathrm{emit}_1\}$, while $t^{\bullet}_e$ does an $\{\mathrm{emit}_2\}$. In contrast, the original F is state-determined. Therefore $\mathrm{h}^{\mathrm{s}}_{\mathrm{y}}$ is destructive of F, a contradiction.

Finally, since GRAPH-3C is polynomial-time reducible to sensor minimization, we conclude that it is NP-hard. □

Note, *a fortiori*, that the proof of Theorem 5.5 does not depend any essential way on the specific number 3. In fact the chromatic number of the graph coloring instance and the image size of the smallest nondestructive observation map for the corresponding p-graph filter are always equal. Combined with known results on the inapproximability of chromatic numbers (Zuckerman, 2007), this leads directly to the following stronger result.

Corollary 5.5.1. *The optimization problem of finding, for a given filter, the nondestructive sensor map with the smallest image size, is NP-hard to approximate to within $n^{1-\epsilon}$.*

Proof. Let $\epsilon > 0$. Suppose that there exists a polynomial time approximation algorithm A to solve sensor minimization with approximation ratio $n^{1-\epsilon}$. Let B denote an approximation algorithm for graph coloring that works as follows.

1. Given an instance G of graph coloring problem, form the filter F as described in the proof of Theorem 5.5.

2. Use algorithm A to apply map h_{y} on F.

3. Find a coloring of G from the applied map $\mathrm{h}_{\mathrm{y}}(F)$.

We now argue that B has approximation ratio $n^{1-\epsilon}$.

Let $B(G)$ denote the number of colors used by algorithm B to color G, and, likewise, let $A(\mathrm{h}_{\mathrm{y}}(F))$ denote the image size of filter produced by algorithm A from input filter F and with map h_{y} applied. Let $OPT(\mathrm{h}_{\mathrm{y}}(F))$ and $OPT(G)$ represent the smallest image size of applying h_{y} on filter F and minimum number of colors for coloring G, respectively. According to the assumption, we have

$A(F) \leqslant n^{1-\epsilon} OPT(\mathrm{h_y}(F))$.
Thus, the above construction would be such an approximation algorithm B for graph coloring problem. So, we have $OPT(\mathrm{h_y}(F)) = OPT(G)$. Then, for sufficiently large n,

$$\begin{aligned} B(G) &= A(\mathrm{h_y}(F(G))) && (1) \\ &\leqslant n^{1-\epsilon} OPT(\mathrm{h_y}(F)) && (2) \\ &\leqslant n^{1-\epsilon} OPT(G). && (3) \end{aligned}$$

Therefore, B is an approximation algorithm for graph coloring with approximation ratio $n^{1-\epsilon}$, which contradicts Zuckerman (2007).

□

5.3 Case study: Minimizing the iRobot Create

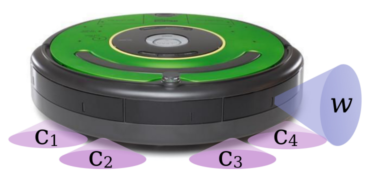

Figure 16: An iRobot Create is equipped with a collection of simple sensors including four cliff sensors and a wall sensor, each uses IR to measure distance. As a simple example, we consider a filter which maps sensor readings into motor commands on a robot tasked with following a wall on its left, while avoiding negative obstacles. (A suitable environment is shown in Figure 3.)

The following simple scenario, of the sort that the authors have often assigned in introductory robotics courses, illustrates the utility of the machinery developed in this paper. Here, we report transformations computed by our Python implementation of the algorithms described above. Revisiting the scenario in Figure 3, we wish to have an iRobot Create vacuum cleaning robot follow walls (on its port side) while avoiding negative obstacles. The five range sensors on the robot provide sufficient information to carry out this basic task. (See Figure 16 for elaboration of the sensing details.) We approach this problem by constructing a p-graph whose outputs map directly to actions for the robot, and then we are able to analyze the effect of observation maps on this p-graph (as a filter) with our implementation of the algorithms described in the earlier sections of the paper.

First we describe the set of observations for idealized versions of the robot's sensors. Each of $\{w, c_1, c_2, c_3, c_4\}$ is fundamentally a device that measures distance, so it is useful to model each output with a real number that represents

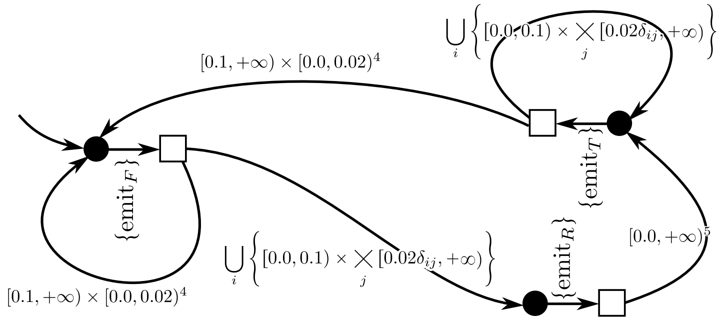

Figure 17: A visual representation of the filter (IDEAL) that solves navigation problem for the Create where edge labels are subsets of $\mathbb{R}^5$, and the values in each vertex are velocities that the robot executes for some small finite time. The action emit_F generates a 'Forward-with-slight-left-bias' motion with $\dot{x} = 0.2, \dot{\theta} = 0.1$, the emit_T generates a 'Turn' motion via $\dot{x} = 0, \dot{\theta} = -0.2$, and the emit_R generates a 'Reverse' motion via $\dot{x} = -0.1, \dot{\theta} = 0$. (The δ_{ij} is the Kronecker delta, $i, j \in \{1, 2, 3, 4\}$.)

the range reading; naturally, the product of these five sensors gives a label space with $\mathbb{R}^5$. Each state in the filter produces an output that is interpreted as velocity commands—linear as $\dot{x}$ and angular as $\dot{\theta}$. The filter is shown pictorially in Figure 17.

