DOI:10.1145/3441686

BioScript: Programming Safe Chemistry on Laboratories-on-a-Chip

By Jason Ott, Tyson Loveless, Chris Curtis, Mohsen Lesani, and Philip Brisk

Abstract

This paper introduces *BioScript*, a domain-specific language (DSL) for programmable biochemistry that executes on emerging microfluidic platforms. The goal of this research is to provide a simple, intuitive, and type-safe DSL that is accessible to life science practitioners. The novel feature of the language is its syntax, which aims to optimize human readability; the technical contribution of the paper is the *BioScript* type system. The type system ensures that certain types of errors, specific to biochemistry, do not occur, such as the interaction of chemicals that may be unsafe. Results are obtained using a custom-built compiler that implements the *BioScript* language and type system.

1. INTRODUCTION

The last two decades have witnessed the emergence of software-programmable laboratory-on-a-chip (pLoC) technology, enabled by technological advances in microfabrication and coupled with scientific understanding of microfluidics, the fundamental science of fluid behavior at the micro- to nanoliter scale. The net result of these collective advancements is that many experimental laboratory procedures have been miniaturized, accelerated, and automated, similar in principle to how the world's earliest computers automated tedious mathematical calculations that were previously performed by hand. Although the vast majority of microfluidic devices are effectively application-specific integrated circuits (ASICs), a variety of programmable LoCs have been demonstrated.[16, 18]

With a handful of exceptions, research on programming languages and compiler design for programmable LoCs has lagged behind their silicon counterparts. To address this need, this paper presents a domain-specific programming language (DSL) and type system for a specific class of pLoC that manipulate discrete droplets of liquid on a two-dimensional grid. The basic principles of the language and type system readily generalize to programmable LoCs, realized across a wide variety of microfluidic technologies.

The presented language, *BioScript*, offers a user-friendly syntax that reads like a cookbook recipe. *BioScript* features a combination of fluidic/chemical variables and operations that can be interleaved seamlessly with computation, if desired. Its intended user base is not traditional software developers, but life science practitioners, who are likely to balk at a language that has a steep learning curve.

BioScript's type system ensures that each fluid is never consumed more than once, and that unsafe combinations of chemicals—those belonging to conflicting reactivity groups, as determined by appropriately qualified government agencies—never interact on-chip; *BioScript*'s type system is based on union types and was designed to ensure that type inference is decidable. This will set the stage for future research on formal validation of biochemical programs.

The *BioScript* language and type system are evaluated using a set of benchmark applications obtained from scientific literature. We use a microfluidic simulator to assess performance under ideal operating conditions and also execute them on a real device, which is much smaller and supports a subset of *BioScript*'s operational capabilities. This result establishes the feasibility of high-level programming language and compiler design for programmable chemistry, and opens up future avenues for research in type systems and formal verification techniques within this nontraditional computing domain.

1.1. Digital microfluidic biochips (DMFBs)

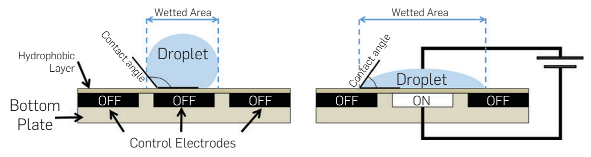
This paper targets a specific class of programmable LoCs that manipulate discrete droplets of fluid via electrostatic actuation. Figure 1a illustrates the electrowetting principle: applying an electrostatic potential to a droplet modifies the shape of the droplet and its contact angle with the surface.[10, 13] As shown in Figure 1b, droplet transport can be induced by activating and deactivating a sequence of electrodes adjacent to the droplet; the ground electrode, on top of the array, improves the fidelity of droplet motion and reduces the voltage required to induce droplet transport.

Figure 2a depicts a programmable 2D electrowetting array, called a digital microfluidic biochip (DMFB). A DMFB can support five basic operations, as shown in Figure 2b: transport (move a droplet from position (x, y) to (x', y')), split (create two smaller droplets from one larger droplet), merge (combine two droplets into 1), mix (rotate a merged droplet in a rectangular region around one or more pivots), and storage (place a droplet at position (x, y) for later use). A DMFB is reconfigurable, as these operations can be performed anywhere on the array, and any given electrode can be used to perform different operations at different times. Droplet I/O is performed using reservoirs on the perimeter of the chip, which are not depicted in Figure 2.

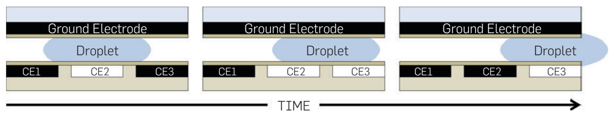
The DMFB instruction-set architecture (ISA) can be extended by integrating sensors, optical detectors, or

The original version of this paper was published in the *Proceedings of OOPSLA '18* (Boston, MA, Nov. 2018), Article 128.

Figure 1. The electrowetting principle (a) enables droplet transport (b).

