Formalizing Sample Transformation Plans

Daniel Bryce† and **Robert P. Goldman**† and **Jacob Beal**‡
Ugur Kuter† and Alexander F. Plotnick† and Matthew DeHaven† and Christopher W. Geib†
Nicholas Roehner‡ and Bryan Bartley‡
†SIFT, LLC
Minneapolis, MN, USA
{dbryce, rpgoldman, ukuter, mdehaven, aplotnick, cgeib}@sift.net
‡Raytheon BBN Technologies
Cambridge, MA, USA
jacob.beal@ieee.org, nicholas.roehner@raytheon.com, bryan.a.bartley@raytheon.com

Abstract

Experimental protocols are typically represented in either a natural language that is hard to replicate or compare, or in procedural languages that are difficult to automatically synthesize, detach from a specific experimental design for reuse, or analyze. We introduce a new approach based on techniques from automated planning. We describe how to represent transformation operators that manipulate samples in terms of applying conditions to samples. We define the semantics of this representation. We also present a simplified version of the notation that removes much of the modeling burden required of scientists. The resulting representation supports automated planning, provides sample provenance and metadata tracking at no cost by virtue of a plan's causal structure, and separates protocol specification from experimental design.

Introduction

Synthetic biology experiments involve preparing and then measuring samples under a number of conditions. In this work, we formalize sample preparation as reasoning about plans. Our contributions are to: 1) decouple experimental design from experimental protocols, and 2) simplify the specification of experimental protocols. Separating the design of experiments consideration of *what* samples to measure from *how* to create those samples supports reuse of costly-to-develop experimental protocols. Simplifying experimental protocol specifications enables scientists to rapidly develop new experiments. We focus upon using planning representations to structure experiment specifications, but also briefly discuss how automated planning engines can use these representations to search alternative protocol instantiations and steps.

Two of the representational challenges for applying planning to reasoning about samples are: 1) capturing information about the conditions applied to each sample, 2) describing how protocol operators apply conditions to samples. We must describe the conditions applied to each sample so that we can ensure that the protocol constructs the correct set of samples specified by the experimental design. Protocol operators may apply to only some samples, and apply only some conditions. There can be many combinations of condition-value pairs, making them hard to manually enumerate. Furthermore, manually enumerating these condition-value pairs couples the protocol with the experimental design.

Prior work has described experimental protocols with either procedural (Klavins 2018) or low-level declarative (Autoprotocol 2018) languages. While these works can offer the convenience and power of general purpose programming languges, they are not immediately amenable to automated search, require custom implementations of protocol operators, and do not inherently track sample provenance. Automating search to construct a protocol in a general purpose language can amount to the notoriously difficult problem of program synthesis. Using custom implementations of operators does not necessarily enforce operator structure that generalizes to a common condition-application semantics. Procedural protocol descriptions require protocol developers to adopt a custom provenance tracking methodology that is otherwise readily available in the causal structure of plans.

We introduce a semantics for sample transformation operators that involves three main conventions: 1) disjunctive preconditions that specify condition-values that samples must satisfy to be transformed by an operator, 2) disjunctive effects that specify condition-values to associate with alternative output samples based upon input samples, and 3) a way to distinguish between conditions assigned versus applied to samples. We then present a simplified representation of the operators that effectively distributes disjunction in preconditions and effects from a disjunctive normal form into a conjunctive normal form. The simplified form: 1) decouples operator specifications from a specific experimental design, 2) is more compact, and 3) applies the experimental design with the operator specification to compute sample transformations. We illustrate several operators that use our formalization in two genetic circuit characterizations, protein design, and riboswitch characterization. We also discuss how the automated planner `XPlan` uses these operators to construct experimental plans.

Background

An experimental design is a set D of samples. Each sample $s \in D$ is a set of condition-value pairs of the form (c, v). For example, one of our circuit characterization problems includes a design of the form:

$$\begin{aligned}\{&\{(\text{strain}, \text{UWBF\_6389}), (\text{replicate}, 0), (\text{media}, m4),\\ &\quad (OD, 0.003), (\text{timepoint}, 1)\},\\ &\{(\text{strain}, \text{UWBF\_6390}), (\text{replicate}, 0), (\text{media}, m4),\\ &\quad (OD, 0.003), (\text{timepoint}, 1)\},\\ &\{(\text{strain}, \text{UWBF\_6391}), (\text{replicate}, 0), (\text{media}, m4),\\ &\quad (OD, 0.003), (\text{timepoint}, 1)\},\\ &\{(\text{strain}, \text{UWBF\_6388}), (\text{replicate}, 0), (\text{media}, m4),\\ &\quad (OD, 0.003), (\text{timepoint}, 1)\},\\ &\quad \ldots\\ \}\end{aligned}$$

where there are a number of conditions such as strains, replicates, media, target optical densities, and timepoints (in a time series).