From this a series of filters is constructed via transformations that coarsen the label space. Observation map $\mathrm{h_y}$ clips to a maximum ranges supported by the sensors. It is applied to the IDEAL filter, giving a filter CREATE (SIGNALS), whose labels are based on the data that can be read from the physical sensors through the software interface (see iRobot Corp. (2015)). Observation map $\mathrm{f_y}$, transforms CREATE (SIGNALS) into CREATE (SYMBOLS) representing a second level of abstraction —in this case, a quantization based on thresholding— available through the robot's hardware interface. Map $\mathrm{g_y}$ further reduces the set of labels, while the final map we define, $\mathrm{k_y}$, is destructive. Table 1 collects this information. The rows in the table also summarize the relationships visually. Starting from IDEAL one produces the others via composition of the sensor maps, for example, COMBINED SENSOR results from applying map the $\mathrm{g_y} \circ \mathrm{f_y} \circ \mathrm{h_y}$. Only the final map to a sensorless model is destructive. The conclusion we reach from this is that, for this filter, neither the specific distance measurements nor the individual identities of the sensors themselves are necessary. A robot designed exclusively for this task could, therefore, likely be designed to be simpler than a Create.

Name	Observation Space	Notes
IDEAL	$\mathbb{R} \times \mathbb{R} \times \mathbb{R} \times \mathbb{R} \times \mathbb{R}$	
$\downarrow$ $h_y = \textbf{clipToRange}(\cdot)$		
CREATE (SIGNALS)	$[0, 1023] \times [0, 4095]^4$	*cf.* iRobot Corp. (2015, pg. 27).
$\downarrow$ $f_y = \textbf{threshold}_T(\cdot)$	($T = 10$ and 20 for w and c_i respectively.)	
CREATE (SYMBOLS)	$\{0, 1\} \times \{0, 1\}^4$	*cf.* iRobot Corp. (2015, pgs. 22–23).
$\downarrow$ $g_y = \textbf{min}(\cdot)$		
COMBINED SENSOR	$\{0, 1\}$	
$\downarrow$ $k_y = 0$	(Constant map)	
SENSORLESS	$\{0\}$	(Destructive)

Table 1: A hierarchy of filters for the iRobot Create.

6 Destructiveness in planning problems

6.1 Plans and Planning Problems

The final section of Tovar et al. (2014), a substantial and recent paper on the topic of combinatorial filters, concludes with the following:

> *"Since the methods so far provide only inference, how can their output be used to design motion plans? In other words, how can the output be used as a filter that provides feedback for controlling how the bodies move to achieve some task?"*

Next, we make some progress in that direction by using p-graphs to model planning problems and plans. The example of the iRobot Create in Section 5.3 had a direct correspondence between filter outputs and actions that the robot executed. In general, the sequences of actions that a robot performs will depend on the task it is performing, but in the previous section there was no direct representation of tasks. Thus, though p-graphs have been used up to this point to encode state space structure, more information must be provided to talk meaningfully about plans and planning problems.

Definition 6.1 (planning problem)**.** *A* planning problem *is a p-graph G equipped with a* goal region $V_{\text{goal}} \subseteq V(G)$.

The idea is that for a pair that make up the planning problem, the p-graph describes the setting and form in which decisions must be made, while the V_{goal} characterizes what must be achieved. Recall that, because a p-graph may have multiple initial states, this definition can encompass planning problems in which the system is known to start from one of possibly many starting states.

Definition 6.2 (plan)**.** *A* plan *is a p-graph P equipped with a* termination region $V_{\text{term}} \subseteq V(P)$.

The intuition is that the out-edges of each action state of the plan show one or more actions that may be taken from that point—if there is more than one such action, the robot selects one nondeterministically—and the out-edges of each observation state show how the robot should respond to the observations received from the environment. If the robot reaches a state in its termination region, it may decide to terminate there and declare success, or it may decide to continue on normally. This, then, gives an interpretation of p-graphs as plans. We can now establish the core relationship between planning problems and plans.

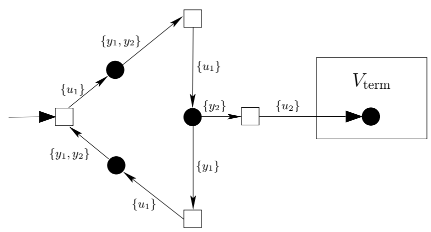

Figure 18: A plan that directs the robot of Figure 8 to its charging station, along a hyperkinetic (that is, exhibiting more motion than is strictly necessary) path.