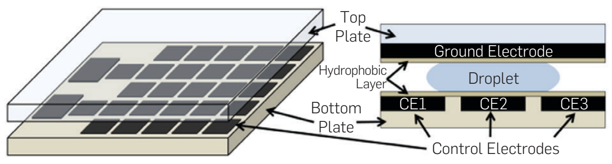


(a) The electrowetting principle:[10, 13] applying an electrostatic potential to a droplet at rest reduces the contact angle with the surface, thereby increasing the surface area in contact with the droplet

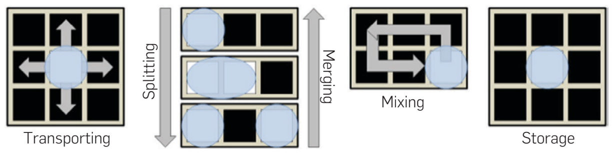


(b) A droplet is transported from control electrode CE2 to neighboring electrode CE3 by activating CE3, and then deactivating CE2 (white: activated electrode; black: deactivated electrode).

Figure 2. A DMFB (a) and its reconfigurable instruction set (b).



(a) Left: A DMFB is a planar array of electrodes.[15] Right: Cross-sectional view.



(b) The DMFB ISA supports five basic operations: transporting, merging, splitting, mixing and storage, in addition to I/O on the perimeter of the array.

online video monitoring capabilities. Sensors and actuators create a "cyber-physical" feedback loop between the host PC controller and the DMFB. The ability to perform sensing, computation, and actuation based on the results of the computation adds control flow to the instruction set of the DMFB. Prior work has applied feedback control for precise droplet positioning and online error detection and recovery[11, 12, 19] efforts to leverage these capabilities to provide control flow constructs at the language syntax level have been far more limited.

2. OVERVIEW

***BioScript* Syntax and Semantics.** *BioScript* is a language for programmable microfluidics whose syntax aims to be palatable to life science practitioners, most of whom are not experienced programmers. The *BioScript* syntax and semantics were designed to enable scientists to express operations in a manner that closely resembles plain English. To keep the language small, we do not include operations in the language syntax that can automatically be inferred by the compiler and/or execution engine. For example, the compiler can automatically infer implicit fluid transfers for a mix operation. *BioScript* features a semantics that targets pLoC technologies. The syntax and semantics of *BioScript*'s type system are formally described in Section 3.

We begin with a self-contained example to illustrate the expressive capabilities of *BioScript*.

Example: PCR with Droplet Replenishment. Figure 3 presents a *BioScript* program for a DMFB-compatible implementation of the *polymerase chain reaction (PCR)*, used to amplify DNA.[14] PCR involves *thermocycling* (repeatedly heating and then cooling) a droplet containing the DNA mixture undergoing amplification [lines 5–17]. In this implementation, thermocycling may cause excess droplet evaporation. This implementation uses a weight sensor to detect the droplet

Figure 3. PCR with droplet replenishment.[9] It uses the target-specific save instruction.

```
 1  // Initialization omitted. PCRMasterMix is a
 2  // commercially available pre-mixed solution
 3  // used to perform PCR.
 4  PCRMix = mix PCRMasterMix with Template for 1s
 5  repeat 50 times {
 6    heat PCRMix at 95C for 20s
 7    volumeWeight = detect Weight on PCRMix
 8    if (volumeWeight <= 50uL) {
 9      replacement = mix 25uL of PCRMasterMix
10                       with 25uL of Template for 5s
11      heat replacement at 95C for 45s
12      PCRMix = mix PCRMix with replacement for 5s
13    }
14    heat PCRMix at 68C for 30s
15    heat PCRMix at 95C for 45s
16  }
17  heat PCRMix at 68C for 5min
18  save PCRMix
```

volume after each iteration [line 8]; if too much evaporation occurs [line 9], the algorithm injects a new droplet to replenish the sample volume [line 10–11], preheating a template solution [line 12] to ensure that replenishment does not affect the temperature of the DNA.

Type Systems and Safety. The Environmental Protection Agency (EPA) and National Oceanic and Atmospheric Administration (NOAA) have categorized 9800 chemicals into 68 reactivity groups,[7] defined by common physical properties of discrete chemicals. It is known that mixing materials from certain reactivity groups can produce materials from other reactivity groups; for example, mixing acids and bases induces a strong reaction that produces salt and water. *BioScript*'s type system models reactivity groups as types. As a material can belong to multiple reactivity groups, a union type is associated with a material. Using standard reaction corpora, we calculate the type signature of the mix operation, which is fundamental throughout chemistry, as a table of abstract reactions between pairs of types, which results in a union of types.

At the same time, reactions vary in terms of safety. The EPA/NOAA categorization assigns one of three outcomes to the combination of chemicals: *Incompatible*, *Caution*, or *Compatible*. If the union type resulting from a mix operation includes a hazardous type, then the corresponding cell in the table is marked as being unsafe. Any biochemical procedure, or *assay*, specified in *BioScript* is allowed to execute only if it is safe. The signature of the mix operation does not include unsafe abstract reactions, which correspond to unsafe table cells. Therefore, the type system exclusively type-checks mix statements that do not produce hazardous materials. This is fundamental to the soundness of *BioScript*'s type system: it only type-checks assays that do not produce unsafe materials.

