Configuration Synthesis for Programmable Analog Devices with Arco

Sara Achour
MIT CSAIL, USA
sachour@csail.mit.edu

Rahul Sarpeshkar
Dartmouth College, MIT RLE, USA
rahul.sarpeshkar@dartmouth.edu

Martin C. Rinard
MIT CSAIL, USA
rinard@csail.mit.edu

Abstract

Programmable analog devices have emerged as a powerful computing substrate for performing complex neuromorphic and cytomorphic computations. We present Arco, a new solver that, given a dynamical system specification in the form of a set of differential equations, generates physically realizable configurations for programmable analog devices that are algebraically equivalent to the specified system. On a set of benchmarks from the biological domain, Arco generates configurations with 35 to 534 connections and 28 to 326 components in 1 to 54 minutes.

Categories and Subject Descriptors D.3.4 [*Processors*]: Compilers; C.1.3 [*Processor Styles*]: Analog Computers

Keywords Compilers, Analog Computing, Languages

1. Introduction

Programmable analog devices have emerged as a powerful computing substrate for performing complex neuromorphic and cytomorphic computations [5, 7, 8, 31, 33, 34, 36, 37, 41]. These systems directly map state variables, transient variables, and parameters in the underlying scientific model to physical aspects of the circuit such as voltage and current.

Programmable analog devices target large biological dynamical systems such as gene–protein and neuron networks. Computations involving these networks are often used for medical dosage optimization, disease prediction, and understanding biological phenomena [23, 35]. The dynamics of biological systems are often oversimplified to make the computations tractable on digital systems, which reduces the accuracy of the model in relation to the corresponding physical system [6, 15, 40]. Studying the modeled dynamics on digital hardware is challenging since these systems are often stiff and therefore prone to numerical instability [13, 29]. In this context, a stiff system is a system that is prone to numerical instability unless the time steps taken are extremely small. Examples of stiff systems include biological systems with both fast and slow dynamics. Analog devices are excellent candidates for these computations:

- **Continuous Time Domain**: Analog devices operate in the continuous time domain, circumventing the time scale issues that often plague stiff dynamical system computations implemented with discretized time steps. Time does not "tick" in intervals as in standard clocked digital systems, but runs continuously and asynchronously.
- **Stochastic Behavior**: The noise present in these analog devices is directly analogous to the noise present in biological systems as they are mapped onto the hardware [33, 37]. This property promotes analog computations that accurately model the underlying biological phenomena.
- **Direct Mapping**: Since the dynamical system is directly mapped to components on the analog device, the maximum dynamical system size is directly proportional to the scale of the analog device.

Currently, there is no compiler support for these devices, so they are programmed manually. The result is a configuration that specifies which analog components to connect and which dynamical system variables and constants to assign to input and output components. The following properties complicate the automatic generation of these configurations:

- **Complex Analog Components**: The components in many analog devices implement complex dynamics and nontrivial relations such as transcendental functions [19, 28, 41]. Because the components are typically engineered for analog efficiency and generality rather than ease of use, there may be a substantial semantic gap between the dynamical system and the analog hardware.
- **Relation Entanglement:** The complex multifunctional components in our target analog hardware devices implement multiple interdependent relations. Mapping a differential equation onto even one such component can there-

PLDI'16, June 13–17, 2016, Santa Barbara, CA, USA
© 2016 ACM. 978-1-4503-4261-2/16/06...$15.00
http://dx.doi.org/10.1145/2908080.2908116

fore activate a cascading sequence of interactions between the component and the remaining dynamical system variables.
- **Resource Constraints:** Analog devices typically provide a finite set of only partially connected components. Any successful mapping of the dynamical system onto the device must live within these constraints.

We present Arco, a solver which, given an analog device specification and dynamical system, synthesizes an algebraically equivalent configuration of the analog device:

- **Algebraic Unification**: Arco fully exploits complex analog components by using algebraic unification to find compact and efficient isomorphisms between the relations in the dynamical system and the provided hardware primitives.
- **Materialized Constants:** The Arco solver materializes new constants that configure and specialize general analog components so that they effectively implement the relations in the specified dynamical system.
- **Relation Entanglement:** Arco successfully manages cascaded relation entanglements by detecting and harmonizing otherwise potentially conflicting dynamical system and hardware variable implementations.
- **Resource Constraints:** Arco effectively works within the hardware resource constraints by tracking and exploiting (potentially partially) used hardware components and using an SMT solver to solve component interconnection problems.

Because Arco works with a hardware specification language that defines the capabilities of the underlying analog device, it can work with the full range of analog devices, including devices that provide powerful new analog primitives. The resulting analog hardware configurations are algebraically equivalent to the specified dynamical systems.

1.1 Contributions

We claim the following contributions:

- **Basic Approach:** We present a new approach for mapping dynamical systems that model complex biological phenomena onto programmable analog devices. Starting with a specification of the dynamical system, our approach automatically synthesizes an algebraically equivalent configuration of the programmable analog device.
- **Analog Circuit Solver**: We present a new analog circuit solver that uses a deductive algebraic approach to solve the analog circuit synthesis problem. Algebraic unification enables Arco to fully exploit the powerful but complex analog components that modern programmable analog devices offer. Arco successfully manages cascading relation entanglements, materializes constants that appropriately configure general analog components, and works within the resource constraints of the programmable analog device.
- **Evaluation**: We evaluate Arco on five biological dynamical systems extracted from an artifact repository for biological models that have been accepted into computational biology conferences [1]. Given a programmable analog hardware description, Arco synthesizes configurations for the models with from 28 to 326 components and 35 to 534 connections in 1 to 54 minutes.

To enable these contributions, Arco (1) works with analog device descriptions written in a novel hardware specification language and (2) accepts dynamical systems written in a novel high level dynamical system specification language. To the best of our knowledge, Arco is the first system to automatically synthesize programmable analog device configurations from dynamical system specifications.

2. Example

We next present an example that demonstrates (1) how to express a dynamical system in the dynamical system specification language, (2) how to define a programmable analog device in the hardware specification language, and (3) the resulting solution generated by the solver.

2.1 Dynamical System: Enzyme-Substrate Reaction

Consider a chemical reaction where an enzyme (E) reacts with a substrate (S) to produce a compound (ES), where the forward reaction rate is catalyzed by catalyst (Q). The formation rate of ES (k_f) is the sum of some nominal rate (k_0) and the amount of the catalyst present multiplied by its efficiency constant k_q, which we assume to be one. We therefore express the reaction rate for producing ES as k_0+Q. The reverse reaction rate is k_r.

$$E + S \rightarrow_{k_0+Q} ES \qquad\qquad ES \rightarrow_{k_r} E + S$$

We may express this reaction as a dynamical system comprised of differential equations, where E_{tot} and S_{tot} are the total amounts of E and S, respectively. The following set of equations specifies the behavior of the dynamical system:

$$E = E_{tot} - ES \qquad\qquad S = S_{tot} - ES$$
$$\partial ES/\partial t = (Q + k_0) \cdot E \cdot S - k_r \cdot ES$$

Figure 1 presents the differential equations written in the Arco dynamical system specification language (DSSL). The specification defines two units, seconds (`s`, unit of time) and molarity (`M`, unit of concentration). The compound Q is expressed as an input quantity with unit of measure `M` (the quantity is measured as a concentration). The compounds E and S are expressed as intermediate (`local`) variables also measured in molarity M. The reaction parameters k_0 and k_r are assigned appropriate values and units. The time variable is `t` measured in seconds (`s`).

The relation statements (`rel`) describe the dynamics of each `local` and `output` variable. These statements implement the set of equations outlined above. The `init` clause in the definition of `ES` sets the initial value of `ES` to `0.423`.

2.2 Hardware Specification

Figure 2 presents the hardware specification for a simple programmable analog device. The specification is inspired by existing and envisioned programmable analog chips that Arco is designed to target [8, 37, 41]. The specification defines microseconds (`us`), milliamps (`mA`), and millivolts (`V`) as units then maps time (`t`) to microseconds, the current property (`I`) to milliamps, and the voltage property (`V`) to millivolts.

The specification defines three components, a current adder (`iadd`), a current multiplier (`igain`) and a Michaelis-Menten component (`mm`) [37]. The current adder has two input ports (`W` and `U`) and one output port (`Y`). The current at the output port is the sum of the currents of the two input ports. The Michaelis-Menten component has five input ports `A0`, `B0`, `Kf`, `Kr`, and `C0` and three output ports `A`, `B`, and `C`. The component simulates a basic enzyme-substrate reaction of the form $A + B \leftrightarrow C$:

$$\mathrm{A_V} = \mathrm{B0_V} - \mathrm{C_V} \qquad \mathrm{B_V} = \mathrm{B0_V} - \mathrm{C_V}$$
$$\partial \mathrm{C_V}/\partial t = \mathrm{Kf_I} \cdot \mathrm{A_V} \cdot \mathrm{B_V} - \mathrm{Kr_I} \cdot \mathrm{C_V} \ \texttt{init}\ \mathrm{C0_V}$$

`inst` statements define how many instances of each component are available — for example, there are two instances of the Michaelis-Menten (`mm`) component. `input V` and `input I` components contain a digital to analog converter (DAC) that converts a digital input to voltage or current, respectively, for import into the analog circuit. `output V` components contain an analog to digital converter (ADC) that samples an analog property, in this case voltage, to produce a sequence of digital values for export to the surrounding digital computing environment.

The specification also defines available connections. The `conn iadd->mm` statement, for example, states that output ports of `iadd` components can be connected to input ports of `mm` components. A `*` specifies either all output (`* ->`) or input (`-> *`) ports of all components.