Definition 6.3 (solves)**.** *A plan* $(P, P_{\rm term})$ solves *the planning problem* $(W, V_{\rm goal})$ *if* P *is finite and safe on* W*, and every joint-execution* $e_1 \cdots e_k$ *of* P *on* W *either reaches a vertex in* $P_{\rm term}$*, or is a prefix of some joint-execution that reaches* $P_{\rm term}$ *and, moreover, all the* $e_1 \cdots e_k$ *that reach a vertex* $v \in V(P)$ *with* $v \in P_{\rm term}$*, always reach vertices* $w \in V(W)$ *with* $w \in V_{\rm goal}$*.*

The solution concept here, with its stipulation of finiteness reminiscent of notions of computability, is concerned only with processes that terminate in some bounded time. (We defer questions about extensions to other prevalent concepts — such as infinite horizons, models of rewards, and the like — to future work.)

Example 6.1 (Charging around and in the pentagonal world)**.** We can construct a planning problem from the p-graph of Figure 8, along with a goal region consisting of only the fully-charged state reached by action u_2. Figure 18 shows a plan that solves this problem. However, that plan, a cycle of three actions, is a bit surprising since it will take the robot along three full laps around its environment before terminating. The existence of such bizarre plans motivates our consideration of homomorphic plans, which behave rather more sensibly, in Section 6.2.

Both plans and planning problems are pairs consisting of a p-graph and a set of states. In each case, the p-graph can be converted into a state-determined presentation using Algorithm 2 and doing so preserves the interaction language. But the semantics for both structures, plans and planning problems, is tied together through the definition of 'solves' (Definition 6.3). That definition has two parts. The first concerns finiteness and safety, properties of joint-executions only, and is consequently unaffected by transformations that preserve the interaction language. The second depends on vertices and their relationship to the associated sets. Thus, forming something analogous to a state-determined presentation must require some alteration of the second element of the pair, i.e., the set of states.

Definition 6.4 (state-determined planning problems)**.** *The state-determined presentation of planning problem* $(W, V_{\rm goal})$ *is* $(W', V'_{\rm goal})$ *where* W' *is the state-determined presentation of p-graph* W*, and* $V'_{\rm goal}$ *is the subset of* $V(W')$ *where*

$v \in V(W')$ is included in V'_{goal} only if all the vertices in $V(W)$ that correspond with v are in V_{goal}.

Definition 6.5 (state-determined plans)**.** *The state-determined presentation of problem (P, V_{term}) is (P', V'_{term}) where P' is the state-determined presentation of p-graph P, and V'_{term} is the subset of $V(P')$ where $v \in V(P')$ is included in V'_{term} only if there exists a vertex in $V(P)$ corresponding with v that is in V_{term}.*

The previous two definitions differ only in terms of the quantifier involved in their conditions on associated vertices. These mirror the 'reach a vertex' and 'always reach vertices' in Definition 6.3. Practically, the conditions can be computed easily by via the Corresp$[\cdot]$ map used in Algorithm 2.

Lemma 6.1 (state-determined presentations preserve solubility)**.** *If (P, V_{term}) is a plan, and (W, V_{goal}) a planning problem, with their state-determined presentations being (P', V'_{term}) and (W', V'_{goal}) respectively, then the following are equivalent:*

1. *(P, V_{term}) solves (W, V_{goal})*
2. *(P', V'_{term}) solves (W, V_{goal})*
3. *(P, V_{term}) solves (W', V'_{goal})*
4. *(P', V'_{term}) solves (W', V'_{goal})*

Proof. The correspondence between vertices in the original p-graph and the state-determined presentations allows the joint-executions in one case to be traced in the other. Under the correspondence, one must check the requirements for being a solution do in fact hold. But the logic necessary in updating the goal and termination sets in Definitions 6.4 and 6.5 correspond to the solution requirements (an event sequence reaching *a* vertex in V_{term} will reach a set of vertices *all* of which are in V_{goal}), so they do hold. □

Now, given a plan (P, P_{term}) and a planning problem (W, V_{goal}), we can decide whether (P, P_{term}) solves (W, V_{goal}) in a relatively straightforward way. First, we convert both into state-determined presentations, as just described.Then, the algorithm conducts a forward search using a queue of ordered pairs (v, w), in which $v \in V(P)$ and $w \in V(W)$, beginning from the (unique, due to Definition 3.7) start states of each. For each state pair (v, w) reached by the search, we can test each of the properties required by Definition 6.3:

- If P and W are not akin, return false.
- If (v, w) has been visited by the search before, then we have detected the possibility of returning to the same situation multiple times in a single execution. This indicates that P is not finite on W. Return false.
- If v and w fail the conditions of Definition 2.8 (that is, if v is missing an observation that appears in w, or w omits an action that appears in v) then P is not safe on W. Return false.
- If v is a sink state not in P_{term}, or w is a sink state not in V_{goal}, then we have detected an execution that does not achieve the goal. (A vertex is a sink if it has no departing edges.) Return false.
- If $v \in P_{\text{term}}$ and $w \notin V_{\text{goal}}$, then the plan might terminate outside the goal region. Return false.