For the purpose of this discussion, let C be a finite set of conditions. We also assume a given sample transformation plan O that corresponds to an experimental protocol. A transformation plan $O = (o_1, \ldots, o_n)$ is a sequence of operators o. For example, we use the following plan in our circuit characterization problem:

$$\begin{aligned}(&\text{streak}, \text{incubate}, \text{pick}, \text{incubate}, \text{spectrophotometry}, \text{dilute},\\ &\text{incubate}, \text{transfer}, \text{mix}, \text{spectrophotometry}, \text{flow\_cytometry})\end{aligned}$$

where we include streak, incubate, spectrophotometry, mix, and flow_cytometry for the sake of completeness but these operators do not transform samples in the sense described here.

An *experiment* is a tuple (S_I, O, D) where S_I is an initial set of samples, O is the operator sequence, and D is the experimental design. An experiment satisfies the experimental design if it produces a set of samples $S_G = \mathtt{transform}(S_I, O)$ so that S_G satisfies D. In the following sections, we define two ways to compute S_G that use alternative representations of the transformation operators O. In both cases, the result of transforming a sample set S_I by a sequence of operators $O = (o_1, \ldots, o_n)$ is

$$\begin{aligned}S'_G &= \mathtt{transform}(S, O)\\ &= \mathtt{transform}(\ldots \mathtt{transform}(S_I, o_1) \ldots, o_n)\end{aligned}$$

where the difference is in how we define the transformation $\mathtt{transform}(S, o)$ involving a single operator o.

Semantics of Sample Transformation Plans

We define the semantics of a *sample transformation plan* as a sequence of operators $(o_1, \ldots, o_n)$ that either assign or apply conditions $c \subseteq C$ to a set of initial samples S_I to produce a set of goal samples S_G. We distinguish assignment from application to capture plans that pre-allocate identical samples to different physical containers prior to applying a condition that distinguishes the samples.

Each condition $c \in C$ can take on a value $v \in V(c)$ where V is the domain of values. Each sample s is a set of condition-value-applied triples (c, v, a), where $c \in C$, $v \in V(c)$, and a is a boolean (i.e., $a \in \mathbb{B} = \{\top, \bot\}$). For example, a triple $(c_1, v_1, \bot)$ denotes that the sample has been assigned condition c_1 with value v_1. The triple $(c_2, v_2, \top)$ represents that we've applied condition c_2 with value v_2. For example, in our circuit characterization experiment, S_I contains samples of the form:

$$\{(\text{strain}, \text{UWBF\_6390}, \top)\}$$

where we start the experiment with samples where the strain has been applied.

Each operator $o \in O$ defines a disjunctive set of precondition sets $\mathtt{Pre}(o)$. Each $\mathtt{pre}(o) \in \mathtt{Pre}(o)$ is a conjunctive set of triples so that $\mathtt{pre}(o) \subseteq C \times V \times \mathbb{B}$. An operator o is *applicable* to a sample s iff s satisfies at least one $\mathtt{pre}(o) \in \mathtt{Pre}(o)$ (i.e., $\mathtt{pre}(o) \subseteq s$). In the following, we will assume that each pair of precondition sets $\mathtt{pre}(o), \mathtt{pre}'(o) \in \mathtt{Pre}(o)$ is mutually exclusive. For example, $\mathtt{Pre}(\text{pick})$ defines which strains to pick from a culture and inoculate a resulting sample:

$$\begin{aligned}\{&\{(\text{strain}, \text{UWBF\_6389}, \top)\},\\ &\{(\text{strain}, \text{UWBF\_6390}, \top)\},\\ &\{(\text{strain}, \text{UWBF\_6391}, \top)\},\\ &\{(\text{strain}, \text{UWBF\_6388}, \top)\},\\ &\ldots\}\end{aligned}$$