2.3 Arco Solver

The Arco solver works with a set of goals. Initially the goals are simply the relations in the dynamical system. As the solver operates it maps computations within the relations onto the hardware components, with the dynamical system variables and expressions mapped to properties (voltage, current) of the component ports. In our example the solver starts with the following goal table:

$$S = S_{tot} - ES \qquad E = E_{tot} - ES$$
$$\partial ES/\partial t = (k_0 + Q) \cdot S_{tot} - ES \ \texttt{init}\ 0.423$$

The solver first chooses to map the $S = S_{tot} - ES$ computation onto one of the Michaelis-Menten (mm) hardware components. The initial step is to map the dynamical system variable S onto the hardware variable $\mathrm{A_V}$. Each hardware variable is the combination of a port (here A) and a property

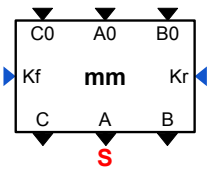
```
type s ; type M
input Q : 1/(s*M)
local E : M; local S : M; output ES : M
param ETOT : M = 0.15; param STOT : M = 0.11
param k0 : 1/(s*M) = 0.36; param kr : 1/s = 0.2
time t : s
rel S = STOT - ES
rel E = ETOT - ES
rel deriv(ES,t)=((k0 + Q)*E*S)-kr*ES init 0.423
```

Figure 1. Enzyme Substrate Reaction Dynamical System

```
type us; type mA; type V
prop I : mA; prop V : mV; time t: us
comp input I
  input X; output Z; rel I(Z) = D(X)
end
comp output V
  input X; output Z; rel D(Z) = V(X)
end
comp input V
  input X; output Z; rel V(Z) = D(X)
end
comp iadd
  input W; input U; output Y; rel I(Y) = I(W)+I(U)
end
comp mm
 input A0; input B0; input C0;input Kf; input Kr
 output A; output B; output C
 rel V(A) = V(A0) - V(C)
 rel V(B) = V(B0) - V(C)
 rel deriv(V(C),t)=I(Kf)*V(A)*V(B)-I(Kr)*V(C)
  init V(C0)
end
schematic
 inst mm : 2; inst iadd : 4
 inst input I : 5; inst output I : 3;
 inst input V : 5; inst output V : 3
 conn iadd -> mm; conn * -> output(V)
 conn input(I) -> *; conn input(V) -> *
end
```

Figure 2. Analog Hardware Specification

such as voltage (V) or current (I). So $\mathrm{A_V}$ is the voltage V property (in black) of the output port A:



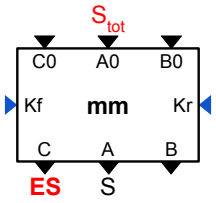
The next step is to map the dynamical system expression $S_{tot} - ES$ onto the mm component. Arco retrieves the hardware relation that defines $\mathrm{A_V}$ and the dynamical system relation that defines S:

$$\mathrm{A_V} = \mathrm{A0_V} - \mathrm{C_V} \qquad S = S_{tot} - ES$$

Arco then algebraically unifies the two relations to obtain the following assignments from hardware variables to dynamical system variables and expressions:

$$\mathrm{A_V} = S \qquad \mathrm{A0_V} = S_{tot} \qquad \mathrm{C_V} = ES$$

In effect, the unification maps variables and expressions from the dynamical system onto the hardware variables of the mm component. In this case, the unification maps S_{tot} onto $A0_V$ (the voltage property of port A0) and ES onto C_V (the voltage property of port C):



Arco now encounters *relation entanglement*. Because of a cyclic dependence between A_V and C_V, it has mapped ES to an output hardware variable of the mm component (specifically C_V). The result is that Arco now has two definitions of ES: one from the mm component (because $C_V = ES$):

$$\partial C_V/\partial t = Kf_I \cdot A_V \cdot B_V - Kr_I \cdot C_V \texttt{ init } C0_V$$

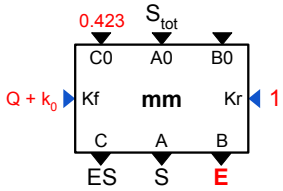
and one from the dynamical system:

$$\partial ES/\partial t = (k_0 + Q) \cdot E \cdot S - ES \texttt{ init } 0.423$$

It is the responsibility of the Arco solver to harmonize these two definitions so that the hardware dynamics of C_V correctly implement the dynamics of ES as defined in the dynamical system (Figure 1). Because Arco has already partially consumed the mm component, it applies the assignments from the unification of $A0_V - C_V$ with $S_{tot} - ES$. The result is the following *mixed-variable relation* (a relation with variables from both the dynamical system specification and the analog hardware specification) that defines how the hardware computes ES:

$$\partial ES/\partial t = Kf_I \cdot S \cdot B_V - Kr_I \cdot ES \texttt{ init } C0_V$$

Now relation entanglement manifests itself as two potentially conflicting definitions of $\partial ES/\partial t$: one from the above mixed-variable relation and one from the original relation in the dynamical system. Arco harmonizes these two definitions by algebraically unifying these two relations. The unification maps dynamical system variables and expressions to the mm component as follows:

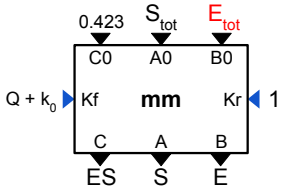


to deliver the following hardware variable assignments: $Kf_I = Q + k_0$, $B_V = E$, $Kr_I = 1$, and $C0_V = 0.423$. Note that Kf_I and Kr_I are current properties (not voltage properties) and are therefore colored blue (not black). Note also that the unification created the *materialized constant* $Kr_I = 1$, in effect specializing the mm component for this specific use.

At this point Arco again encounters relation entanglement: it has mapped E to an output port property, specifically B_V. It has two potentially conflicting definitions of E, one from the hardware and one from the dynamical system. It applies the current set of assignments to the hardware relation that defines B_V ($B_V = B0_V - C_V$) to obtain the following mixed-variable relation that the hardware implements to compute E:

$$E = B0_V - ES$$

Arco retrieves the definition of E from the dynamical system ($E = E_{tot} - ES$) and harmonizes the two relations with another algebraic unification. The unification maps E_{tot} to $B0_V$ as follows:



Arco has now successfully mapped the original goal ($S = S_{tot} - ES$) onto the mm component, including performing any cascaded mappings required to harmonize any generated potential conflicts. Because of these cascaded mappings, Arco has partially mapped all of the following dynamical system relations onto the hardware:

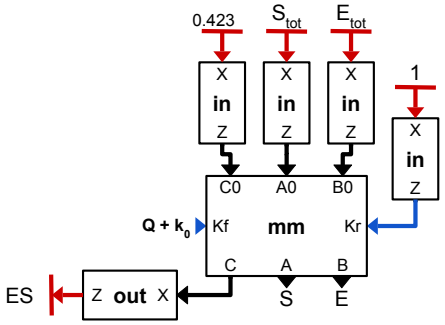
$$S = S_{tot} - ES \qquad E = E_{tot} - ES$$
$$\partial ES/\partial t = (k_0 + Q) \cdot S_{tot} - ES \texttt{ init } 0.423$$

The partial mapping generated a set of subgoals, which consist mostly of assignments of dynamical system variables to (analog) hardware variables. Arco therefore removes the partially mapped original goals and adds the generated subgoals to obtain the following goal table:

$$Kf_I = Q + k_0 \qquad Kr_I = 1 \qquad B0_V = E_{tot}$$
$$C0_V = 0.423 \qquad A0_V = S_{tot} \qquad C_V = ES$$

With the exception of the Kf_I goal, the new goal table contains only assignments of dynamical system variables to hardware variables. Arco assumes that the analog hardware device will be embedded within a larger digital computing framework that will supply dynamical system input variables and values as digital numbers. The framework will receive dynamical system output variables as a sequence of digital values derived by sampling the analog representation of these variables at a specific frequency. The analog hardware device therefore contains *input components*, which use a digital to analog converter to import a digital input value into the analog computation, and *output components*, which use an analog to digital converter to produce the sampled digital sequence of output variable values.

Arco solves all of the goals that assign dynamical system input or output variables or values to analog hardware variables by inserting appropriate input or output components to import or export the relevant values to or from the analog computation:

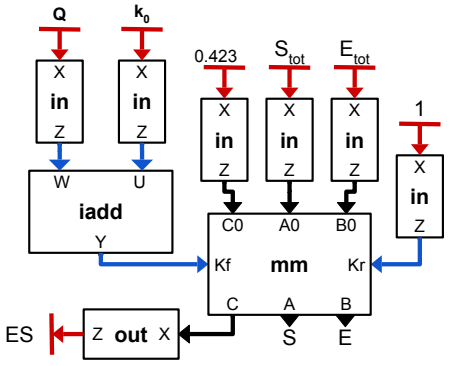


At this point the goal table contains only one goal:

$$\mathrm{Kf_I} = Q + k_0$$

Arco solves this goal by mapping the $Q + k_0$ computation onto an iadd hardware component, then connecting the iadd output port to the Kf input port of the mm component. It then uses appropriate input components to solve the remaining goals, which assign the Q and k_0 variables to iadd hardware input variables. Unlike the previous collection of inputs, the iadd requires current, not voltage, analog inputs. Arco therefore selects input components that produce current, not voltage, values.