If none of these conditions hold, then we continue the forward search, adding to the queue each state pair (v', w') reached by a single event from (v, w). Finally, if the queue is exhausted, then—knowing that no other state pair can be reached by any execution—we can correctly conclude that (P, P_{term}) does solve (W, V_{goal}).

It may perhaps be surprising that both planning problems and plans are defined by giving a p-graph, along with a set of states at which executions should end. We view this symmetry as a feature, rather than a bug, in the sense that it clearly illuminates the duality between the robot and the environment with which it interacts. As alluded to in Section 2, observations can be viewed as merely "actions taken by nature" and vice versa. At an extreme, the planning problem and the plan may be identical:

Lemma 6.2 (self-solving plans)**.** *If P is a p-graph which is acyclic and the set of its sink nodes is V_{sink}, then the plan (P, V_{sink}) solves the planning problem (P, V_{sink}).*

Proof. The plan is obviously finite and safe on itself. Because the set of joint-executions is simply the set of executions, the result follows from the fact that every execution on P either reaches an element of V_{sink}, or is the prefix of one that does. □

We have described, in Definitions 3.6 and 3.7, operations to construct new p-graphs out of old ones. We can extend these in natural ways to apply to plans.[4]

[4]. . . and—via the symmetry between Definitions 6.1 and 6.2—in the same stroke, to planning problems, though in this paper we'll use these operations only on plans.

Definition 6.6 ($\cup$-product of plans)**.** *The* $\cup$-product *of two plans* (P, P_{term}) *and* (Q, Q_{term})*, with* P *and* Q *akin, is a plan* $(P \uplus Q, P_{\text{term}} \cup Q_{\text{term}}$.

Theorem 6.3 (state-determined $\cup$-products)**.** *Given plans* (P, P_{term}) *and* (Q, Q_{term})*, with* P *and* Q *akin, construct a new plan whose p-graph, denoted* R*, is the expansion of* $P \uplus Q$ *into a state-determined presentation. Recall that the expansion means that every state* $s \in V(R)$ *corresponds to sets* $P_s \subseteq V(P)$ *and* $Q_s \subseteq V(Q)$ *of states in the original p-graphs (either set can be empty, but never both). Define a termination region* R_{term} *as follows:*

$$R_{\text{term}} \coloneqq \{s \in V(R) \mid (P_s \neq \varnothing \wedge P_s \setminus P_{\text{term}} = \varnothing) \vee (Q_s \neq \varnothing \wedge Q_s \setminus Q_{\text{term}} = \varnothing)\}\,.$$

Then (R, R_{term}) *is equivalent to* $(P \uplus Q, P_{\text{term}} \cup Q_{\text{term}})$ *in the sense of having identical sets of executions. Moreover, any planning problem solved by the former is also solved by the latter.*

Proof. This follows directly from the executions underlying the state-determined expansion, and the definition of the $\cup$-product. $\square$

This result illustrates how the state-determined expansion is useful — it permits a construction that captures the desired behavioral properties and, by working from a standardized presentation, can do this directly by examining states rather than posing questions quantified over the set of executions.

6.2 Homomorphic solutions

The following are a subclass of all solutions to a planning problem.

Definition 6.7 (homomorphic solution)**.** *For a plan* (P, V_{term}) *that solves planning problem* (W, V_{goal})*, consider the relation* $R \subseteq V(P) \times V(W)$*, in which* $(v, w) \in R$ *if and only if there exists a joint-execution on* P *and* W *that can end at* v *in* P *and in* w *in* W*. A plan for which this relation is a function is called a* homomorphic solution.

The name for this class of solutions comes via analogy to the homomorphisms — that is, structure-preserving maps— which arise in algebra. In this context, a homomorphic solution is one for which each state in the plan corresponds to exactly one state in the planning problem.

Example 6.2. Recall Example 6.1, which shows a cyclic solution that involves tracing around the cyclic planning problem multiple times (until the least common multiple of their cycle lengths is found, in this case a series of 30 states in each graph). This plan is *not* a homomorphic solution because each plan state corresponds to multiple problem states. However, a simpler plan, depicted in Figure 19, can be formed in which each plan state maps to only one problem state. This solution is therefore a homomorphic one.

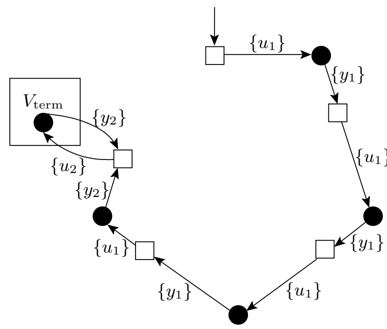

Figure 19: An alternative, more direct plan that solves the problem of navigating Figure 8's robot to its charger. This plan is a homomorphic solution.

The preceding example is a particular instance of a more general pattern.