BioScript allows, but does not require, type annotations, saving the programmer from the burden of annotating programs with overly complicated union types. The assay specifications presented in Figure 3 do not use type annotations. *BioScript*'s type inference system can automatically infer types. As the EPA/NOAA classification begins with a finite set of material types, type inference can be reduced to efficiently decidable theories. Type inference is sound: if a typing assignment is inferred, it can be used to type-check the assay; it is also complete: if there is a typing assignment with which the assay can be type-checked, the inference will discover it. Otherwise, the assay is rejected and marked as a potential hazard if no typing assignment can be inferred for it. Our experiments show that the type system is expressive enough to reject hazardous assays and accept those that are safe. Proofs for these attributes can be found in Ott et al.[15] and its supplemental material.

3. TYPE SYSTEM

This section presents interesting aspects of the core *BioScript* language. We begin by presenting the simple, yet robust, *BioScript* syntax. Next, we describe the novel aspects of the operational semantics—or mathematical model—describing the runtime execution of assays on pLoC devices (Definition 1). We then provide technical details on how *BioScript*'s type system prevents unsafe operations from occurring. Unsafe operations include the interaction of materials that may cause an explosion or create noxious gasses, as well as access materials that have already been consumed. We explore the syntax, operational semantics, and type system using two statements: variable assignment and mix semantics in great detail. The full *BioScript* language, operational semantics, and type system are described in Ott et al.[15] and its supplemental material.

3.1. Syntax

BioScript's set of instructions is modeled after the ISA discussed in Section 1.1. *BioScript* supports heat and detect instructions but omits move and store instructions, as they are inferred from data-flow analysis. The *BioScript* language is imperative and a statement is a sequence of effectful instructions that involve side effect-free terms. To model state, or memory, we define σ, a mapping of variables to their values. A side effect, in this context, is changing σ—updating the values of the variables. As terms are side effect-free, a term does not alter σ. A term can take one of many forms: a variable, a math operation, a detection of a physical property on a material, or a concrete value.

Unlike terms, instructions are not side effect-free; they alter memory. *BioScript* supports traditional assignment of terms to variables, manipulation of variables, and control-flow constructs.

BioScript utilizes a conservative type system capable of analyzing how chemical interactions work in a cyber-physical context. Mixing chemicals during experiments yields a new chemical, functionally expiring the input chemicals. However, not all of the input chemicals participate in the reaction, and trace amounts of the input chemicals are present in the new chemical. For instance, mixing an acid and base yields salt water. There are still acid and base molecules that have not reacted in the salt water. To model this, *BioScript* employs union types that allow variables to belong to multiple types (see Definition 2). In other words, a variable can store any combination of scalar types in the union type. As usual, the typing environment Γ represents a mapping from variables to their types.

Operational Semantics:
The operational semantics describe how a program is executed as a sequence of computational steps. It is represented as inference rules that define valid steps. Inference rules are comprised of premises and conclusions, whereby all the premises must be met for the conclusion to hold. As shown in Figures 4 and 5, the inference rules represent the premises above the line and conclusion below the line.

Definition 1

3.2. Operational Semantics for Assay Execution

We model execution of *BioScript* assays on a DMFB as an operational semantics. When execution of an instruction occurs, for example, a mix, the model must use the appropriate rules to "step" or handle the change of state. All the premises of a stepping rule must be satisfied. If no rule can step, the program is *stuck* and cannot continue execution.

We highlight two sets of rules that showcase some interesting challenges *BioScript* faces and discuss how they are overcome. We begin with variable assignment. It is syntax: $x := t$ allows a variable x to be assigned some term t. To model execution, we define E-AssignR, E-Assign, and E-Assign′, represented in Figure 4a.

The rule E-AssignR evaluates the right-hand side term, t (if it is not a variable); the rule E-Assign assigns the reduced value to the variable in σ, the store. The rule E-Assign′ transfers a material from the right-hand side variable to the left-hand side variable; preventing aliasing.

In traditional computing, variable assignment is an elementary operation that most computer scientists do not even bother thinking about. However, when modeling assignment in the physical world, things are not so simple. In *BioScript*, the value of a chemical variable is consumed when it is assigned to another variable, restricting variable aliasing. In other words,

Figure 4. a and b depict the operational semantics for only variable assignment and mixing in *BioScript*.

E-AssignR
$$\frac{(\sigma, t) \to t' \qquad t \notin \mathcal{X}}{(\sigma,\ x := t;\ s) \to (\sigma,\ x := t';\ s)}$$

E-Assign
$$(\sigma,\ x := v;\ s) \to (\sigma[x \mapsto v],\ s)$$

E-Assign′
$$\frac{\sigma' = (\sigma \setminus \{x'\})[x \to \sigma(x')]}{(\sigma,\ x := x';\ s) \to (\sigma', s)}$$

(a)

E-MixR
$$\frac{(\sigma, t) \to t'}{(\sigma, x := \textbf{mix}\ x_1\ \textbf{with}\ x_2\ \textbf{for}\ t;\ s) \to (\sigma, x := \textbf{mix}\ x_1\ \textbf{with}\ x_2\ \textbf{for}\ t';\ s)}$$

E-Mix
$$\frac{\begin{array}{c}\sigma(x_1) \in Mat \qquad \sigma(x_2) \in Mat \\ interact(\sigma(x_1),\ \sigma(x_2),\ r) \neq \lightning \\ \sigma' = (\sigma \setminus \{x_1, x_2\})[x \mapsto interact(\sigma(x_1),\ \sigma(x_2),\ r)]\end{array}}{(\sigma, x := \textbf{mix}\ x_1\ \textbf{with}\ x_2\ \textbf{for}\ r;\ s) \to (\sigma',\ s)}$$

(b)

Figure 5. a and b depict the typing rules for only variable assignment and mixing in *BioScript*.