At this point Arco has successfully solved all of the goals in the goal table. The result is an analog hardware configuration that correctly implements the specified dynamical system. The configuration specifies the generated inputs, outputs, and analog hardware connections:



ES	$\mapsto$	$\mathrm{outV}_0.X_D$	$\mathrm{mm}_0.C$	$\bullet\!\!-\!\!\bullet$	$\mathrm{outV}_0.X$
0.423	$\mapsto$	$\mathrm{inV}_0.X_D$	$\mathrm{inV}_0.Z$	$\bullet\!\!-\!\!\bullet$	$\mathrm{mm}_0.C0$
S_{tot}	$\mapsto$	$\mathrm{inV}_1.X_D$	$\mathrm{inV}_1.Z$	$\bullet\!\!-\!\!\bullet$	$\mathrm{mm}_0.A0$
E_{tot}	$\mapsto$	$\mathrm{inV}_2.X_D$	$\mathrm{inV}_2.Z$	$\bullet\!\!-\!\!\bullet$	$\mathrm{mm}_0.B0$
1	$\mapsto$	$\mathrm{inI}_0.X_D$	$\mathrm{inI}_0.Z$	$\bullet\!\!-\!\!\bullet$	$\mathrm{mm}_0.Kr$
Q	$\mapsto$	$\mathrm{inI}_1.X_D$	$\mathrm{inI}_1.Z$	$\bullet\!\!-\!\!\bullet$	$\mathrm{iadd}_0.W$
k_0	$\mapsto$	$\mathrm{inI}_2.X_D$	$\mathrm{inI}_2.Z$	$\bullet\!\!-\!\!\bullet$	$\mathrm{iadd}_0.U$
			$\mathrm{iadd}_0.Y$	$\bullet\!\!-\!\!\bullet$	$\mathrm{mm}_0.Kf$

2.4 Arco Search Space and Tableau Configurations

In practice, the Arco solver typically has many potential options at each step as it maps the dynamical system onto the hardware. Not all of these options lead to a successful hardware configuration. Arco therefore deploys a search algorithm to explore a search space of partial hardware configurations. The algorithm uses a variety of optimizations and heuristics to make the search tractable (Section 4.3).

Arco organizes the search with a *tableau* (Section 4.1). This data structure tracks the goals, partially used hardware, and generated partial hardware configurations. The search starts with a tableau configuration containing the analog hardware and the dynamical system. It generates a space of tableau configurations by mapping dynamical system variables, expressions, and relations onto the analog hardware components. The search of the generated tableau space terminates when Arco finds a solved tableau configuration with an empty goal table (Section 4.2).

3. Languages

Arco works with (1) a dynamical system specification that describes the dynamical system to model and (2) a hardware specification that describes the target programmable analog device.

3.1 Dynamical System Specification Language

The dynamical system specification language (DSSL) allows the programmer to describe the dynamical system to simulate. Figure 3 presents the syntax for the DSSL.

Unit: A unit (`type` x) is a named unit of measurement that may be composed into unit expressions. Inputs, outputs, and parameters are all declared with unit expressions that define the units in which they are measured.

Time: The time (`time` x `:` x') is a named time variable x with a unit expression x' in which time is measured.

Variables: A variable (`input|output|local` x `:` U) is a named quantity x with an attached unit expression U.

Parameter: A parameter (`param` $x : U = f$) is set to a fixed floating point value f that does not change over time.

Relations: A relation (`rel` x=E, `rel deriv(`x,x'`)`=E init f) defines the relationship between a `local` or `output` variable and the other variables in the dynamical system.

Type Checking Rules: Appendix A presents type checking rules for the dynamical system specification language.

3.2 Analog Hardware Specification Language

Figure 4 presents the syntax of the hardware specification language. Specifications contain named component definitions, which specify the computational primitives of the programmable analog device. These primitives are defined as a set of relations over properties (such as current and voltage) of the input and output ports of the component.

$f \in \mathbb{F}$ $n \in \mathbb{N}$ $x \in$ Ident

U	$\in$	Unit	::=	U * U | U / U | x | none
E	$\in$	Expr	::=	E+E' | E-E' | -E | E * E' | E / E'
				exp(E) | E ^ E | n | x | f
S	$\in$	Stmt	::=	type | time x : x' | input x : U
				output x : U | local x : U
				param x : U = x | rel x = E
				rel deriv(x,x') = E init f
P	$\in$	Prog	::=	S^*

Figure 3. Dynamical System Specification Language

Units: A unit (type x) is a named unit of measurement for a specific property of the circuit or time. Units relate circuit properties to computational properties.
Time: The time (time x:x') is a named time variable x and the associated units x' in which it is measured.
Properties: A property (prop x:x'), is a named circuit property x, such as current or voltage, and the associated unit x' in which it is measured. The combination of a property x and a port x'', written $x(x'')$, is a hardware variable.
Components: Components (comp x ...end block) are the basic computational primitives of the circuit. Components have input ports (input x), output ports (output x), and parameters (param x:u=f). Stateless relations rel Q=E define the value of an output port property Q as an expression E. Stateful relations rel deriv(Q,x)=E init Q', define the value of output port property Q with a differential equation. Q' provides the initial value.

Arco also supports input components, which provide a digital interface for importing dynamical system values into the analog circuit, and digital output components, which provide a digital interface for exporting dynamical system values from the analog circuit to the surrounding digital computing environment.
Schematic: A schematic..end description block specifies resources R, including the number of instances of each component inst x : n and the potential connections between components inst x : n (at the granularity of components, instances of components, or individual component ports).
Type Checking Rules: Appendix B presents the type checking rules for the hardware specification language.

4. Arco Solver

Dynamical System: The Arco solver works with a dynamical system with input variables $\widehat{i} \in \widehat{I}$, output variables $\widehat{o} \in \widehat{O}$, and local variables $\widehat{l} \in \widehat{L}$. Together these variables are the dynamical system variables $\widehat{v} \in \widehat{V} = \widehat{I} \cup \widehat{O} \cup \widehat{L}$. It also works with sets of dynamical system relations $\widehat{r} \in \widehat{\mathrm{R}}$, where each relation $\widehat{r}$ is of the form $\widehat{r} : \widehat{v} = \widehat{\mathrm{E}}$ or $\widehat{r} : \partial\widehat{v}/\partial t = \widehat{\mathrm{E}}$ init x, $\widehat{v} \in \widehat{L} \cup \widehat{O}$, $x \in \mathbb{R}$, and $\widehat{\mathrm{E}}$ is an expression over $\widehat{V}$ so that $\mathsf{vars}(\widehat{\mathrm{E}}) \subseteq \widehat{V}$.
Hardware: The hardware has digital to analog (DAC) input components $\overline{\mathtt{ic}} \in \overline{\mathtt{IC}}$, analog to digital (ADC) output components $\overline{\mathtt{oc}} \in \overline{\mathtt{OC}}$, and analog computation components $\overline{\mathtt{cc}} \in \overline{\mathtt{CC}}$. Together these components are the hardware components $\overline{\mathtt{c}} \in \overline{\mathtt{C}}$. The hardware specification language enables the designer to specify multiple instances of declared components. Each such instance is a separate component $\overline{\mathtt{c}} \in \overline{\mathtt{C}}$.

$f \in \mathbb{F}$ $n \in \mathbb{N}$ $x \in$ Ident

Q	$\in$	Port-Prop	::=	x (x')
U	$\in$	Unit	::=	U * U | U / U | x | none
E	$\in$	Expr	::=	E +E | E-E | -E | E*E | E / E
				exp(E) | E ^ E | Q | f | n | x
D	$\in$	Defn	::=	input x | output x | param x : U = f
				rel deriv(Q,x) = E init Q'
				rel Q = E
W	$\in$	Wire	::=	* | x | W . x | W [n : n']
R	$\in$	Res	::=	inst x : n | conn W -> W'
Y	$\in$	Comp	::=	D^*
Z	$\in$	Schem	::=	R^*
S	$\in$	Stmt	::=	type x | time x : x' | prop x : x'
				comp x : Y end | schematic : Z end
P	$\in$	Program	::=	S^*

Figure 4. Hardware Specification Language

Components have input ports $\overline{\mathtt{i}} \in \overline{\mathtt{IP}}$ and output ports $\overline{\mathtt{o}} \in \overline{\mathtt{OP}}$. Together these ports are the ports $\overline{\mathtt{p}} \in \overline{\mathtt{P}} = \overline{\mathtt{IP}} \cup \overline{\mathtt{OP}}$. Each port $\overline{\mathtt{p}}$ belongs to a component $\overline{\mathtt{c}}$, denoted $\overline{\mathtt{p}}$ @ $\overline{\mathtt{c}}$. Each port $\overline{\mathtt{p}}$ has properties q $\in$ Q, denoted $\overline{\mathtt{p}}_{\mathrm{q}}$. Common properties q include current and voltage.

The combination of a port and a property is a hardware variable. There are input variables $\overline{\mathtt{i}}_{\mathrm{q}} \in \overline{\mathtt{I}} = \overline{\mathtt{IP}} \times \mathrm{Q}$ and output variables $\overline{\mathtt{o}_{\mathrm{q}}} \in \overline{\mathtt{O}} = \overline{\mathtt{OP}} \times \mathrm{Q}$. Together these variables are the hardware variables $\overline{\mathtt{v}} \in \overline{\mathtt{V}} = \overline{\mathtt{I}} \cup \overline{\mathtt{O}}$. Each variable $\overline{\mathtt{v}} \in \overline{\mathtt{V}}$ is a property q of a port $\overline{\mathtt{p}}$, denoted $\overline{\mathtt{v}} : \overline{\mathtt{p}}_{\mathrm{q}}$.

Each component $\overline{\mathtt{c}} \in \overline{\mathtt{C}}$ has a set of variables $\mathsf{vars}(\overline{\mathtt{c}}) \subseteq \overline{\mathtt{V}}$. Components have disjoint variables so that $\overline{\mathtt{c}}_1 \neq \overline{\mathtt{c}}_2$ implies $\mathsf{vars}(\overline{\mathtt{c}}_1) \cap \mathsf{vars}(\overline{\mathtt{c}}_2) = \emptyset$. Input components $\overline{\mathtt{ic}} \in \overline{\mathtt{IC}}$ have a single digital input variable $\overline{\mathtt{i}}_{\mathrm{d}}$ (the input port $\overline{\mathtt{i}}$ with the digital property d). Output components $\overline{\mathtt{oc}} \in \overline{\mathtt{OC}}$ have a single digital output variable $\overline{\mathtt{o}}_{\mathrm{d}}$ (the output port $\overline{\mathtt{o}}$ with the digital property d).