Each operator $o \in O$ also defines a disjunctive set of effect sets $\mathtt{Eff}(o)$. Each $\mathtt{eff}(o) \in \mathtt{Eff}(o)$ is a conjunctive set of triples so that $\mathtt{eff}(o) \subseteq C \times V \times \mathbb{B}$. Unlike the traditional interpretation that disjunctive effects are non-deterministic, we use the disjunction to signify "splitting" an input sample into output samples. Each output sample is consistent with an alternative disjunct. For example, $\mathtt{Eff}(\text{pick})$ defines which media and replicate id to apply to each input sample:

$$\begin{aligned}\{&\{(\text{media}, \text{m4}, \top), (\text{replicate}, 1, \top)\},\\ &\{(\text{media}, \text{m4}, \top), (\text{replicate}, 2, \top)\},\\ &\{(\text{media}, \text{m4}, \top), (\text{replicate}, 3, \top)\},\\ &\{(\text{media}, \text{m4}, \top), (\text{replicate}, 4, \top)\},\\ &\ldots\}\end{aligned}$$

If we apply operator $o \in O$ to a sample set S, then we will create sample set S', defined as follows. The resulting sample set S' is the union of transforming each sample in S:

$$S' = \mathtt{transform}(S, o) = \bigcup_{s \in S} \mathtt{transform}(s, o)$$

If a sample s satisfies the precondition (i.e., $\exists \mathtt{pre}(o) \in \mathtt{Pre}(o).\mathtt{pre}(o) \subseteq s$), then we transform it with all possible effect sets:

$$\mathtt{transform}(s, o) = \{\mathtt{transform}(s, \mathtt{eff}(o)) | \mathtt{eff}(o) \in \mathtt{Eff}(o)\}$$

otherwise samples not satisfying the precondition define $\mathtt{transform}(s, o) = \{s\}$ (a frame axiom). Transforming a sample s with a single effect set $\mathtt{eff}(o) = \{(c_1, v_1, a_1), \ldots, (c_k, v_k, a_k)\}$ will result in a sample s' defined by

$$\begin{aligned}s' &= \mathtt{transform}(s, \mathtt{eff}(o))\\ &= \mathtt{transform}(\ldots \mathtt{transform}(s, (c_k, v_k, a_k)) \ldots, (c_1, v_1, a_1))\end{aligned}$$

where $s' = s$ if $\texttt{eff}(o) = \{\}$. Transforming a sample s by a single triple (c, v, a) defines a sample s' as

$$\begin{aligned} s' &= \texttt{transform}(s, (c, v, a)) \\ &= s \backslash \{(c, v', a') | (c, v', a') \in s\} \cup \{(c, v, a)\} \end{aligned}$$

which removes any triples referring to the same condition c and adds the new triple. For example, if we transform the sample set S:

$$\begin{aligned} \{&\{(\text{strain}, \text{UWBF\_6389}, \top)\}, \\ &\{(\text{strain}, \text{UWBF\_6390}, \top)\}, \\ &\{(\text{strain}, \text{UWBF\_6391}, \top)\}, \\ &\{(\text{strain}, \text{UWBF\_6388}, \top)\}, \\ &\ldots\} \end{aligned}$$

with the pick operator, then we will create the sample set S':

$$\begin{aligned} \{&\{(\text{strain}, \text{UWBF\_6389}, \top), (\text{media}, \text{m4}, \top), (\text{replicate}, 1, \top)\}, \\ &\{(\text{strain}, \text{UWBF\_6390}, \top), (\text{media}, \text{m4}, \top), (\text{replicate}, 1, \top)\}, \\ &\{(\text{strain}, \text{UWBF\_6391}, \top), (\text{media}, \text{m4}, \top), (\text{replicate}, 1, \top)\}, \\ &\{(\text{strain}, \text{UWBF\_6388}, \top), (\text{media}, \text{m4}, \top), (\text{replicate}, 1, \top)\}, \\ &\{(\text{strain}, \text{UWBF\_6389}, \top), (\text{media}, \text{m4}, \top), (\text{replicate}, 2, \top)\}, \\ &\{(\text{strain}, \text{UWBF\_6390}, \top), (\text{media}, \text{m4}, \top), (\text{replicate}, 2, \top)\}, \\ &\{(\text{strain}, \text{UWBF\_6391}, \top), (\text{media}, \text{m4}, \top), (\text{replicate}, 2, \top)\}, \\ &\{(\text{strain}, \text{UWBF\_6388}, \top), (\text{media}, \text{m4}, \top), (\text{replicate}, 2, \top)\}, \\ &\ldots\} \end{aligned}$$

A plan *satisfies* the experimental design if S_G satisfies D. That is, $\forall s \in D. \exists s' \in S_G. \forall (c, v) \in s. \exists (c, v, \top) \in s'$.