T-Assign-1
$$\frac{x : T \in \Gamma \qquad \Gamma, X \vdash v : T' \qquad T' \subseteq T}{\Gamma, X \vdash x := v, X \cup \{x\}}$$

T-Assign-2
$$\frac{x : T \in \Gamma \qquad \Gamma, X \vdash x' : T' \qquad T' \subseteq T}{\Gamma, X \vdash x := x', X \setminus \{x'\} \cup \{x\}}$$

T-Assign-3
$$\frac{\begin{array}{c}x : T \in \Gamma \qquad t \notin \mathcal{V} \cup \mathcal{X} \\ \Gamma, X \vdash t : T' \qquad T' = \mathbb{R} \vee T' = \mathbb{N} \qquad T' \subseteq T\end{array}}{\Gamma, X \vdash x := t, X \cup \{x\}}$$

(a)

T-Mix
$$\frac{\begin{array}{c}\Gamma, X \vdash x_1 : \cup\overline{Mat_i} \qquad \Gamma, X \vdash x_2 : \cup\overline{Mat_j} \qquad \Gamma, X \vdash t : \mathbb{R} \\ interact\text{-}abs(Mat_i, Mat_j) \subseteq \Gamma(x) \ \text{ for each } i \text{ and } j\end{array}}{\Gamma, X \vdash x := \textbf{mix}\ x_1\ \textbf{with}\ x_2\ \textbf{for}\ t, X \setminus \{x_1, x_2\} \cup \{x\}}$$

(b)

the *BioScript* program $x = mat$; $y = x$; $z = x$ is stuck at the third assignment as the second assignment consumes x. This restriction is necessary for material variables, but can be easily lifted for numeric variables.

Mixing is a frequent activity that chemists and biologists employ in their discipline. *BioScript*'s syntax for mixing is simple and intuitive: $x :=$ mix x_1 with x_2 for t. A mix instruction takes two variables (x_1 and x_2), mixes them for some time t and stores the resulting chemical in the new variable x. To model execution of the mix instruction, we define E-MixR and E-Mix, defined in Figure 4b.

E-MixR first evaluates the time term of a mix instruction, eventually reducing it to a real number, r. After the time term has been reduced, E-Mix is evaluated. E-Mix prescribes that both x_1 and x_2 in σ must be materials. The variables x_1 and x_2 must also be safe to interact; the function *interact* determines safety at run time. *interact* returns the resulting material if mixing is safe; otherwise, *interact* returns $\lightning$—the mixture is unsafe. When a scientist mixes two chemicals together in a flask, the two distinct chemicals no longer exist; to model this, the used variables x_1 and x_2 are removed from σ and the variable x is mapped to the resulting material. The evaluation of a mix instruction is *stuck* if either of the two variables are not material values, any of the variables are already used and removed from the store, or the interaction of the materials is unsafe ($\lightning$).

The full runtime model, detailing all terms and instructions, is available in the supplemental material.

3.3. Type Checking and Inference

Similar to modeling execution, inference rules describe how *BioScript*'s type system type-checks a program. Again, we focus on the interesting typing rules that *BioScript* defines to keep scientists safe while writing and executing assays on DMFB devices.

We begin with typing assignment instructions, defined in Figure 5a. The rule T-Assign-1 types an assignment of a value to a variable and adds the variable to the set of available variables. Rule T-Assign-2 strictly prevents aliasing by consuming the right-hand side while adding the left-hand side variable to the set of available variables. (At the cost of brevity, the rule can be easily relaxed to not remove numeric variables from the available set.) Finally, rule T-Assign-3 addresses typing for numeric terms. It allows assigning numeric terms to variables.

In spite of a scientist's training regarding safe and unsafe chemical interactions, countless incidents occur involving chemical interactions that result in explosions or noxious gasses, causing harm to the laboratory or worse, the scientist. To help prevent incidents, *BioScript* defines the typing rule T-Mix, described in Figure 5b, which helps ensure that no chemical interaction exhibits adverse reactions as well as guaranteeing no chemical is used more than once.

To guarantee safety during a mix instruction, x_1 and x_2 must be a union of material types, that is, $\Gamma, X \vdash x_1 : \cup\overline{Mat_i}$ and $\Gamma, X \vdash x_2 : \cup\overline{Mat_j}$, respectively. Similarly, the time term of the mix instruction must be a real number ($\Gamma, X \vdash t: \mathbb{R}$, which is to say that the value of the term t must be in the set of real numbers).

Union Types:
A typing convention allows a variable to assume a set of types. We differentiate between *scalar types*, denoted by S, and *union types*, denoted by $\cup\overline{S}$; a union type is a set of scalar types. In the context of *BioScript*, scalar types are the material types $Mat_1 \,|..|\, Mat_n$. A union of material types can then be expressed as $\cup\overline{Mat}$.