The hardware implements a set of hardware relations $\overline{\mathtt{r}} \in \overline{\mathtt{R}}$, where each relation $\overline{\mathtt{r}}$ is of the form $\overline{\mathtt{r}} : \overline{\mathtt{v}} = \overline{\mathtt{E}}$ or $\overline{\mathtt{r}} : \partial\overline{\mathtt{v}}/\partial t = \overline{\mathtt{E}}$ init $\overline{\mathtt{v}'}$, where $\overline{\mathtt{v}} \in \overline{\mathtt{O}}$ and $\overline{\mathtt{E}}$ is an expression over $\overline{\mathtt{V}}$ so that $\mathsf{vars}(\overline{\mathtt{E}}) \subseteq \overline{\mathtt{V}}$.

The hardware also has a set of wires $\overline{\mathtt{W}} \subseteq \overline{\mathtt{OP}} \times \overline{\mathtt{IP}}$. Each wire is a configurable connection between an output port and an input port.
Goals: As the solver operates, it maps the dynamical system relations onto the hardware (by mapping variables and expressions from the dynamical system to properties of hardware ports). The solver works with a set of goals $\widetilde{\mathtt{r}} \in \widetilde{\mathrm{R}}$, each of which is derived from an original dynamical system relation $\widehat{r} \in \widehat{\mathrm{R}}$ and each of which captures some remaining unmapped part of $\widehat{r}$. The solver terminates when all of the relations in the dynamical system have been fully mapped onto the hardware and the set of goals is empty.

$$\mathsf{solve}(\overline{\mathrm{R}}, \overline{\mathrm{W}}, \widehat{\mathrm{R}}) = \{\mathrm{Z} \mid \langle \overline{\mathrm{R}}, \emptyset, \overline{\mathrm{W}}, \widehat{\mathrm{R}}, \emptyset\rangle \rightarrow * \langle \overline{\mathrm{R}}', \dot{\mathrm{R}}', \mathrm{W}', \emptyset, \mathrm{Z}\rangle\}$$

$$\textsc{unify}\ \frac{\mathrm{r} \in \overline{\mathrm{R}} \cup \dot{\mathrm{R}} \qquad \widetilde{\mathrm{r}} \in \widetilde{\mathrm{R}} \qquad \mathsf{unify}(\mathrm{r}, \widetilde{\mathrm{r}}, \overline{\mathrm{R}}, \dot{\mathrm{R}}, \widetilde{\mathrm{R}}) = \langle \overline{\mathrm{R}}', \dot{\mathrm{R}}', \widetilde{\mathrm{R}}'\rangle}{\langle \overline{\mathrm{R}}, \dot{\mathrm{R}}, \mathrm{W}, \widetilde{\mathrm{R}}, \mathrm{Z}\rangle \rightarrow \langle \overline{\mathrm{R}}', \dot{\mathrm{R}}', \mathrm{W}, \widetilde{\mathrm{R}}', \mathrm{Z}\rangle}$$

$$\textsc{connect}\ \frac{\widetilde{\mathrm{r}} : \overline{\mathrm{i}}_\mathrm{q} = \overline{\mathrm{o}}_\mathrm{q} \in \widetilde{\mathrm{R}} \qquad \mathrm{w} : \langle \overline{\mathrm{o}}, \overline{\mathrm{i}}\rangle \in \mathrm{W}}{\langle \overline{\mathrm{R}}, \dot{\mathrm{R}}, \mathrm{W}, \widetilde{\mathrm{R}}, \mathrm{Z}\rangle \rightarrow \langle \overline{\mathrm{R}}, \dot{\mathrm{R}}, \mathrm{W} - \{\mathrm{w}\}, \widetilde{\mathrm{R}} - \{\widetilde{\mathrm{r}}\}, \mathrm{Z} \cup \{\overline{\mathrm{o}} \bullet\!\!-\!\!\bullet\, \overline{\mathrm{i}}\}\rangle}$$

$$\textsc{input-var-map}\ \frac{\widetilde{\mathrm{r}} : \overline{\mathrm{i}}_\mathrm{d} = \widehat{i} \in \widetilde{\mathrm{R}} \qquad \widehat{i} \in \widehat{I} \qquad \overline{\mathrm{i}} \ @\ \overline{\mathrm{c}} \qquad \overline{\mathrm{c}} \in \overline{\mathrm{IC}}}{\langle \overline{\mathrm{R}}, \dot{\mathrm{R}}, \mathrm{W}, \widetilde{\mathrm{R}}, \mathrm{Z}\rangle \rightarrow \langle \overline{\mathrm{R}}, \dot{\mathrm{R}}, \mathrm{W}, \widetilde{\mathrm{R}} - \{\widetilde{\mathrm{r}}\}, \mathrm{Z} \cup \{\widehat{i} \mapsto \overline{\mathrm{i}}_\mathrm{d}\}\rangle}$$

$$\textsc{input-val-map}\ \frac{\widetilde{\mathrm{r}} : \overline{\mathrm{i}}_\mathrm{d} = x \in \widetilde{\mathrm{R}} \qquad \overline{\mathrm{i}} \in \overline{\mathrm{I}} \qquad x \in \mathbb{R} \qquad \overline{\mathrm{i}} \ @\ \overline{\mathrm{c}} \qquad \overline{\mathrm{c}} \in \overline{\mathrm{IC}}}{\langle \overline{\mathrm{R}}, \dot{\mathrm{R}}, \mathrm{W}, \widetilde{\mathrm{R}}, \mathrm{Z}\rangle \rightarrow \langle \overline{\mathrm{R}}, \dot{\mathrm{R}}, \mathrm{W}, \widetilde{\mathrm{R}} - \{\widetilde{\mathrm{r}}\}, \mathrm{Z} \cup \{x \mapsto \overline{\mathrm{i}}_\mathrm{d}\}\rangle}$$

$$\textsc{local-var-map}\ \frac{\widetilde{\mathrm{r}} : \widehat{l} \mapsto \overline{\mathrm{o}}_\mathrm{q} \in \widetilde{\mathrm{R}} \qquad \widehat{l} \in \widehat{L}}{\langle \overline{\mathrm{R}}, \dot{\mathrm{R}}, \mathrm{W}, \widetilde{\mathrm{R}}, \mathrm{Z}\rangle \rightarrow \langle \overline{\mathrm{R}}, \dot{\mathrm{R}} \cup \{\overline{\mathrm{o}}_\mathrm{q} = \widehat{l}\}, \mathrm{W}, \widetilde{\mathrm{R}} - \{\widetilde{\mathrm{r}}\}, \mathrm{Z} \cup \{\widehat{l} \mapsto \overline{\mathrm{o}}_\mathrm{q}\}\rangle}$$

$$\textsc{output-var-map}\ \frac{\widetilde{\mathrm{r}} : \widehat{o} \mapsto \overline{\mathrm{o}}'_\mathrm{q} \in \widetilde{\mathrm{R}} \qquad \widehat{o} \in \widehat{O} \qquad \widetilde{\mathrm{r}} : \overline{\mathrm{o}}_\mathrm{d} = \overline{\mathrm{i}}_\mathrm{q} \in \overline{\mathrm{R}} \qquad \overline{\mathrm{o}} \ @\ \overline{\mathrm{c}} \qquad \overline{\mathrm{c}} \in \overline{\mathrm{OC}}}{\langle \overline{\mathrm{R}}, \dot{\mathrm{R}}, \mathrm{W}, \widetilde{\mathrm{R}}, \mathrm{Z}\rangle \rightarrow \langle \overline{\mathrm{R}} - \{\widetilde{\mathrm{r}}\}, \dot{\mathrm{R}} \cup \{\overline{\mathrm{o}}'_\mathrm{q} = \widehat{o}\}, \mathrm{W}, (\widetilde{\mathrm{R}} - \{\widetilde{\mathrm{r}}\}) \cup \{\overline{\mathrm{i}}_\mathrm{q} = \overline{\mathrm{o}}'_\mathrm{q}\}, \mathrm{Z} \cup \{\widehat{o} \mapsto \overline{\mathrm{o}}_\mathrm{d}\}\rangle}$$

Figure 5. Definition of the Tableau Transition Relation →

Each goal $\widetilde{\mathrm{r}} \in \widetilde{\mathrm{R}}$ is a relation $\widetilde{\mathrm{r}}$ of the form $\widetilde{\mathrm{r}} : \widehat{v} = \widehat{\mathrm{E}}$, $\widetilde{\mathrm{r}} : \partial\widehat{v}/\partial t = \widehat{\mathrm{E}}$ `init` x (relations from the dynamical system), $\widetilde{\mathrm{r}} : \overline{\mathrm{i}}_\mathrm{q} = \mathrm{E}$ (a mixed-variable derived relation that may include both dynamical system and hardware variables), or $\widetilde{\mathrm{r}} : \widehat{v} \mapsto \overline{\mathrm{o}}_\mathrm{q}$ (mapping a dynamical system local or output variable to a hardware output variable), where $\widehat{v} \in \widehat{L} \cup \widehat{O}$, $x \in \mathbb{R}$, $\widehat{\mathrm{E}}$ is an expression over $\widehat{V}$ so that $\mathsf{vars}(\widehat{\mathrm{E}}) \subseteq \widehat{V}$, and E is an expression over $\mathrm{V} = \overline{\mathrm{V}} \cup \widehat{V}$ so that $\mathsf{vars}(\mathrm{E}) \subseteq \mathrm{V} = \overline{\mathrm{V}} \cup \widehat{V}$, $\overline{\mathrm{i}}_\mathrm{q} \in \overline{\mathrm{I}}$, and $\overline{\mathrm{o}}_\mathrm{q} \in \overline{\mathrm{O}}$.