Simplified Transformations

The $\texttt{Pre}(o)$ and $\texttt{Eff}(o)$ sets can be quite large, and are specific to a given domain V of values for each condition. Furthermore, for a given operator sequence $O = (o_1, \ldots, o_n)$ this presumes a fixed experimental design that (implicitly) prescribes which samples to produce. In order to improve the reuse of operators for a variety of experiments, we present a domain agnostic language for specifying transformations. However, what we gain in simplicity, we give up in precision – the language assumes that all values in the domain $V(c)$ apply to every transformation for a given condition c. To regain the lost precision, we assume an experimental design D that culls undesirable condition values.

We define simplified preconditions $\overline{\texttt{pre}}(o) = \{(c_1, V_1, a_1), \ldots, (c_k, V_k, a_k)\}$ and effects $\overline{\texttt{eff}}(o) = \{(c_1, V_1, a_1), \ldots, (c_k, V_k, a_k)\}$ for each operator. Each set of values $V_i \subseteq V(c_i)$ specifies the possible values for condition c_i. We interpret $V_i = \{\}$ to mean that all values in $V(c_i)$ are allowed, which allows us (in this case) to decouple the operator description from any specific domain $V(c_i)$. (Tuples of the form $(c, \{\}, a)$ would otherwise assert that the condition must hold no value, which doesn't bear meaning for us.) For example, the pick operator from the previous section defines $\overline{\texttt{pre}}(\text{pick})$ as:

$$\{(\text{strain}, \{\}, \top)\}$$

and $\overline{\texttt{eff}}(\text{pick})$ as:

$$\{(\text{media}, \{\}, \top), (\text{replicate}, \{\}, \top)\}$$

We can convert from the simplified format to the original format by distributing the disjunction so that if $\overline{\texttt{pre}}(o) = \{(c_1, V_1, a_1), \ldots, (c_k, V_k, a_k)\}$, then for each $\{v_1, \ldots, v_k\} \in V_1 \times \ldots \times V_k$ there exists a $\texttt{pre}(o) \in \texttt{Pre}(o)$ where $\texttt{pre}(o) = \{(c_1, v_1, a_1), \ldots, (c_k, v_k, a_k)\}$. For example, if we have the following condition value domains:

$$\begin{aligned} V(\text{strain}) &= \{\text{UWBF\_6389}, \\ &\qquad \text{UWBF\_6390}, \\ &\qquad \text{UWBF\_6391}, \\ &\qquad \text{UWBF\_6388}\} \\ V(\text{media}) &= \{\text{m1}, \text{m2}, \text{m3}, \text{m4}\} \\ V(\text{replicate}) &= \{1, 2, 3, 4, 5, 6\} \end{aligned}$$

we can covert our pick operator in the simplified representation to the original representation, as listed in the previous section.

Under the simplified model, we define transformations differently. As before, if we apply a simplified operator $o \in O$ to a sample set S, then we will create sample set S', where S' is the union of transforming each sample in S:

$$S' = \overline{\texttt{transform}}(S, o) = \bigcup_{s \in S} \overline{\texttt{transform}}(s, o)$$

If a sample s satisfies the simplified precondition (i.e., $\forall (c, V, a) \in \overline{\texttt{pre}}(o) \exists (c, v, a) \in s. v \in V \vee V = \{\}$), then we transform it with $\overline{\texttt{eff}}(o)$, so that $\overline{\texttt{transform}}(s, o) = \overline{\texttt{transform}}(s, \overline{\texttt{eff}}(o))$ is defined as:

$$\begin{aligned} &\overline{\texttt{transform}}(s, \overline{\texttt{eff}}(o)) = \\ &\quad \overline{\texttt{transform}}(\ldots \overline{\texttt{transform}}(s, (c_k, V_k, a_k)) \ldots, (c_1, V_1, a_1)) \end{aligned}$$

otherwise samples not satisfying the precondition define $\overline{\texttt{transform}}(s, o) = \{s\}$. Transforming a sample s with a triple (c, V, a) will result in a set of samples defined by

$$\begin{aligned} \overline{\texttt{transform}}(s, (c, V, a)) = \{s_v | s_v &= \texttt{transform}(s, (c, v, a)), \\ &\texttt{consistent}(s_v, D), \\ &v \in V\} \end{aligned}$$

where $\texttt{transform}(s, (c, v, a))$ is defined as in the previous section. The $\texttt{consistent}$ check is defined as:

$$\texttt{consistent}(s, D) = \exists s' \in D. \forall (c, v, a) \in s. (c, v) \in s'$$