Definition 2

For a mix instruction to type-check, the interaction of the input materials must be safe. To determine this, we define the function *interact-abs*, which accepts two scalar material $\overline{\text{types}}$ as arguments and returns a union type of materials ($\cup Mat$). The abstract interaction *interact-abs* is conservative with respect to the concrete interaction function: *interact*. If two material values mat_i and mat_j are members of two material types Mat_i and Mat_j, and the concrete interaction of mat_i and mat_j is unsafe, then the abstract interaction of Mat_i and Mat_j is undefined, rendering the program unable to type-check. Otherwise, the result of the concrete interaction is a member of the type resulting from the abstract interaction of Mat_i and Mat_j. If the interaction of all such pairs of materials mat_i and mat_j is safe, then the abstract interaction of Mat_i and Mat_j is safe. A full discussion of how the interact-abs function is used is presented in Section 4.

Finally, the result of the mix is assigned to x, whose type in Γ should be a superset of the resulting material types. In the physical world, mixing chemicals uses those chemicals—they no longer exist. To model this, the materials represented by x_1 and x_2 are consumed and replaced by x in the set of available variables.

We proved that the *BioScript* type system is sound. All type-checked programs are correct, that is, never get stuck during execution; conversely, incorrect programs cannot type-check. As explained for the operational semantics, there is no inference rule for unsafe operations; that is, incorrect programs are stuck. The soundness is proved as tandem progress and preservation lemmas (see Definition 3). The progress lemma states that well-typed programs are not stuck; that is, they can take a step. More precisely, if a statement is typed, then it is either the terminal statement or it can make a step. The preservation lemma states that if a well-typed program steps, the resulting program is also well-typed.

BioScript features a type inference system. Type inference helps the biologists and chemists by lifting the burden of manually annotating assays with union types. The rules for type inference match the corresponding type-checking rules but restate the conditions as constraints. After the type inference system derives the constraints for a program, a satisfying model for the constraints yields types for the variables of the program. We proved that the type inference system infers types for a program if it is typeable. This is proved as a pair of soundness and completeness lemmas for the type inference system. The soundness lemma states that, if the type inference system infers types for a program, then with the inferred types, the type-checking system can type-check the program. The completeness lemma states that if, for a program, there exist types for variables under which the type-checking system can type-check the program, then the type inference system can infer those types.

We provide a full discussion of the above theorems in the supplemental materials for the interested reader.

Progress:
A well-typed program is not stuck: that is, it can take a step.
Preservation:
If a well-typed term takes a step, the resulting term is also well-typed.

Definition 3

4. IMPLEMENTATION

This section describes the underlying implementation details of the *BioScript* language and its type system.

***BioScript*.** The *BioScript* language was implemented as described in Section 2. As DMFBs do not offer external fluidic storage, there is no possibility to implement a stack or heap of substantial size. For these reasons, *BioScript* provides *inline* functions exclusively and does not support recursion; similarly, *BioScript* does not support arrays, even of constant size, as doing so would significantly inhibit portability. We hope to address these issues in greater detail in a future publication. *BioScript* handles variable assignment implicitly, for example, Figure 8d. However, the scientist declares a `manifest` of chemicals that is used throughout the assay ("blood" and "water," for this assay) and the *BioScript* compiler infers the *dispense* and *move* operations.

The Type System. *BioScript*'s type system utilizes static type checking, which runs during compilation. The type system automatically infers types using an abstract interaction function that is a conservative overapproximation of the resulting chemical types of each interaction. The type system $\overline{\text{uses th}}$e 68 EPA/NOAA reactivity groups as the material types Mat_i that together with natural $\mathbb{N}$, and real $\mathbb{R}$ numbers, constitute the set of scalar types S.

We calculate the abstract interaction function interact-abs (defined in Section 3.3) as a table that is indexed by two material types and stores union types. Each reactivity group or type Mat_i comprises a nonempty set of chemicals C_i. Abstract mixing of a pair of material types Mat_i and Mat_j effectively mixes each pair of chemicals (c_i, c_j) in the cross product $C_i \times C_j$. If any interaction is *Incompatible*, the table entry for (Mat_i, Mat_j) is marked as hazardous (or undefined, as modeled in Section 3). Otherwise, if the mix operation yields a new chemical c_k, we use a *ChemAxon*,[4] an industry-stand$\overline{\text{ard co}}$mputational chemistry library to assign a union type $\cup Mat_k$ to c_k, which is added to the union type of the cell for Mat_i and Mat_j. In practice, molecules of c_i and c_j will remain after mixing c_i and c_j, even if a reaction occurs, and the presence of extra molecules at the microliter scale, or smaller, may have a nonnegligible impact on the underlying chemistry or biology. To account for this fact, Mat_i and Mat_j are also added to the cell. As type assignment to concrete chemicals is conservative and we include the input types in the resulting union type, the types in the table represent an overapproximation of the chemicals that can result from concrete interactions.

There may be instances where scientists need to create hazardous reactions, which the type system would correctly reject. In this case, the type system generates all relevant errors and warnings, but allows the programmer to override the type system in order to finish compilation and execute the assay.