Note that the solver may generate partially solved mixed-variable goals that include both dynamical system and hardware variables.

Partially Used Hardware: As the solver operates, it consumes the starting hardware $\overline{\mathrm{R}}$ to produce partially used hardware $\dot{\mathrm{r}} \in \dot{\mathrm{R}}$. Each relation $\dot{\mathrm{r}} \in \dot{\mathrm{R}}$ is of the form $\dot{\mathrm{r}} : \overline{\mathrm{o}}_\mathrm{q} = \mathrm{E}$ or $\dot{\mathrm{r}} : \partial\overline{\mathrm{o}}_\mathrm{q}/\partial t = \mathrm{E}$ `init` $\overline{\mathrm{i}}_{\mathrm{q}'}$, where $\mathsf{vars}(\mathrm{E}) \subseteq \mathrm{V} = \overline{\mathrm{V}} \cup \widehat{V}$. Each $\dot{\mathrm{r}} \in \dot{\mathrm{R}}$ is therefore a partially instantiated mixed-variable relation derived from an original hardware relation.

Hardware Configuration: As it operates, the solver generates a hardware configuration Z. Each $\mathrm{z} \in \mathrm{Z}$ is of the form:

- $\overline{\mathrm{o}} \bullet\!\!-\!\!\bullet\, \overline{\mathrm{i}}$: Connect a wire from hardware output port $\overline{\mathrm{o}}$ to hardware input port $\overline{\mathrm{i}}$.
- $x \mapsto \overline{\mathrm{i}}_\mathrm{d}$: Set a digital input $\overline{\mathrm{i}}_\mathrm{d} \in \overline{\mathrm{I}}$ to a value $x \in \mathbb{R}$.
- $\widehat{i} \mapsto \overline{\mathrm{i}}_\mathrm{d}$: Map a dynamical system input variable $\widehat{i} \in \widehat{I}$ to a digital input $\overline{\mathrm{i}}_\mathrm{d} \in \overline{\mathrm{I}}$
- $\widehat{v} \mapsto \overline{\mathrm{o}}_\mathrm{q}$: Map a dynamical system output or local variable $\widehat{v} \in \widehat{L} \cup \widehat{O}$ to a hardware output port property $\overline{\mathrm{o}}_\mathrm{q} \in \overline{\mathrm{O}}$.

4.1 The Tableau Transition Relation →

We formalize the operation of the solver as a transition relation $\langle \overline{\mathrm{R}}, \dot{\mathrm{R}}, \mathrm{W}, \widetilde{\mathrm{R}}, \mathrm{Z}\rangle \rightarrow \langle \overline{\mathrm{R}}', \dot{\mathrm{R}}', \mathrm{W}', \widetilde{\mathrm{R}}', \mathrm{Z}'\rangle$ on tableau configurations $\langle \overline{\mathrm{R}}, \dot{\mathrm{R}}, \mathrm{W}, \widetilde{\mathrm{R}}, \mathrm{Z}\rangle$, where each transition corresponds to a solver step on the tableau. The solver implements the $\mathsf{solve}(\overline{\mathrm{R}}, \overline{\mathrm{W}}, \widehat{\mathrm{R}})$ function, which returns a set of solutions Z, where each solution is reachable by applying → transition rules to the starting tableau configuration $\langle \overline{\mathrm{R}}, \emptyset, \overline{\mathrm{W}}, \widehat{\mathrm{R}}, \emptyset\rangle$, which contains the initial unused hardware environment $\overline{\mathrm{R}}$, wires $\overline{\mathrm{W}}$, and dynamical system relations $\widehat{\mathrm{R}}$ as goals.

Figure 5 presents the rules that define the solver steps. Rule UNIFY maps a (potentially already partially mapped) relation $\widetilde{\mathrm{r}} \in \widetilde{\mathrm{R}}$ onto a hardware relation $\mathrm{r} \in \overline{\mathrm{R}} \cup \dot{\mathrm{R}}$. Rules CONNECT through OUTPUT-VAR-MAP solve goals that directly configure the hardware:

- CONNECT: The connect rule solves a goal of the form $\overline{\mathrm{i}}_\mathrm{q} = \overline{\mathrm{o}}_\mathrm{q}$ by adding a wire $\overline{\mathrm{o}} \bullet\!\!-\!\!\bullet\, \overline{\mathrm{i}}$ to the configuration Z. The wire connects the output port $\overline{\mathrm{o}}$ to the input port $\overline{\mathrm{i}}$.
- INPUT-VAR-MAP or INPUT-VAL-MAP: The input port mapping rules solve goals of the form $\overline{\mathrm{i}}_\mathrm{q} = x$ or $\overline{\mathrm{i}}_\mathrm{q} = \widehat{i}$. The rules add the mapping $x \mapsto \overline{\mathrm{i}}_\mathrm{q}$ or $\widehat{i} \mapsto \overline{\mathrm{i}}_\mathrm{q}$ to the configuration Z. The mapping binds the value x or dynamical system input $\widehat{i}$ to the hardware variable $\overline{\mathrm{i}}_\mathrm{q}$.
- LOCAL-VAR-MAP: The local variable mapping rule solves goals of the form $\widehat{l} \mapsto \overline{\mathrm{o}}_\mathrm{q}$. It adds the mapping to the configuration Z and adds $\overline{\mathrm{o}}_\mathrm{q} = \widehat{l}$ to $\dot{\mathrm{R}}$. Arco solves resulting future subgoals of the form $\overline{\mathrm{i}}_q = \widehat{l}$ through unification, provided $\overline{\mathrm{i}}_q = \overline{\mathrm{o}}_\mathrm{q}$ holds.
- OUTPUT-VAR-MAP: The output variable mapping rule solves goals of the form $\widehat{o} \mapsto \overline{\mathrm{o}}'_\mathrm{q}$. The rule uses a digital output component $\overline{\mathrm{c}} : \overline{\mathrm{o}}_\mathrm{d} = \overline{\mathrm{i}}_\mathrm{q}$. It also adds the mapping $\widehat{o} \mapsto \overline{\mathrm{o}}_\mathrm{d}$ to X, the relation $\overline{\mathrm{o}}'_\mathrm{q} = \widehat{o}$ to $\dot{\mathrm{R}}$, and the goal $\overline{\mathrm{i}}_\mathrm{q} = \overline{\mathrm{o}}_q$ to $\widetilde{\mathrm{R}}$.

The UNIFY rule uses the $\mathsf{unify}(\mathrm{r}, \widetilde{\mathrm{r}}, \overline{\mathrm{R}}, \dot{\mathrm{R}}, \widetilde{\mathrm{R}})$ function, which unifies r with $\widetilde{\mathrm{r}}$ and resolves any entangled and affected relations. Figure 6 presents the transition rules that define the unify function. Each rule first unifies r and $\widetilde{\mathrm{r}}$ to obtain a set of assignments A. It then applies the assignments A to $\dot{\mathrm{R}}$ (if $\mathrm{r} \in \dot{\mathrm{R}}$) or to $\overline{\mathrm{R}}$ if $(\mathrm{r} \in \overline{\mathrm{R}})$.

The solver steps unify relations r. But the basic Arco unification algorithm $A \in \mathsf{unifyExpr}(E, E', \mathrm{V})$ operates on expressions E and E'. It produces sets of assignments A that unify E and E' so that $\mathsf{sub}(E, A) = \mathsf{sub}(E', A)$, where $\mathsf{sub}(E, A)$ applies the assignments A to the expression E. The rules in Figure 7 lift the unification algorithm from expressions E to relations $\bar{\mathrm{r}}$.

$$\mathsf{unify}(\mathrm{r}, \widetilde{\mathrm{r}}, \overline{\mathrm{R}}, \dot{\mathrm{R}}, \widetilde{\mathrm{R}}) = \{\langle \overline{\mathrm{R}}', \dot{\mathrm{R}}', \widetilde{\mathrm{R}}' \rangle \mid \langle \mathrm{r}, \widetilde{\mathrm{r}}, \overline{\mathrm{R}}, \dot{\mathrm{R}}, \widetilde{\mathrm{R}} \rangle \rightarrow_u \langle \overline{\mathrm{R}}', \dot{\mathrm{R}}', \widetilde{\mathrm{R}}' \rangle\}$$

UNIFY-PARTIAL

$$\frac{\mathrm{r} \in \dot{\mathrm{R}} \qquad \langle A, a, \mathrm{r}', \widetilde{\mathrm{r}}' \rangle \in \mathsf{unifyRelation}(\mathrm{r}, \widetilde{\mathrm{r}}, V) \qquad \langle \dot{\mathrm{R}}', \widetilde{\mathrm{R}}'', D \rangle \in \mathsf{apply}(\dot{\mathrm{R}} - \{\mathrm{r}\}, (\widetilde{\mathrm{R}} \cup \{\widetilde{\mathrm{r}}'\}) - \{\widetilde{\mathrm{r}}\}, A, a)}{\langle \mathrm{r}, \widetilde{\mathrm{r}}, \overline{\mathrm{R}}, \dot{\mathrm{R}}, \widetilde{\mathrm{R}} \rangle \rightarrow_u \langle \overline{\mathrm{R}}, \dot{\mathrm{R}}' \cup D \cup \{\mathrm{r}'\}, \widetilde{\mathrm{R}}'' \rangle}$$