Theorem 6.4. *If there exists a plan to solve a state-determined planning problem, then there exists a homomorphic solution.*

Proof. Suppose (P, P_{term}) is a solution to (W, V_{goal}). If every joint-execution arriving at v in P arrives at the same w in W, then $(v, w) \in R$ is a function, so (P, P_{term}) is a homomorphic solution. Thus, consider the cases for which there are elements $(v, w) \in R$ and $(v, y) \in R$, with $w \neq y$. Let $R_{\text{last}} \subset R$ be the relation where $(v_p, v_w) \in R_{\text{last}}$ iff there is a joint-execution $e_1 \cdots e_k$ arriving at v_p on P and v_w on W, and there are no joint-executions which extend the execution (e.g., $e_1 \cdots e_k \cdots e_m$, $m > k$) that arrive at v_w again. Then construct a new plan (Q, Q_{term}) with $V(Q) = V(W)$ and $V_0(Q) = V_0(W)$. For all edges departing $v \in P$ associated with $w \in Q$ where $(v, w) \in R_{\text{last}}$, we collect the label sets by unioning them to form V_e. Then edges departing w are included in Q by carrying over edges from W, intersecting V_e with all the labels of edges departing w, and dropping those for which the result is empty. Finally, an element w is included in Q_{term} if there is a $v \in P_{\text{term}}$ with $(v, w) \in R_{\text{last}}$. Then (Q, Q_{term}) is a solution to (W, V_{goal}) because, though (P, P_{term}) and (Q, Q_{term}) have different sets of executions, every execution on P that reaches P_{term} is transformed into another on Q reaching Q_{term} (and V_{goal}). Moreover, this ensures that the relation from Definition 6.7 is a bijection, so that (Q, Q_{term}) is a homomorphic solution to (W, V_{goal}). $\square$

6.3 Destructive or not?

If a label map can express a change in a p-graph, the question is whether this change matters. One can pose this question meaningfully for planning problems as the added ingredients provide semantics that yield the notion of solubility.

Definition 6.8 (destructive and non-destructive on plans)**.** *A label map h is* destructive *on a set of solutions S to planning problem (G, V_{goal}) if, for every plan $(P, V_{\text{term}}) \in S$, $(h(P), V_{\text{term}})$ cannot solve $(h(G), V_{\text{goal}})$. We say that h is* non-destructive *on S if for every $(P, V_{\text{term}}) \in S$, the plan $(h(P), V_{\text{term}})$ does solve $(h(G), V_{\text{goal}})$.*

Intuitively, destructiveness requires that the label map break all existing solutions; non-destructiveness requires that the label map break none of them.

Example 6.3 (single plans)**.** If $S = \{s\}$ is a singleton set, then we can determine whether h is destructive on S by applying the label map h —recall Definition 4.1— to compute $h(s)$ and $h(G)$, and then testing whether $h(s)$ solves $h(G)$ —recall the algorithm described in Section 6. If $h(s)$ solves $h(G)$, then h is nondestructive on S; otherwise, h is destructive on S. In this singleton case, we say simply that h is (non-)destructive on s.

Definition 6.8 depends on a selection of some class of solutions. Of particular interest is the maximal case, in which every solution is part of the class.

Definition 6.9 (strongly destructive and strongly non-destructive)**.** *A label map h is* strongly (non-)destructive *on a planning problem (G, V_{goal}) if it is (non-)destructive on the set of all solutions to (G, V_{goal}).*

Note that, while strong destructiveness may be decided by attempting to generate a plan for $h(G)$ (perhaps by backchaining from V_{goal}), strong non-destructiveness may be quite difficult to verify in general, if only due to the sheer variety of extant solutions. (Recall Example 6.1, which solves its problem in an unexpected way.) The next results, while not sufficient in general to decide whether a map is strongly non-destructive, do perhaps shed some light on how that might be accomplished.

Lemma 6.5 (label maps preserve safety)**.** *If P is safe on G, then for any label map h, $h(P)$ is safe on $h(G)$.*

Proof. Consider each pair of states (v, w), with $v \in V(P)$ and $w \in V(G)$ reached by some joint-execution on P and G. Suppose for simplicity that v is an action state. (The observation case is similar.) Let E_1 denote the union of all labels for edges outgoing from v, and likewise E_2 for labels of edges outgoing from w. Since P is safe on G, we have $E_1 \subseteq E_2$. Then, in $h(P)$ and $h(G)$, observe that

$$h(E_1) = \bigcup_{e \in E_1} h(e) \subseteq \bigcup_{e \in E_2} h(e) = h(E_2),$$

and conclude that $h(P)$ is safe on $h(G)$. □

Lemma 6.6 (label maps never introduce homomorphism)**.** *If* (P, P_{term}) *is a non-homomorphic solution to* (G, V_{goal}) *then no label map* h *results in* $(h(P), P_{\text{term}})$ *being a homomorphic solution to* $(h(G), V_{\text{goal}})$.