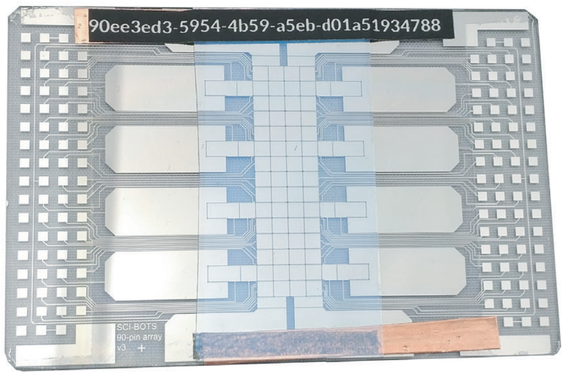
Execution. *BioScript* targets a real-world DMFB platform called DropBot,[8] as shown in Figure 6. Although DropBot features real-time object tracking, it does not, at present, support execution of assays that feature control flow. *BioScript* can produce a DropBot-compatible electrode activation sequence, in the form of a JSON file, to execute on the chip depicted in Figure 6.

5. EVALUATION

The objectives of *BioScript* are to reduce the time and cost of scientific research and to provide a safe execution environment for chemists and biologists with respect to chemical interactions. As noted earlier, *BioScript* is a DSL that enables high-level programming and direct execution of bioassay on pLoCs. These objectives inform our selection of metrics to evaluate *BioScript*.

Language. Compared to other languages, *BioScript* offers an intuitive and readable syntax and a type system. We do *not* claim that *BioScript* offers any performance advantages over other languages; performance primarily depends on the algorithms implemented in the compiler back-end and execution engine, which are compatible, in principle, with any language and front-end. Hence, our evaluation emphasizes qualitative metrics of the language.

Figure 6. A DMFB chip used by DropBot devices.

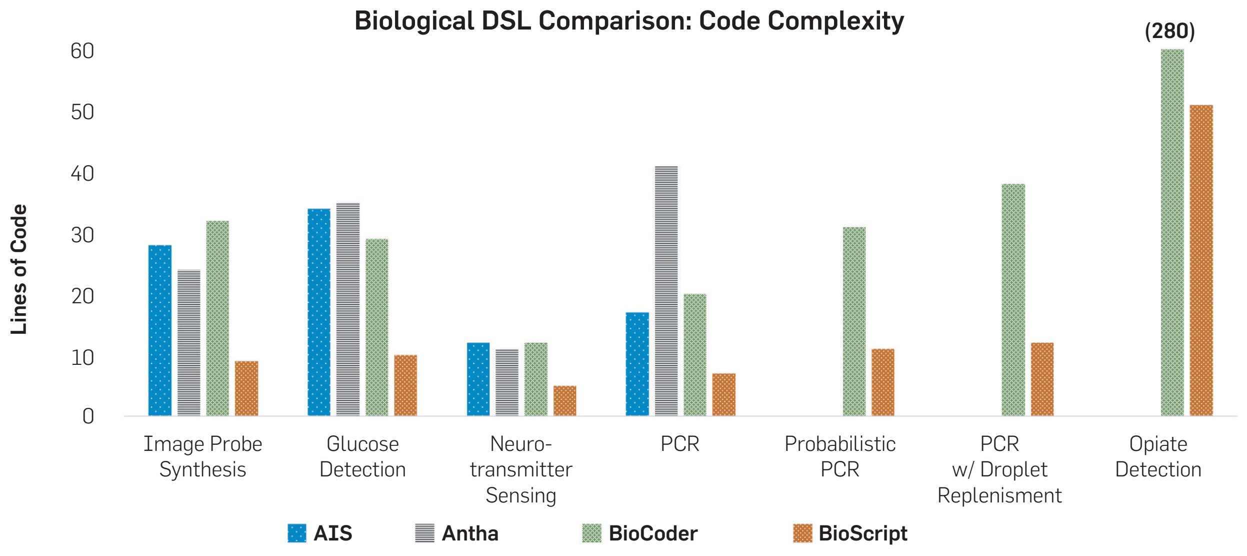


First, we compare *BioScript*'s syntax to three other languages: the *AquaCore Instruction Set (AIS)*, a target-specific assembly-like language;[2] *Antha*, a language for cloud-based laboratory automation;[17] and BioCoder, a C++ library that has been previously specialized for DMFBs.[5] Our comparison uses a set of compact, yet representative, bioassays taken from published literature. As an illustrative example, Figure 8 shows a simple assay (a Mix followed by a Heat instruction) in all four languages; *BioScript*, by far, has the shortest description and is easier to read.

Figure 7 compares the number of lines of code required to specify seven representative bioassays using the four languages; three of the seven assays were not compatible with *AIS* (which is tethered to a specific pLoC[2]) and *Antha* (which is tethered to a cloud laboratory), so we only report four assays for those languages. We do not count empty lines (for spacing/aesthetic purposes) or lines that contain comments. We wrote each assay based on our notion of human readability, which generally meant one statement/operation per line for *AIS*, *BioCoder*, and *Antha*. As shown in Figure 8d, the mixture statement in *BioScript* succinctly encompasses two implicit variable declarations with fluid type and volume information.