UNIFY-UNUSED

$$\frac{\mathrm{r} \in \overline{\mathrm{R}} \qquad \langle A, a, \mathrm{r}', \widetilde{\mathrm{r}}' \rangle \in \mathsf{unifyRelation}(\mathrm{r}, \widetilde{\mathrm{r}}, V) \qquad \langle \overline{\mathrm{R}}', \widetilde{\mathrm{R}}'', D \rangle \in \mathsf{apply}(\dot{\mathrm{R}} - \{\mathrm{r}\}, (\widetilde{\mathrm{R}} \cup \{\widetilde{\mathrm{r}}'\}) - \{\widetilde{\mathrm{r}}\}, A, a)}{\langle \mathrm{r}, \widetilde{\mathrm{r}}, \overline{\mathrm{R}}, \dot{\mathrm{R}}, \widetilde{\mathrm{R}} \rangle \rightarrow_u \langle \overline{\mathrm{R}}', \dot{\mathrm{R}} \cup D \cup \{\mathrm{r}'\}, \widetilde{\mathrm{R}}'' \rangle}$$

Figure 6. Definition of the Unify Relation $\rightarrow_u$

$$\mathsf{unifyRelation}(\mathrm{r}, \mathrm{r}', \widetilde{\mathrm{r}}, \mathrm{V}) = \{\langle A, a, \mathrm{r}', \widetilde{\mathrm{r}}' \rangle \mid \langle \mathrm{r}, \widetilde{\mathrm{r}}, \mathrm{V} \rangle \rightarrow_f \langle A, a, \mathrm{r}', \widetilde{\mathrm{r}}' \rangle\}$$

UNIFY-HW-REL

$$\frac{\overline{\mathrm{o}}_\mathrm{q} \in \overline{0} \qquad \overline{\mathrm{i}}_{\mathrm{q}'} \in \overline{\mathrm{I}} \qquad \overline{\mathrm{o}} \mathbin{@} \overline{\mathrm{c}} \qquad \mathrm{V} = \mathsf{vars}(\overline{\mathrm{c}}) \qquad A \in \mathsf{unifyExpr}(\mathrm{E}, \mathrm{E}', \mathrm{V})}{\langle \overline{\mathrm{o}}_\mathrm{q} = \mathrm{E}, \overline{\mathrm{i}}_{\mathrm{q}'} = \mathrm{E}', \mathrm{V} \rangle \rightarrow_f \langle A, \overline{\mathrm{o}}_\mathrm{q} = \overline{\mathrm{i}}_{\mathrm{q}'}, \overline{\mathrm{o}}_\mathrm{q} = \mathrm{E}', \overline{\mathrm{i}}_{\mathrm{q}'} = \overline{\mathrm{o}}_\mathrm{q} \rangle}$$

UNIFY-DS-REL

$$\frac{\overline{\mathrm{o}}_\mathrm{q} \in \overline{0} \qquad \widehat{v} \in \widehat{V} \qquad \overline{\mathrm{o}} \mathbin{@} \overline{\mathrm{c}} \qquad \mathrm{V} = \mathsf{vars}(\overline{\mathrm{c}}) \qquad A \in \mathsf{unifyExpr}(\mathrm{E}, \widehat{\mathrm{E}}', \mathrm{V})}{\langle \overline{\mathrm{o}}_\mathrm{q} = \mathrm{E}, \widehat{v} = \widehat{\mathrm{E}}', \mathrm{V} \rangle \rightarrow_f \langle A, \overline{\mathrm{o}}_\mathrm{q} = \widehat{v}, \overline{\mathrm{o}}_\mathrm{q} = \widehat{\mathrm{E}}', \widehat{v} \mapsto \overline{\mathrm{o}}_\mathrm{q} \rangle}$$

UNIFY-DS-DERIV

$$\frac{\overline{\mathrm{o}}_\mathrm{q} \in \overline{0} \qquad \overline{\mathrm{i}}_{\mathrm{q}'} \in \overline{\mathrm{I}} \qquad \widehat{v} \in \widehat{V} \qquad x \in \mathbb{R} \qquad \overline{\mathrm{o}} \mathbin{@} \overline{\mathrm{c}} \qquad \mathrm{V} = \mathsf{vars}(\overline{\mathrm{c}}) \qquad A \in \mathsf{unifyExpr}(\mathrm{E}, \widehat{\mathrm{E}}', \mathrm{V})}{\langle \partial \overline{\mathrm{o}}_\mathrm{q} / \partial t = \mathrm{E}\ \mathtt{init}\ \overline{\mathrm{i}}_{\mathrm{q}'}, \partial \widehat{v} / \partial t = \widehat{\mathrm{E}}'\ \mathtt{init}\ x, \mathrm{V} \rangle \rightarrow_f \langle A \cup \{\overline{\mathrm{i}}_{\mathrm{q}'} = x\}, \overline{\mathrm{o}}_\mathrm{q} = \widehat{v}, \overline{\mathrm{o}}_\mathrm{q} = \widehat{v}, \widehat{v} \mapsto \overline{\mathrm{o}}_\mathrm{q} \rangle}$$

Figure 7. Definition of the Relation Unification Relation $\rightarrow_f$

$$\mathsf{apply}(\mathrm{R}, \widetilde{\mathrm{R}}, A, a) = \{\langle \mathrm{R}', \widetilde{\mathrm{R}}', D' \rangle \mid \langle \mathrm{R}, \widetilde{\mathrm{R}}, A \cup \{a\}, A, \emptyset \rangle \rightarrow_d^* \rightarrow_c \rightarrow_x^* \langle \mathrm{R}', \widetilde{\mathrm{R}}', \emptyset, D' \rangle\}$$

APPLY-INP-ASSIGN

$$\frac{a : \overline{\mathrm{i}}_\mathrm{q} = E \in \widetilde{A} \qquad \overline{\mathrm{i}}_\mathrm{q} \in \overline{\mathrm{I}}}{\langle \mathrm{R}, \widetilde{\mathrm{R}}, A, \widetilde{A}, D \rangle \rightarrow_d \langle \mathrm{R}, \widetilde{\mathrm{R}} \cup \{a\}, A, \widetilde{A} - \{a\}, D \rangle}$$

APPLY-OUT-ASSIGN

$$\frac{a : \overline{\mathrm{o}}_\mathrm{q} = E \in \widetilde{A} \qquad \overline{\mathrm{o}}_\mathrm{q} \in \overline{0} \qquad \mathrm{r} : \overline{\mathrm{o}}_\mathrm{q} = E' \in R \qquad A' \in \mathsf{unifyExpr}(\mathsf{sub}(E', A), E, \mathrm{V})}{\langle \mathrm{R}, \widetilde{\mathrm{R}}, A, \widetilde{A}, D \rangle \rightarrow_d \langle \mathrm{R} - \{\mathrm{r}\}, \widetilde{\mathrm{R}}, A \cup A', (\widetilde{A} \cup A') - \{a\}, D \cup \{a\} \rangle}$$

APPLY-OUT-ASSIGN2

$$\frac{a : \overline{\mathrm{o}}_\mathrm{q} = \widehat{v} \in \widetilde{A} \qquad \overline{\mathrm{o}}_\mathrm{q} \in \overline{0} \qquad \widehat{v} \in \widehat{V} \qquad \mathrm{r} : \overline{\mathrm{o}}_\mathrm{q} = E' \in R \qquad \widetilde{\mathrm{r}} : \widehat{v} = \widehat{\mathrm{E}} \in \widetilde{\mathrm{R}} \qquad \overline{\mathrm{o}}_\mathrm{q} \mathbin{@} \overline{\mathrm{c}} \qquad \mathrm{V} = \mathsf{vars}(\overline{\mathrm{c}}) \qquad A' \in \mathsf{unifyExpr}(\mathsf{sub}(E', A), \widehat{\mathrm{E}}, \mathrm{V})}{\langle \mathrm{R}, \widetilde{\mathrm{R}}, A, \widetilde{A}, D \rangle \rightarrow_d \langle \mathrm{R} - \{\mathrm{r}\}, \widetilde{\mathrm{R}} - \{\widetilde{\mathrm{r}}\}, A \cup A', (\widetilde{A} \cup A') - \{a\}, D \cup \{a, \overline{\mathrm{o}}_\mathrm{q} = \widehat{\mathrm{E}}\} \rangle}$$

CONV

$$\frac{\widetilde{A} = \emptyset}{\langle \mathrm{R}, \widetilde{\mathrm{R}}, A, \widetilde{A}, D \rangle \rightarrow_c \langle \mathrm{R}, \widetilde{\mathrm{R}}, A, D \rangle}$$

XFORM-UNUSED-REL

$$\frac{\mathrm{r} : \overline{\mathrm{o}}_\mathrm{q} = E \in \mathrm{R} \qquad \mathsf{vars}(E) \cap \mathsf{dom}(A) \neq \emptyset}{\langle \mathrm{R}, \widetilde{\mathrm{R}}, A, D \rangle \rightarrow_r \langle \mathrm{R} - \{\mathrm{r}\}, \widetilde{\mathrm{R}}, A, D \cup \{\overline{\mathrm{o}}_\mathrm{q} = \mathsf{sub}(E, A)\} \rangle}$$

APPLY-UNUSED-DERIV

$$\frac{\mathrm{r} : \partial \overline{\mathrm{o}}_\mathrm{q} / \partial t = E\ \mathtt{init}\ \overline{\mathrm{i}}_\mathrm{q} \in \mathrm{R} \qquad \mathsf{vars}(E) \cap \mathsf{dom}(A) \neq \emptyset \qquad \overline{\mathrm{i}}_\mathrm{q} \notin \mathsf{dom}(A)}{\langle \mathrm{R}, \widetilde{\mathrm{R}}, A, D \rangle \rightarrow_r \langle \mathrm{R} - \{\mathrm{r}\}, \widetilde{\mathrm{R}}, A, D \cup \{\partial \overline{\mathrm{o}}_\mathrm{q} / \partial t = \mathsf{sub}(E, A)\ \mathtt{init}\ \overline{\mathrm{i}}_\mathrm{q}\} \rangle}$$