We require the extra $\texttt{consistent}$ check on all intermediate samples because of the effect independence introduced by the simplification. Otherwise, it is possible construct samples that are inconsistent with the experimental design D. To see this, consider a hypothetical effect set $\texttt{Eff}(o)$ defined as:

$$\begin{aligned} \{&\{(c_1, v_{11}, a_1), (c_2, v_{12}, a_2)\}, \\ &\{(c_1, v_{21}, a_1), (c_2, v_{22}, a_2)\}, \\ &\{(c_1, v_{11}, a_1), (c_2, v_{22}, a_2)\}\} \end{aligned}$$

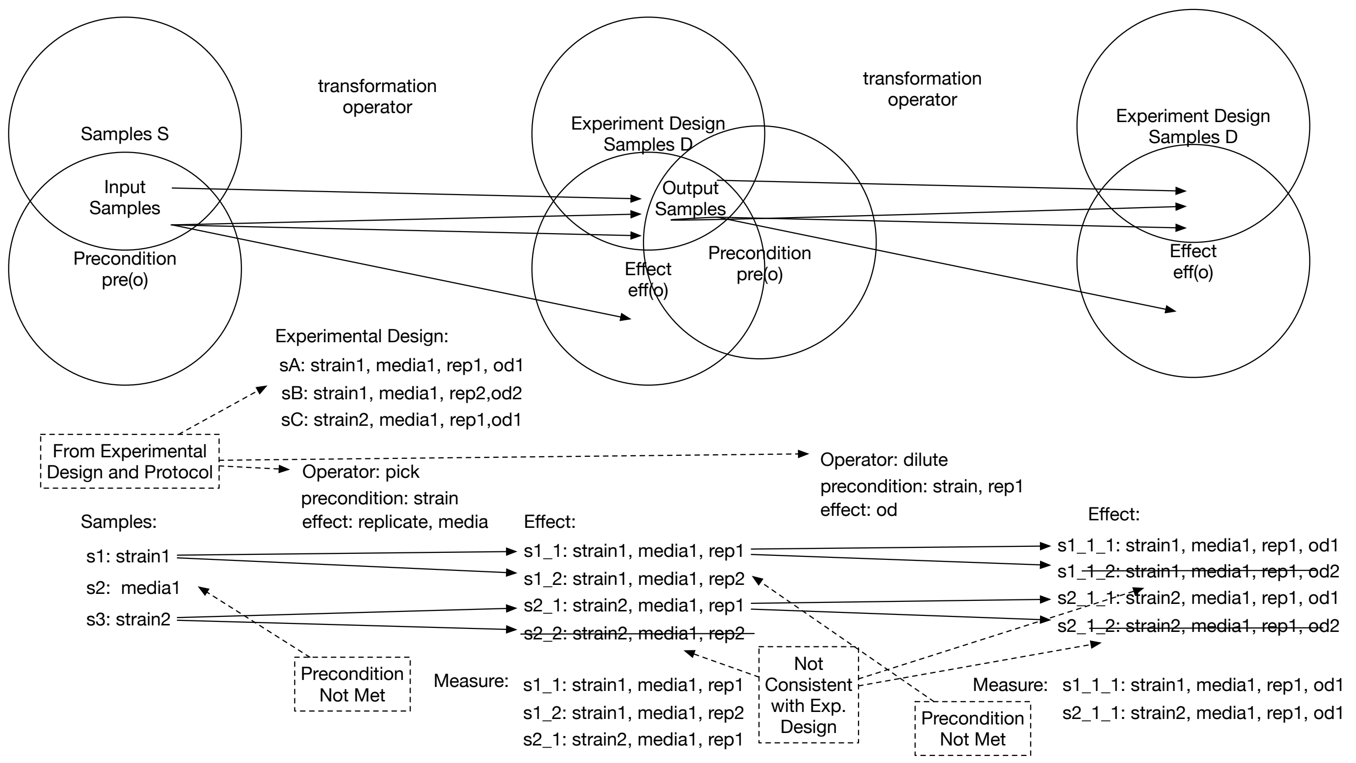
transformation operator
Samples S
Input Samples
Precondition pre(o)
Experiment Design Samples D
Output Samples
Effect eff(o)
Precondition pre(o)
transformation operator
Experiment Design Samples D
Effect eff(o)
Experimental Design:
sA: strain1, media1, rep1, od1
sB: strain1, media1, rep2,od2
sC: strain2, media1, rep1,od1
From Experimental Design and Protocol
Operator: pick
precondition: strain
effect: replicate, media
Operator: dilute
precondition: strain, rep1
effect: od
Samples:
s1: strain1
s2: media1
s3: strain2
Effect:
s1_1: strain1, media1, rep1
s1_2: strain1, media1, rep2
s2_1: strain2, media1, rep1
~~s2_2: strain2, media1, rep2~~
Precondition Not Met
Measure:
s1_1: strain1, media1, rep1
s1_2: strain1, media1, rep2
s2_1: strain2, media1, rep1
Not Consistent with Exp. Design
Effect:
s1_1_1: strain1, media1, rep1, od1
s1~~_1_2: strain1, media1, rep1, od2~~
s2_1_1: strain2, media1, rep1, od1
s2~~_1_2: strain2, media1, rep1, od2~~
Precondition Not Met
Measure:
s1_1_1: strain1, media1, rep1, od1
s2_1_1: strain2, media1, rep1, od1