Across the four compatible assays, *BioScript* required 68% fewer lines of code than *AIS* and 73% fewer lines of code than *Antha*. Across all seven assays, *BioScript* required 65% fewer lines of code than *BioCoder*, which can target DMFBs, unlike *AIS* and *Antha*. Although these results do not account for subjective experience, we believe that they convey the same basic sentiments as shown in Figure 8: *BioScript* has an intuitive syntax and will be far easier for scientists to learn and use compared to existing languages in this space. Source code for all implementations of the bioassays reported in Figure 7 is included in our supplementary materials.

Figure 7. The number of lines of code to specify image probe synthesis, glucose detection, neurotransmitter sensing, PCR, probabilistic PCR, PCR w/ droplet replacement, and opiate detection in AIS, BioCoder, Antha, and *BioScript*. We were unable to specify the latter three assays in AIS and Antha.



Type System Evaluation. *BioScript*'s type system's main purpose is to prevent inadvertent production of hazardous chemicals. We evaluate its ability to detect hazardous mixing in *BioScript* descriptions of five reported real-world incidents.[1, 3] To the best of our understanding, *BioScript*'s type system is first-of-its-kind, so there are no prior type systems for biochemistry to compare against.

Table 1 summarizes the results of our experiments. The results denoted by the † are real-world situations in which safety precautions were ignored while carrying out experiments. The first three are incidents documented by the *American Industrial Hygiene Association (AIHA)*.[1] *Mustard gas* refers to a documented situation where an individual mixed two common reagents used to clean swimming pools, inadvertently creating mustard gas. *SafetyZone* refers to a documented explosion where a student mixed a sulfuric acid/hydrogen peroxide mixture with acetone[6] (it remains unknown whether this explosion was intentional or accidental). The type system correctly identified the presence of safety hazards in all of these cases.

We also tested the type system on 14 assays that were known to be safe; *BioScript*'s type system successfully inferred types in all of these cases. These assays, listed in Table 1, are currently used in the physical sciences today.

Compilation Time. We compiled the safe and unsafe assays described here, targeting the DropBot platform, which is a 4×15 array (not including I/O reservoirs which reside on the perimeter of the device), assuming the default electrode actuation time of 750 ms. The experiments were run on a 2.7 GHz Intel™ Core i7 processor, 8 GB RAM, machine running macOS™. Construction of the type system's *abstract interaction table* took 31 min running on a 2.53 GHz Intel™ Xeon™ processor, with 24 GB RAM, running CentOS 5.

Table 1 reports the compilation time, constraint solving time, and number of constraints gathered. The unsafe, real-world, assays were correctly identified as unsafe by *BioScript*. On average, each material defined in the benchmarks belonged to 3.015 distinct reactive groups; average benchmark compilation time was 0.0190 s; and the average time spent solving constraints was 1.594 s. We must note that these programs are significantly smaller than typical software programs today.

BioScript assays, along with additional synthetic benchmarks, are made available in the supplemental materials.

Figure 8. Example assay specified using BioCoder (a), Antha (b), AIS (c), and *BioScript* (d). We omit initialization for all examples.

```
b.first_step();
b.measure_fluid(blood, tube);
b.measure_fluid(water, tube);
b.next_step();
b.tap(tube, tenSec);
b.next_step();
b.incubate(tube, 100, tenSec);
b.end_protocol();
```

(a)

```
smpl:= make([]*wtype.LHComponent, 0)
Bld := mixer.SampleForTotalVolume(Blood, BldVol)
smpl = append(smpl, Bld)
Wtr := mixer.Sample(Water, WtrVol)
smpl = append(smpl, Wtr)
rctn := MixInto(OutPlate, "", smpl...)
r1 := Incubate(rctn, mltTemp, InitDenatime, false)
```

(b)

```
input s1, ip1
input s2, ip2
move mixer1, s1;
move mixer1, s2;
mix mixer1, 10;
move heater1, mixer1;
incubate heater1, 100, 10;
```

(c)

```
mixture = mix water with blood for 10s
heat mixture at 100C for 10s
```

(d)

6. CONCLUSION AND FUTURE WORK

BioScript enables scientists to express assays in a comfortable manner, similar in principle to laboratory notebooks. Its type system, which defines the operational semantics of *BioScript*, can provide safety guarantees for chemicals used. *BioScript* is extensible, allowing it to target pLoC compilation and LoC synthesis across multiple technologies. *BioScript* and its software stack pave the way for many life science subdisciplines to increase productivity due to automation and programmability. This paper reports a full system implementation, which can compile and type-check a high-level language program and execute it on the real-world DropBot platform by transmitting commands (electrode actuation sequences) via the DropBot software interface.

Being nascent, *BioScript*'s type system statically type-checks only chemical reactivity groups. Extending the type system, introducing dependent types to account for properties such as temperature, pH, volume, or concentration is a natural next step.