APPLY-UNUSED-DONE

$$\frac{\{r \mid \mathrm{r} \in \mathrm{R} \wedge \mathsf{vars}(\mathrm{r}) \cap \mathsf{dom}(A) \neq \emptyset\} = \emptyset}{\langle \mathrm{R}, \widetilde{\mathrm{R}}, A, D \rangle \rightarrow_r \langle \mathrm{R}, \widetilde{\mathrm{R}}, \emptyset, D \rangle}$$

Figure 8. Definition of the Detangling Relation $\rightarrow_d$ and the Resolving Relation $\rightarrow_r$

Relation Entanglement: Unifying a partially used ($\mathrm{r} \in \dot{\mathrm{R}}$) or unused ($\mathrm{r} \in \overline{\mathrm{R}}$) hardware relation with some goal $\widetilde{\mathrm{r}} \in \widetilde{\mathrm{R}}$ (where the variables $v \in V$ are the hardware variables in $\overline{\mathrm{c}}$: $\mathrm{r} \mathbin{@} \overline{\mathrm{c}}$) may produce assignments which are entangled with other relations that belong to $\overline{\mathrm{c}}$ in R. A hardware relation is *entangled* with an assignment $a : \overline{\mathrm{o}}_\mathrm{q} = E$ if the relation defines $\overline{\mathrm{o}}_\mathrm{q}$ (for example, $\overline{\mathrm{o}}_\mathrm{q} = E'$). A hardware relation is *affected* by a if $\overline{\mathrm{o}}_\mathrm{q}$ is used (but not defined) by the relation (for example, $\overline{\mathrm{o}}_\mathrm{q} = E$ where $\overline{\mathrm{o}}_\mathrm{q} \in \mathsf{vars}(E)$).

Detangling Relations: Figure 8 presents the transition relations that apply a set of assignments A to the tableau. The rules (1) detangle any relations entangled with A ($\rightarrow_d$) and (2) move any relations affected by A out of R ($\rightarrow_r$). *Harmonization* detangles entangled relations. Given an assignment $\overline{\mathrm{o}}_\mathrm{q} = E$ and a relation $\overline{\mathrm{o}}_\mathrm{q} = E'$, Arco first applies the assignments A to E' to create the specialized expression $E'' = \mathsf{sub}(E', A)$. It unifies expressions E and E'' (assigning only variables V), then incorporates the resulting set of assignments A' (chosen from the space of possible assignments) into A.

The $\rightarrow_d$ relation detangles $r \in \mathrm{R}$ from A by incrementally applying assignments $a \in \widetilde{\mathrm{A}}$ to the tableau and detangling any entangled relations as needed, where $\widetilde{\mathrm{A}}$ are the remaining assignments. The rules move detangled relations into D:

- APPLY-INP-ASSIGN: Assignments of the form $\overline{\mathrm{i}}_\mathrm{q} = E$ create no entanglements. The rule therefore moves such assignments directly into the goal set $\widetilde{\mathrm{R}}$.
- APPLY-OUT-ASSIGN: The rule harmonizes entangled assignments of the form $a : \overline{\mathrm{o}}_\mathrm{q} = E$ where $r : \overline{\mathrm{o}}_\mathrm{q} = E' \in \mathrm{R}$. It then adds the resulting set of assignments A' to $\widetilde{\mathrm{A}}$ and

adds the fully specialized relation $\overline{o}_q = \mathsf{sub}(E', A \cup A') = E$ to D. Because the harmonization satisfies assignment a, the rule does not add a to the goal set $\widetilde{R}$.

- APPLY-OUT2-ASSIGN: The rule harmonizes entangled assignments of the form $a : \overline{o}_q = \widehat{v}$ where there is an entangled $r : \overline{o}_q = E' \in R$, and a definition of $\widehat{v}$ of the form $\widetilde{r} : \widehat{v} = \widehat{E} \in \widetilde{R}$. The rule harmonizes $\widehat{E}$ and E' (with V). The rule makes the same changes to $\widetilde{A}$, A, and D as APPLY-OUT-ASSIGN, but also removes the now–mapped relation $\widetilde{r}$ from the goal set $\widetilde{R}$.

Resolving Affected Relations: The $\rightarrow_r$ relation resolves any affected relations $r \in R$ after all potentially entangled relations have been resolved (when $\widetilde{A} = \emptyset$). Each $\rightarrow_r$ rule finds a relation $r \in R$ affected by the assignments A. It then applies the assignments A to r to create a specialized relation which it adds to D. It then removes r from R. It applies these transitions until no affected relations are left.

4.2 The Search Algorithm

Figure 9 presents the Arco search algorithm. The algorithm explores the search space defined by the tableau configuration transition relation $\rightarrow$. It maintains a set of tableau configurations F at the frontier of the explored space. At each step, the algorithm chooses a tableau configuration $t : \langle \overline{R}', \dot{R}, \overline{W}', \widetilde{R}', Z \rangle \in F$ to explore. If all of the goals in t have been solved (i.e., $\widetilde{R} = \emptyset$), the algorithm returns the generated configuration Z. Otherwise, it selects a subset T of the set of tableau configurations directly reachable from t under $\rightarrow$ and replaces t in F with T. The decisions that Arco makes when it chooses a tableau configuration t (line 3) and selects a subset T to explore (line 5) have a significant impact on the effectiveness of the search algorithm.

Choosing t: At line 3, Arco applies a heuristic designed to find the tableau configuration t that is closest to being solved. This heuristic scores each tableau configuration based on the number and complexity of the remaining nontrivial goals (goals solved only by rule UNIFY in Figure 5). It chooses the configuration t with the smallest score (i.e., fewest and simplest remaining nontrivial goals).

Selecting T: Arco first selects a goal $\widetilde{r} \in \widetilde{R}$, prioritizing trivial goals (solved by rules CONNECT through OUTPUT-VAR-MAP in Figure 5) over nontrivial goals (solved by rule UNIFY in Figure 5). It then computes the set of tableau configurations generated by applying the appropriate rule from Figure 5 to $\widetilde{r}$ to obtain T. If any of the trivial goals are unsolvable, Arco sets $T = \emptyset$, effectively pruning the entire search subspace rooted at t. The rationale is that the unsolvable trivial goal ensures that the pruned subspace contains no solved tableau configurations.

The UNIFY rule from Figure 5 may generate a large number of tableau configurations. To maintain the tractability of the search algorithm, the current Arco implementation discards generated tableau configurations so that T contains at most five tableau configurations from each combination of $\overline{r}$ and $\widetilde{r}$ in the UNIFY rule.

```
1  search(R̂₀, R̄₀, W̄₀)
2    F = {⟨R̄₀, ∅, W̄₀, R̃₀, ∅⟩}
3    while choose t : ⟨R̄, Ṙ, W̄, R̃, Z⟩ ∈ F
4      if R̃ = ∅ return Z
5      select T ⊆ {t' | t → t'}
6      F = (F − {t}) ∪ T
7    end
8    return no solution
```

Figure 9. Arco Search Algorithm

4.3 Search Optimizations

We next describe several optimizations that significantly improve the performance of the Arco search algorithm.

Component Aggregation: Our target programmable analog hardware platforms typically contain multiple instances of a given component. Arco reduces the size of the search space by materializing component instances on demand to unify the chosen goal against at most one unused component instance (as opposed to unifying against all available unused component instances).

Abstract and Concrete Hardware Configurations: Because of component aggregation, the Arco tableau search algorithm does not distinguish between different instances of the same component. But the hardware may enforce instance-specific wiring constraints so that different component instances are not interchangable. Arco therefore implements a staged synthesis algorithm. The tableau solver first synthesizes an *abstract hardware configuration* that treats all component instances uniformly. A *concretization algorithm* then maps each abstract component instance to a concrete component instance. The result is a *concrete hardware configuration*. Arco formulates the concretization problem as an SMT problem whose solution maps each abstract component instance to a corresponding concrete component instance. The SMT problem has two kinds of clauses:

- **Instance Assignment Clauses**: These clauses ensure that exactly one abstract component maps to exactly one physical component.
- **Connection Clauses:** These clauses ensure that all of the connections in the abstract hardware configuration map to realizable hardware connections.

If the generated SMT problem is unsatisfiable, then there is no physical configuration of the analog hardware that implements the abstract hardware configuration. In this case the Arco search algorithm removes the corresponding tableau configuration from the tableau frontier. The tableau exploration algorithm continues on to find new abstract hardware configurations.

Partial Configuration Caching: Arco caches generated partial configurations for the different dynamical system relations $\widehat{r} \in \widehat{R}$. It reuses the cached partial configurations whenever it subsequently encounters the relation again in another region of the search space.

Benchmark	Parameters	Functions	Differential Equations	Description
menten	3	0	4	Michaelis-Menten equation reaction[30]
gentoggle	9	3	2	genetic toggle switch in E.coli [18]
repri	7	3	6	synthetic oscillatory network of transcriptional regulators [11]
osc	16	16	9	circadian oscillation utilizing activator / repressor [39]
apop	87	48	27	protein stress response [14]

Table 1. Benchmark Characteristics

Search Tree Data Structure: Instead of fully storing each tableau configuration in the frontier F, Arco maintains (1) a full representation of the current tableau configuration and (2) a delta-based representation of the explored tableau search space. Arco navigates the space by applying and unapplying deltas to the current tableau configuration. This optimization significantly reduces the space required to explore the search space.

4.4 Implementation Details

Arco is implemented in OCaml. Arco includes a collection of utility libraries, including OCaml bindings for `sympy` [22], a python library for algebraic simplification, and OCaml bindings for the `Z3` SMT solver [9]. It also includes implementations of several custom data structures and convenience routines.

4.5 Design Decisions

We decided to use algebraic unification rather than pattern-based unification because the analog circuit synthesis problem requires more sophisticated reasoning than pattern-based unification systems can deliver. We found that synthesizing configurations for programmable analog devices often hinges on the solver's ability to apply algebraic reasoning to rewrite algebraic expressions during unification.