Proof. Since (P, P_{term}) is a non-homomorphic solution to (G, V_{goal}), there exist two joint-executions $e_1 \cdots e_k$ and $e'_1 \cdots e'_m$ on P and G such that both arrive at $v \in V(P)$ in P, but on G, the former arrives at $w \in V(G)$ and the latter arrives at $w' \in V(G)$ with $w \neq w'$. Now, given any $h(\cdot)$, pick any particular sequence $(h_1 \in h(e_1)) \cdots (h_k \in h(e_k))$, and $(h'_1 \in h(e'_1)) \cdots (h'_m \in h(e'_m))$, making choices arbitrarily. These are joint-executions on $h(P)$ and $h(G)$. Application of the label map means there is a way of tracing both $(h_1 \in h(e_1)) \cdots (h_k \in h(e_k))$ and $(h'_1 \in h(e'_1)) \cdots (h'_m \in h(e'_m))$ on $h(P)$ to arrive at v, while there is a way of tracing the former on $h(G)$ to arrive at w, and the latter at w'. So $(h(P), P_{\text{term}})$ cannot be a homomorphic solution to $(h(G), V_{\text{goal}})$. □

Theorem 6.7 (extensive destructiveness)**.** *For a state-determined planning problem* (G, V_{goal})*, let* $\mathcal{H}$ *denote the set of homomorphic solutions that problem. Then any label map that is destructive on* $\mathcal{H}$ *is strongly destructive.*

Proof. Since h is destructive on $\mathcal{H}$, we know that $(h(G), V_{\text{goal}})$ can only have homomorphic solutions if some formerly non-homomorphic solution can become a homomorphic one under h, but Lemma 6.6 precludes that eventuality. This implies, via Theorem 6.4, that *no* plan solves $h(G)$. Therefore h is strongly destructive on (G, V_{goal}). □

The interesting thing here is that Theorem 6.7 shows that the class of homomorphic solutions play a special role in the space of all plans: By examining the behavior of h on $\mathcal{H}$, we can gain some insight into its behavior on the space of all plans. Informally, $\mathcal{H}$ seems to function as a 'kernel' of the space of all plans.

7 Related work

In earlier sections of the paper we have interspersed precise connections to specific prior work. This section supplements those links by taking a wider view; the purpose is not merely coverage, but rather broader context.

This work builds most directly on, and is strongly influenced by, the combinatorial filtering perspective, with its use of simple, discrete objects that generalize beyond the methods used in traditional estimation theory, which has a strong reliance on probabilistic models. A gap still remains between the theory of discrete combinatorial filters and the probabilistic, typically recursive Bayes formulations, employed most often in practice on robots today. Both types have a long history.

The probabilistic filters go back to Kalman (1960), having found use in several important problems in mobile robotics, including estimation of robot pose and map information (Dissanayake et al., 2001; Smith et al., 1990). This class of filters is well-known within the community, with a vast surge of interest catalyzed by the publication of the book by Thrun et al. (2005). The discrete filters we focus on in this paper have their roots in the minimalist manipulation work of Erdmann and Mason (1988) and Goldberg (1993). They were formalized more generally by LaValle (2006), though this paper evolves those models in a new direction.

Discrete filters and their related sorts of representations are recherché rather than simply obscure: They have been employed in the form of combinatorial filters to successfully solve a wide a range of useful tasks; recent examples include target tracking (Yu and LaValle, 2012), mobile robot navigation (Lopez-Padilla et al., 2012; Tovar et al., 2007), and manipulation (Kristek and Shell, 2012). Both LaValle (2012), which provides a tutorial introduction and overview to the approach, and the substantial paper on the topic Tovar et al. (2014), recognize that more work is needed to extend the theory. There are two directions which have demanded attention. The first, which the authors of both of the preceding papers identify, is that, thus far, the approach provides only for inference and more work is needed in order to express aspects of feedback-aware control for achieving tasks. Section 6 has begun to address this gap.

The second direction is born of the observation that all the combinatorial filters in the existing work deal with extremely simple sensors. How might combinatorial filters and other discrete models scale up to larger problems? One may quite rightly criticize such filters on the basis of their size or expression complexity. An important contribution of the present paper is increasing in the complexity of sensors that may be treated by discrete filters without necessitating an enormous growth of filter size. Previously, when sets of observations were treated, they required duplication (usually of an edge in a graph structure), causing substantial blow-up of the model. The form of filters we examine does not assume that the set of possible observations is finite. And we have described results for filters with infinite (though finitely described) subsets of $\mathbb{R}$ as labels. This idea was inspired by Veanes et al. (2012), who developed symbolic finite transducers that are concise and expressive for processing strings over large alphabets. It is also worth nothing that a set of techniques have been developed, along quite separate lines, to reduce or simplify the representation of information within such filters (O'Kane, 2011; O'Kane and Shell, 2017; Song and O'Kane, 2012).

Both directions have demanded generalization in slightly different ways. We believe that one of the most useful aspects of the formalism arising from this generalization has been the notion of label maps. These functions allow one to degrade models, starting (as we did in the iRobot Create case study) with physically un-

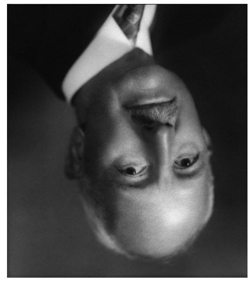

Figure 20: George Stratton, as he would have appeared in the first few minutes of wearing the inversion glasses he pioneered. "If a subject is made to wear glasses which correct the retinal images, the whole landscape at first appears unreal and upside down; on the second day of the experiment normal perception begins to reassert itself..." (Merleau-Ponty, 1962, p.285)

realizable idealizations, and gradually exploring how behavior is altered. There is, in fact, a long history and existing precedent for studying intelligent systems under sensor perturbations. Psychologist George Stratton (shown in Figure 20) pioneered the study of perception in human vision by having subjects wear special glasses that inverted images Stratton (1897). Stratton observed that after a relatively short adaptation period, the subjects began to perceive the world normally, in spite of the vertical inversion. This is effectively a sensor map $(x, y) \mapsto (x, -y)$ for suitably chosen coordinates. It is a continuous transformation, satisfying the monotonicity requirement identified in Example 5.3, and —as Stratton observed— the map is not destructive.