Figure 1: Each transformation operator applies to samples satisfying its precondition, and creates samples consistent with its effect. The simplified representation relies upon target samples to restrict the set of samples produced by each operator.

where we have purposely omitted the effect:

$$\{(c_1, v_{21}, a_1), (c_2, v_{12}, a_2)\}$$

Under the simplified representation we would have the following condition-value domains and effect $\overline{\texttt{eff}}(o)$:

$$\begin{aligned}
V(c_1) &= \{v_{11}, v_{21}\},\\
V(c_2) &= \{v_{12}, v_{22}\},\\
\overline{\texttt{eff}}(o) &= \{(c_1, V(c_1), a_1), (c_2, V(c_2), a_2\}
\end{aligned}$$

which could produce samples:

$$\begin{gathered}
\{\{(c_1, v_{11}, a_1), (c_2, v_{12}, a_2)\},\\
\{(c_1, v_{21}, a_1), (c_2, v_{22}, a_2)\},\\
\{(c_1, v_{11}, a_1), (c_2, v_{22}, a_2)\},\\
\{(c_1, v_{21}, a_1), (c_2, v_{12}, a_2)\}\}
\end{gathered}$$

However, if we assume that the experimental design D defines:

$$\begin{gathered}
\{\{(c_1, v_{11}), (c_2, v_{12})\},\\
\{(c_1, v_{21}), (c_2, v_{22}\},\\
\{(c_1, v_{11}), (c_2, v_{22})\}\}
\end{gathered}$$

then the sample

$$\{(c_1, v_{21}, a_1), (c_2, v_{12}, a_2)\}$$

would not be consistent with D and removed from the result.

Example: Figure 1 illustrates two steps of a sample transformation plan. The Venn diagrams at the top conceptualize transformation operators in the simplified format. The input to a transformation operator is intersection of a current sample set S with the operator precondition $\overline{\texttt{pre}}(o)$. The operator produces samples that are consistent with its effect $\overline{\texttt{eff}}(o)$, which are intersected with the experimental design D. This continues with the second operator in a similar fashion. The lower part of the figure illustrates a toy example where we begin with samples $\{s_1, s_2, s_3\}$ and transform them with the pick operator. We use a shortened notation where, for example, strain1 is equivalent to a condition-value triple $(\text{strain}, 1, \top)$. The pick operator requires that input samples have a strain, and has the effect of applying replicate and media conditions. Not all output samples are consistent with the experimental design, so those with a line through them are removed. We also illustrate the dilute operator, which applies only to samples with a condition-value tripe $(\text{replicate}, 1, \top)$ as denoted by the shortened notation rep1. The example illustrates how both preconditions and the experimental design reduce the possible samples produced by the effects.

Discussion: The simplified representation decouples the experimental design from the experiment protocol. This allows us to use the same operators across multiple experimental designs, which is not possible with the original representation. We also have the added benefit that the simplified operator specification is much more compact, making it easier to engineer and maintain. This division between experimental design and experimental protocol also follows how most scientists conceptualize experiments – separating the "what" and the "how". Finally, we provide a formal semantics for the simplified representation by connecting it to the semantics of the original representation.

Transformation Operators

In this section, we discuss several examples of other transformation operators used across several distinct experiments. We introduce a simple syntax to describe the operators, that resembles the Planning Domain Definition Language (PDDL) (Ghallab et al. 1998). Each operator consists of a name, a precondition list, and an effect list. Each element of a precondition and effect list is a condition-value tuple, as defined by the previous section.