Table 1. Compile time and the number of constraints gathered

Benchmark	Compile time (s)	Type check time (s)	Total types
AIHA 1†	0.012	0.936	70
AIHA 2†	0.012	1.648	68
AIHA 3†	0.014	1.214	17
Broad spectrum opiate	0.011	0.887	11
Ciprofloxacin	0.023	1.722	14
Diazepam	0.024	1.007	14
Dilution	0.014	0.892	9
Fentanyl	0.018	0.900	13
Full morphine	0.048	4.188	19
Glucose detection	0.012	1.633	14
Heroine	0.020	1.553	13
Image probe synthesis	0.015	2.181	13
Morphine	0.018	1.026	13
Mustard gas†	0.015	1.433	83
Oxycodone	0.026	0.959	13
PCR	0.032	3.534	8
Safety zone†	0.013	1.341	76

†Real-world instances that resulted in damages to equipment or personnel that the type system was correctly able to identify as dangerous.

In the long term, this type system could be generalized into a generic type system for cyber-physical systems, transcending even pLoC-based biochemistry. In the future, we hope to extend the *BioScript* language with support for noninlined functions, arrays, SIMD operations, and some notion akin to processes or threads. We view the type system as a starting point for a much deeper foray into formal verification, for example, to ensure that biological media always experience physical properties such as temperature or pH levels within a user-specified range.

Acknowledgments
We would like to thank Philipp Haller for his feedback and insight on elements of this work.

References
1. American Industrial Hygiene Association. 2016. http://bit.ly/2eZtf1m. [Accessed: 2016-11-08].
2. Amin, A.M., Thottethodi, M., Vijaykumar, T.N., Wereley, S., Jacobson, S.C. Aquacore: A programmable architecture for microfluidics. In D.M. Tullsen, and B. Calder, eds. Proceedings of the 34th International Symposium on Computer Architecture (ISCA 2007), June 9–13, 2007, San Diego, California, USA, ACM, 2007. pp. 254–265
3. Blog SPH. Swimming pool chemical incident. 2016. http://bit.ly/2gghGZI. [Accessed: 2016-11-01].
4. ChemAxon. 2016. http://www.chemaxon.com. Marvin was used for characterizing chemical structures, substructures and reactions. Marvin 16.10.3.
5. Curtis, C., Brisk, P. Simulation of feedback-driven PCR assays on a 2d electrowetting array using a domain-specific high-level biological programming language. *Microelectronic Engineering 148*, (2015), 110–116.
6. Dobbs, D.A., Bergman, R.G., Theopold, K.H. Piranha solution explosion. 1990.
7. Environmental Protection Agency & National Oceanic and Atmospheric Administration. 2016. https://cameochemicals.noaa.gov/.
8. Fobel R, Fobel C, Wheeler AR. Dropbot: An open-source digital microfluidic control system with precise control of electrostatic driving force and instantaneous drop velocity measurement. *Appl. Phys. Lett. 19*, 102 (2013).
9. Jebrail, M.J., Renzi, R.F., Sinha, A., Van De Vreugde, J., Gondhalekar, C., Ambriz, C., Meagher, R.J., Branda, S.S. A solvent replenishment solution for managing evaporation of biochemical reactions in air-matrix digital microfluidics devices. *Lab Chip 15*, (2015), 151–158.
10. Lippmann, G. Relations entre les phénomènes électriques et capillaires. Gauthier-Villars. 1875.
11. Luo, Y., Chakrabarty, K., Ho, T. Error recovery in cyberphysical digital microfluidic biochips. *IEEE Trans. CAD Integr. Circuits Sys.* (1), 32 (2013), 59–72.
12. Luo, Y., Chakrabarty, K., Ho, T. Real-time error recovery in cyberphysical digital-microfluidic biochips using a compact dictionary. *IEEE Trans. CAD Integr. Circuits Sys* (12), 32 (2013), 1839–1852.
13. Mugele, F., Baret, J. Electrowetting: From basics to applications. *J. Phys.: Condens. Matter*, 17 (2005), 705–R774.
14. Mullis, K.B., Erlich, H.A., Arnheim, N., Horn, G.T., Saiki, R.K., Scharf, S.J. *Process for amplifying, detecting, and/or-cloning nucleic acid sequences.* US Patent 4,683,195; July 28 1987.
15. Ott, J., Loveless, T., Curtis, C., Lesani, M., Brisk, P. BioScript: Programming safe chemistry on laboratories-on-a-chip. In *Proceedings of OOPSLA '18* (Boston, MA, USA, Nov. 7–9, 2018), Article 124.
16. Pollack, M.G., Shenderov, A.D., Fair, R.B. Electrowetting-based actuation of droplets for integrated microfluidics. *Lab on a Chip* (2), 2 (2002), 96–101.
17. Synthace. Antha-lang, coding biology. 2016. https://www.antha-lang.org. [Accessed: 2016-11-01].
18. Urbanski, J.P., Thies, W., Rhodes, C., Amarasinghe, S., Thorsen, T. Digital microfluidics using soft lithography. *Lab Chip*, 6 (2006), 96–104.
19. Zhao, Y., Xu, T., Chakrabarty, K. Integrated control-path design and error recovery in the synthesis of digital microfluidic lab-on-chip. *JETC* (3), 6 (2010), 11:1–11:28.

Jason Ott, Tyson Loveless, Chris Curtis, Mohsen Lesani, and Philip Brisk ({jott002, tlove004, ccurt002}@ucr.edu, {lesani, philip}@cs.ucr.edu), University of California, Riverside, CA, USA.