One recent formulation that emphasizes action from the outset is Erdmann's more recent work on strategy complexes (Erdmann, 2012, 2010), as referenced in Example 3.5. He uses tools from classic and computational topology to relate plans, formulated broadly to include sources of non-determinism, to high-dimensional objects—his loopback complexes—whose homotopy type provides information about whether the planning problem can be solved. We speculate that preservation of plan existence under label maps might be productively studied across planning problems by examining the map's operation on loopback complexes: classes of maps that can be shown to preserve the homotopy type of such complexes (perhaps over restricted classes of planning problems) can be declared non-destructive.

An alternative approach, with goals similar to our own —namely of identifying representational basis for objects that can manipulated by algorithms in order to guide the design process— is due to Censi (2017). He poses and solves co-design problems; ascertaining the maximal task set achievable for a given set of resources. He shows that, given a network of monotone constraints, the selection of components is a process that can be efficiently automated. Part of the present interest in studying labels maps is that they can model aspects of different components.

Also adopting an algorithmic stance on the design process, are methods based on hybrid automata, which blend discrete and continuous elements. Powerful synthesis and verification techniques are known for these models Belta et al. (2007); Raman et al. (2015); DeCastro and Kress-Gazit (2016). Despite some similarities, including extensive use of non-determinism, the relationship between p-graphs and hybrid automata is somewhat involved: guard expressions in a rich logical specification language have structure missing from the label sets we study; the action labels in p-graphs are not intended to model continuous dynamics.

Though the present paper has focused on generalization and idealization to a degree perhaps uncommon in the robotics literature, this abstract style of approach in fundamental treatments of behavior appears in other settings. The natural question is how these treatments are related. At least for the question of bisimulation, a notion of equivalence employed in process algebras (and, importantly with respect to the present study, along with some generalizations to systems with continuous dynamics, see Haghverdi et al. (2005)), one of the authors has recently obtained a clear result on the relationship between the bisimulation relation and filter reduction. Rahmani and O'Kane (2018) show that filter reduction (see Section 5.2) can be achieved by quotienting an input filter by some relation and that bisimilarity is not the correct notion of equivalence for some types of filters.

8 Conclusion

This paper introduced and explored formalisms for reasoning about interactions between robots and their environments, including interaction languages, p-graphs, and label maps. We believe that the most crucial intellectual contributions of the present work are in attaining a degree abstractness missing from prior ideas in two ways.

First, we separate those entities which have been formalized in robotics because they have some interpretation that is useful (e.g., the idea of a plan, a filter), from their representation. The p-graph, in and of itself, lacks an obvious interpretation. Its definition does not include semantics belying a single anticipated use, rather context and any specific interpretation are only added for the special subclasses. In this sense, it is identical to the abstract treatment of computation as the constructive process of realizing a correspondence from inputs to outputs.

Second, even if something like a p-graph is a representation that is general enough to express many items of interest, it is not a *canonical form*. This paper engenders an important mental shift in lifting most of the notions of equivalence up to sets of executions, via interaction languages, rather than depending on operations on some specific graph. The present work continues to separate the notion of behavior from presentation. This helps establish a foundation for the semantics of the coupled robot-environment system.

The theoretical groundwork laid by this paper for reasoning about sensors and actuators, and their associated estimation and planning processes, aims to strengthen the link between idealized models and practical systems. It is imperative that we close the gap between robotics science and robotics practice, and more work remains to be done. We submit that it should be work aimed unambiguously and explicitly at that gap.

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grants IIS-1527436, IIS-1526862, IIS-0953503, IIS-1453652. We thank Yulin Zhang for his close reading of earlier versions of this work.

References

Belta C, Bicci A, Egerstedt M, Frazzoli E, Klavins E and Pappas GJ (2007) Symbolic Control and Planning of Robotic Motion. *IEEE Transactions on Robotics and Automation* 14(1): 51–70.

Censi A (2017) A Class of Co-Design Problems With Cyclic Constraints and Their Solution. *IEEE Robotics and Automation Letters* 2(1): 96–103.

DeCastro JA and Kress-Gazit H (2016) Nonlinear controller synthesis and automatic workspace partitioning for reactive high-level behaviors. In: *Proc. ACM International Conference on Hybrid Systems: Computation and Control*.

Dissanayake G, Newman P, Clark S, Durrant-Whyte H and Csorba M (2001) A solution to the simultaneous localisation and map building (SLAM) problem. *IEEE Transactions on Robotics and Automation* 17(3): 229–241.

Erdmann M (2010) On the topology of discrete strategies. *International Journal of Robotics Research* 29(7): 855–896.