The pick operator models inoculating a sample in media from a colony. It requires that the sample has a strain, and associates a media and replicate condition-value triple with each output sample:

```
(pick
 :precondition ((strain nil t))
 :effect ((media nil t)
          (replicate nil t)))
```

Experiments use the pick operator to inoculate wells in a container (e.g., 96-well plate) from another container (e.g., a YPAD plate).

The dilute operator models diluting a sample so that it reaches a specified optical density:

```
(dilute
 :precondition ((strain nil t))
 :effect ((od nil t)))
```

In our gate characterization experiments, the dilute operator follows a pick operator. At that step in the experiment, the samples have a number of condition-values applied to them. The dilute operator illustrates the common benefit of action languages – it specifies a complex transition function in terms of a local updates to a few conditions.

In another gate characterization experiment, we use the transfer operator to create multiple samples (filling a 96-well plate):

```
(transfer
 :precondition ((strain nil t))
 :effect ((control (nil) t)
          (replicate nil t)
          (input nil nil)
          (media nil t)))
```

The transfer operator applies to all samples with a strain condition-value. It creates samples that are not controls (the values for the condition 'control' are logical false, i.e., (nil)). It creates replicates of each input sample. It *assigns* the input condition to each sample without applying it. A later operator *applies* the input condition (i.e., introduces small molecules that signal the gate inputs). Rather than prepare a large volume of sample for all possible input conditions, the experiment prepares a small volume for each input *a priori*. Finally, the transfer operator also applies the media condition by combining the strains with media.

The facs-seq-round-1 operator, in a protein design experiment, illustrates a case where the set of values for a condition are partially decoupled from the experimental design.

```
(facs-seq-round-1
 :precondition ((library nil t))
 :effect ((round (1) t)
          (treatment nil t)
          (concentration nil t)))
```

The operator requires protein library samples as inputs, and associates a (round, 1, $\top$) condition-value triple with each output sample. It also associates condition-value triples for treatment and concentration conditions. The operator refers to a specific value for round because it might otherwise produce samples for later rounds, which are part of the experimental design. It does, however, decouple the treatment and concentration associated with each sample from the experimental design. In the fashion, the operator is able to hold certain conditions constant and allows others to be free (up to the freedom specified in the experimental design).

Each of these transformation operators illustrates a different feature of our simplified transformation language. Across four experimental plans developed as part of our work, we required 4, 5, 5, and 20 transformation operators per plan. These respective plans involved transforming 3, 5, 192, and 9 input samples into a total of 34, 298, 2706, and 1573 respective samples. The third plan, a riboswitches characterization experiment, required a small amound of knowledge engineering, but involved transforming the most samples. We anticipate that our representation will lead to more efficient modeling and experiment planning due to its simplicity.

XPlan

Synthetic biology is the systematic engineering of living organisms to perform desired functions. For example, biological sensors have applications in sensing pathogens or biological, chemical, and radioactive weapons; effectors have applications in chemical synthesis and cleanup, and in targeted medical therapies. Because existing models for genetic structures, assembly, and expression are still relatively weak, however, synthetic biology necessarily involves both design and experimentation to assess the success of designs and identify factors responsible for success and failure.

DARPA's Synergistic Discovery and Design (SD2) program seeks to speed scientific and design processes through automated support for experiment planning, automated execution of experimental protocols across laboratories, and high-speed, large-scale exploratory data analysis. In SD2, our `XPlan` planner (Kuter et al. 2018) uses the simplified transformation operator language and semantics to construct experimental plans. Figure 2 shows a high-level `XPlan` architecture diagram, cf. (Kuter et al. 2018). `XPlan` is built on SHOP2 (Nau et al. 2003), a hierarchical task network (HTN) planner. `XPlan` uses HTN methods to structure experimental plans and search for operator sequences $(o_1, \ldots, o_n)$. Instead of computing the sample transformations as PDDL-