Erdmann M (2012) On the topology of discrete planning with uncertainty. in advances in applied and computational topology. In: Zomorodian A (ed.) *Proc. Symposia in Applied Mathematics*, volume 70. American Mathematical Society.

Erdmann M and Mason MT (1988) An Exploration of Sensorless Manipulation. *IEEE Transactions on Robotics and Automation* 4(4): 369–379.

Erickson LE, Yu J, Huang Y and LaValle SM (2014) Counting moving bodies using sparse sensor beams. *IEEE Transactions on Automation Science and Engineering* 10(4): 853–861.

Garey MR and Johnson DS (1979) *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: W.H. Freeman and Company.

Ghasemlou S, Saberifar FZ, O'Kane JM and Shell DA (2016) Beyond the planning potpourri: reasoning about label transformations on procrustean graphs. In: *Proc. International Workshop on the Algorithmic Foundations of Robotics*.

Gierl D, Bobadilla L, Sanchez O and LaValle SM (2014) Stochastic modeling, control, and verification of wild bodies. In: *Proc. IEEE International Conference on Robotics and Automation*.

Goldberg KY (1993) Orienting Polygonal Parts Without Sensors. *Algorithmica* 10: 201–225.

Haghverdi E, Tabuada P and Pappas GJ (2005) Bisimulation relations for dynamical, control and hybrid systems. *Theoretical Computer Science* 342(2–3): 229–261.

iRobot Corp (2015) iRobot® Create® 2 Open Interface (OI). Technical report. Last Updated April 2, 2015.

Kalman RE (1960) A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering* 82(Series D): 35–45.

Kristek S and Shell DA (2012) Orienting Deformable Polygonal Parts without Sensors. In: *Proc. of the IEEE/RSJ International Conference on Intelligent Robot Systems (IROS)*.

LaValle SM (2006) *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press. Available at `http://planning.cs.uiuc.edu/`.

LaValle SM (2012) Sensing and Filtering: A Fresh Perspective Based on Preimages and Information Spaces. *Foundations and Trends in Robotics* 1(4): 253–372.

Lopez-Padilla R, Murrieta-Cid R and LaValle SM (2012) Optimal gap navigation for a disc robot. In: *Proc. Workshop on the Algorithmic Foundations of Robotics*.

Mason MT, Goldberg KY and Taylor RH (1988) Planning Sequences of Squeeze-Grasps to Orient and Grasp Polygonal Objects. In: *Seventh CISM-IFToMM Symposium on Theory and Practice of Robots and Manipulators*.

Merleau-Ponty M (1962) *Phenomenology of perception*. Oxon, U.K.: Routledge. Translated by Colin Smith.

O'Kane JM (2011) Decentralized Tracking of Indistinguishable Targets using Low-Resolution Sensors. In: *Proc. International Conference on Robotics and Automation*.

O'Kane JM and Shell D (2017) Concise planning and filtering: Hardness and algorithms. *IEEE Transactions on Automation Science and Engineering* 14(4): 1666–1681.

Rahmani H and O'Kane JM (2018) On the relationship between bisimulation and combinatorial filter reduction. In: *Proc. IEEE International Conference on Robotics and Automation*.

Raman V, Piterman N, Finucane C and Kress-Gazit H (2015) Timing semantics for abstraction and execution of synthesized high-level robot control. *IEEE Transactions on Robotics* 31.

Saberifar FZ, Ghasemlou S, O'Kane JM and Shell D (2016) Set-labelled filters and sensor transformations. In: *Proc. Robotics: Science and Systems*.

Schoppers MJ (1987) Universal Plans for Reactive Robots in Unpredictable Environments. In: *Proc. International Joint Conference on AI*. pp. 1039–1046.

Smith R, Self M and Cheeseman P (1990) Estimating uncertain spatial relationships in robotics. In: Cox I and Wilfong G (eds.) *Autonomous Robot Vehicles*. Berlin, Heidelberg: Springer, pp. 167–193.

Song Y and O'Kane JM (2012) Comparison of Constrained Geometric Approximation Strategies for Planar Information States. In: *Proc. International Conference on Robotics and Automation*.

Stratton GM (1897) Vision without inversion of the retinal image. *Psychological Review* 4(4): 341–360.

Thrun S, Burgard W and Fox D (2005) *Probabilistic Robotics*. Cambridge, MA.: MIT Press.

Tovar B, Cohen F, Bobadilla L, Czarnowski J and LaValle SM (2014) Combinatorial filters: Sensor beams, obstacles, and possible paths. *ACM Transactions on Sensor Networks* 10(3).

Tovar B, Murrieta-Cid R and LaValle SM (2007) Distance-optimal navigation in an unknown environment without sensing distances. *IEEE Transactions on Robotics* 23(3): 506–518.

Veanes M, Hooimeijer P, Livshits B, Molnar D and Bjorner N (2012) Symbolic finite state transducers: algorithms and applications. In: *Proc. of ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'12)*. New York, NY, USA, pp. 137–150.

Yu J and LaValle SM (2012) Shadow Information Spaces: Combinatorial Filters for Tracking Targets. *IEEE Transactions on Robotics* 28(2): 440–456.

Zuckerman D (2007) Linear degree extractors and the inapproximability of max clique and chromatic number. *Theory of Computing* 3: 103–128.