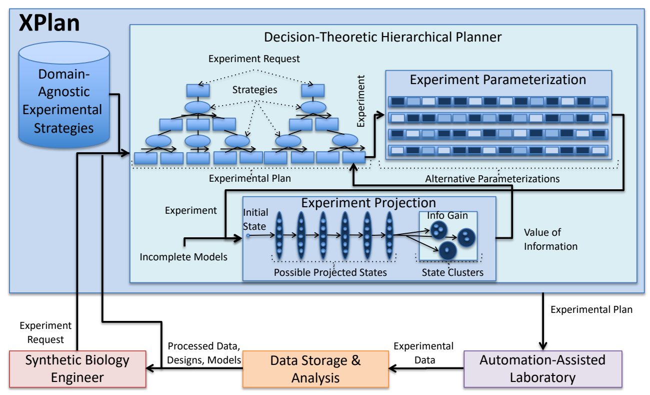
XPlan
Domain-Agnostic Experimental Strategies
Decision-Theoretic Hierarchical Planner
Experiment Request
Strategies
Experiment
Experiment Parameterization
Experimental Plan
Alternative Parameterizations
Experiment
Experiment Projection
Initial State
Info Gain
Incomplete Models
Possible Projected States
State Clusters
Value of Information
Experimental Plan
Experiment Request
Processed Data, Designs, Models
Experimental Data
Synthetic Biology Engineer
Data Storage & Analysis
Automation-Assisted Laboratory

Figure 2: `XPlan` combines domain-agnostic strategies and domain-specific knowledge to expand experiment requests into executable plans, which are then parameterized and projected for VOI analysis. Plans are given to laboratories to execute, producing data that results in updated models and designs and new experiment requests.

like operators, `XPlan` applies the sample transformations described herein to simplify the model development. Describing sample transformations as PDDL operators would require a complex and difficult to maintain set of conditional effects (similar to the original representation) or an ADL (Pednault 1989) operator with conditional effects and quantified variables.

Related Work and Discussion

Sample transformation plans implicitly represent a sample transformation graph that links samples to their predecessors and ancestors. This sample transformation graph is in correspondence with the causal structure of the plan. In order to construct this sample transformation graph, the transformation operators require condition-value triples that describe the input samples. These triples are similar in nature and mechanism to premises in truth maintenance systems (de Kleer 1986). They prescribe the conditions under which it is possible to deduce a sample.

Our language for sample transformation planning is different from existing languages for experimental protocol specification such as Trident (Klavins 2018) and Autoprotocol (Autoprotocol 2018). Trident provides scripting support for automating experiments, but is based upon a high-level procedural programming language (Python). Autoprotocol is a low-level execution language that omits much of the provenance and rationale for the experiment operators, in favor of prescribing unambiguous execution steps. The impact of these differences are that the languages do not natively describe the semantics of experiment steps for automating reasoning with a planner. These languages are similar to plan execution languages, such as PLEXIL (Verma et al. 2005), which the automated planning community has embraced as targets for plan compilation, but not plan synthesis.

Conclusion

We have described a semantics and simplified language for sample transformation planning that decouples experimental design from experimental protocol specification. The language captures common operators across four unique challenge problems in synthetic biology and protein design. The sample transformation plans encode sample provenance and sample attributes as part of the causal structure of the plan, encoding metadata that enables both machine learning applied to experimental data, but also experiment replicability. We have described how to reason about a single sequence of operators, but the language supports automated plan search and optimization. This semantics and language are the basis of the XPlan planner (based upon SHOP2 (Nau et al. 2003)), developed as part of the DARPA SD2 program.

References

Autoprotocol. 2018. *Autoprotocol: An open standard for life science experimental design and automation.* `http://autoprotocol.org` (accessed August 1, 2018).

de Kleer, J. 1986. An assumption-based TMS. *Artificial Intelligence* 28(2):127–162.

Ghallab, M.; Howe, A.; Knoblock, C.; Mcdermott, D.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL—The Planning Domain Definition Language.

Klavins, E. 2018. *Trident: The Aquarium API.* `http://klavinslab.org/trident/` (accessed August 1, 2018).

Kuter, U.; Goldman, R. P.; Bryce, D.; Beal, J.; Dehaven, M.; Geib, C. S.; Plotnick, A. F.; Nguyen, T.; and Roehner, N. 2018. Xplan: Experiment planning for synthetic biology. In *Proceedings of the ICAPS'18 Hierarchical Planning Workshop*.

Nau, D.; Au, T.-C.; Ilghami, O.; Kuter, U.; Murdock, W.; Wu, D.; and F.Yaman. 2003. Shop2: An htn planning system. *JAIR* 20:379–404.

Pednault, E. P. D. 1989. Adl: Exploring the middle ground between strips and the situation calculus. In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, 324–332. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Verma, V.; Estlin, T.; Jónsson, A.; Pasareanu, C.; Simmons, R.; and Tso, K. 2005. Plan execution interchange language (PLEXIL) for executable plans and command sequences. In *International Symposium on Artificial Intelligence, Robotics and Automation in Space (iSAIRAS)*.