We found algebraic reasoning to be particularly effective at materializing the new constants required to appropriately configure general analog components and at algebraically transforming dynamical system computations to exploit efficient analog components with complex interfaces. This kind of algebraic reasoning is beyond the capabilities of pattern-based unification and other methods that pattern match expression trees. For this reason, we used a computer algebra system that came with a corpus of axioms and was already able to reason effectively about algebraic expressions.

We chose to build a deductive solver instead of encoding the synthesis problem as an SMT problem because SMT techniques are not well suited for reasoning about continuous, transcendental, and nonlinear functions. We found that attempting to encode algebraic reasoning directly in SMT quickly became intractable. Utilizing a deductive technique allowed the solver to synthesize solutions in a more focused manner than an exhaustive search over valid configurations. We use an SMT solver for more appropriate subproblems such as the concretization of abstract configurations.

We implemented a specialized procedure that performs relation detanglement and affected relation resolution after observing that the initial relation unification often produced potential conflicts between the definitions of dynamical system variables and corresponding entangled analog hardware computations. Immediate harmonization reduces the size of the search space and allows Arco to use simpler tableau transitions.

5. Experimental Results

We present experimental results for Arco on a set of benchmarks of varying size and complexity, given a hardware model with components commonly found in programmable analog devices that target biological computations. The benchmarks are a selection of published artifacts from well-cited computational biology papers [1]. The benchmarks were automatically converted into the dynamical system specification language from the provided Octave files. We use the following benchmarks (Table 1):

- **Michaelis-Menten**: The Michaelis-Menten system models a reaction of the form: $E + S \rightarrow ES \rightarrow P$ [30]. The corresponding dynamical system has 9 parameters, 2 inputs, 3 functions, and 2 differential equations.
- **Genetic Toggle Switch for E. coli**: The genetic toggle switch is a bistable gene-regulatory network found in E. coli. It consists of two repressible promoters [18]. The corresponding dynamical system has 3 parameters, 1 input, and 4 differential equations.
- **Transcriptional Reprissilator**: The LacI-tetR-Cl transcriptional reprissilator system is an oscillating network composed of genes that are not involved in maintaining a biological clock [11]. The corresponding dynamical system has 7 parameters, 3 functions, and 2 differential equations.
- **Circadian Oscillator**: The circadian oscillator system is a minimal model for circadian oscillations based on a mutually interacting activator and repressor [39]. The corresponding dynamical system has 16 parameters, 16 functions, and 9 differential equations.
- **Cell Apoptosis Pathway**: The cell apoptosis pathway models the mechanism by which unfolded protein stress response controls the decision mechanism for recovery, adaptation, and apoptosis [14]. The corresponding dynamical system has 87 parameters, 48 functions, and 27 differential equations.

Component	Quantity	Description	Hardware	Relation
iin	25	current input	digital to analog converter	$Z_I = X_D$
vin	125	voltage input	digital to analog converter	$Z_V = X_D$
iout	10	current output	analog to digital converter	$Z_D = X_I$
vout	75	voltage output	analog to digital converter	$Z_D = X_V$
vgain	40	voltage gain	summing amplifier	$O_V = (X_V \cdot Z_V \cdot 0.04)/Y_V$
iadd	30	current adder	wire join	$O_I = A_I + B_I + C_I + D_I$
vadd	35	voltage adder	summing amplifier and capacitor	$\partial O2_V/\partial t = 0.1(A_V + B_V - C_V - D_V \cdot O2_V)$ $O1_V = 0.1(A_V + B_V - C_V - D_V)$
vtoi	30	voltage to current converter	operational amplifier	$O_I = X_V/K_V$
itov	30	current to voltage converter	operational amplifier	$O_I = K_V \cdot X_I$
ihill	8	hill function for activation/repression	logic circuit	$S_I = M_V(S_I/K_I)^{nv}/((S_I/K_I)^{nv} + 1)$ $R_I = M_V/((S_I/K_I)^{nv} + 1)$
igenebind	8	gene binding	logic circuit	$O_I = M_I/(1 + K_I \cdot T_I)$
switch	15	genetic switch	logic circuit	$O_I = M_I/(S_I/K_I + 1)^{nv}$
mm	2	Michaelis-Menten dynamics	logic circuit	$X_V = X0_V - XY_V$ $Y_V = Y0_V - XY_V$ $\partial XY_V/\partial t = K_I \cdot X_V \cdot Y_V - R_I \cdot XY_V$

Table 2. Utilized Analog Hardware Components and Relations [7, 8, 33, 37, 41].

Source Component	Target Components
vgain	outv vadd vtoi mm itov vgain
iadd	iadd igenebind iout ihill switch genebind itov
vin	vtoi itov ihill vadd vgain switch mm
vadd	outv vtoi vgain vadd
vtoi	iadd ihill switch iout
itov	outv vadd vgain switch
iin	itov ihill igenebind switch iadd mm
ihill	iout itov iadd igenebind
igenebind	iout itov ihill iadd
switch	itov iadd iout itov
mm	outv itov iadd vgain mm

Table 3. Analog Component Connections

5.1 Programmable Analog Device

The specified analog device utilizes mixed voltage and current analog building blocks commonly used for biological network computations [7, 8, 33, 37, 41]. Table 2 describes the input-output relations of the circuit components, the physical analog circuits that implement these relations, and the specified quantity of each kind of component. Table 3 presents the hardware connections that the device supports.

5.2 Quantitative Analysis of Generated Circuits

Table 4 presents, for each benchmark, statistics about the synthesized hardware configuration and the components that the solver uses. Overall, the configurations make good use of the more complex, specialized components. The complexity of the generated circuit generally increases with the complexity of the dynamical system. All of the circuits would be difficult to manually generate.

The number of utilized output components typically matches the number of dynamical system output variables. But because the input components import constants as well as dynamical system input variables into the analog computation, the number of utilized input components is less predictable. The Arco algebraic unification can easily generate new constants, either by (1) evaluating constant subexpressions that arise as the unification maps the dynamical system onto the analog components (we call such constants *derived constants*), (2) performing unifications that split a single dynamical system constant into multiple constants via algebraic reasoning (we also call these constants derived constants), or (3) materializing new constants as necessary during expression unification to specialize general analog components so that they correctly implement mapped relations from the dynamical system.

Derived constants often adapt dynamical system constants to complex analog component interfaces. Materialized constants often configure general analog components for more specialized use in the synthesized circuit by setting some of their inputs to constant values. All of our benchmarks have both derived and materialized constants. For the larger benchmarks, there are many more derived and materialized constants than constants that appear directly in the dynamical system. We attribute the presence of these derived and materialized constants to (1) the ability of the Arco algebraic unification algorithm to generate these constants as necessary and (2) the fact that many efficient analog building blocks provide complex interfaces that require nontrivial derived and materialized constants for their successful use.

5.3 Materialized and Derived Constants

We next present several examples of materialized (blue) and derived (red) constants in the analog circuits that Arco synthesizes. Black constants are fixed values that are built into the dynamics of the hardware components. These examples highlight how Arco uses algebraic unification to effectively bridge the semantic gap between the dynamical system and the optimized analog hardware components. All of the examples are beyond the reach of standard pattern-based unification systems.

Benchmark	Connections	Inputs	Outputs	vgain	vadd	mm	vtoi	itov	iadd	switch	ihill	igenebind	Components
menten	35	15	4	2	4	0	0	3	0	0	0	0	9
gtoggle	33	20	5	2	2	0	0	0	0	1	2	0	7
repri	78	29	9	3	6	0	0	3	3	0	3	0	18
osc	146	60	25	16	8	2	1	10	1	2	0	0	40
apop	534	140	39	28	27	2	13	30	27	15	4	1	147

Table 4. Utilized Component Statistics

repri: 3.01029995664 · ES with vgain: For the repri benchmark, Arco wires the output of a `vgain` component into a voltage adder `vadd` to implement 3.01029995664 · LacL as:

$$0.1 \cdot \frac{1 \cdot 0.04}{0.00132877123795} \cdot \text{LacL}$$

where 0.1 comes from the `vadd` relation and 0.04 comes from the `vgain` relation (Table 2). The Arco solver materializes 1 and derives 0.00132877123795 to configure the components to perform the correct multiplication.

apop: PERK^{-4} with switch: For the apop benchmark, Arco wires the output of an `iadd` component into a `switch` component to implement PERK^{-4} as:

$$\frac{1}{[(\text{PERK} + 0 - 1 - 0)/1 + 1]^4}$$

where 0 and 1 are materialized constants and 4 is a derived constant. Together, these constants specialize the `iadd` and `switch` components to perform the proper exponentiation. The `iadd` component implements the PERK + 0 − 1 − 0 subterm. Here Arco synthesizes a multicomponent implementation of a single exponentiation operation.

apop: 0.05 · ATF with switch: Arco wires the output of a `switch` component into a current adder `iadd` to implement 0.05 · ATF as:

$$\frac{1}{(1/-1.05263157895 + 1)^{-1}} \cdot \text{ATF}$$

To compute the term with these components, Arco decomposes the parameter 0.05 into the materialized constants 1 and −1 and the derived constant −1.05263157895.

osc: 0.1 · MA with switch: For the osc benchmark, Arco wires the output of a `switch` component into an output component to implement 0.1 · MA as:

$$\frac{1}{(9/1 + 1)^{-1}} \cdot \text{MA}$$

Here Arco decomposes the parameter 0.1 into the materialized constants 1 and −1 and the derived constant 9.

mmrxn: 0.21 · ES with vgain: For the mmrxn benchmark, Arco wires the output of a `vgain` component into a `vadd` component to implement 0.21 · ES as:

$$\frac{1 \cdot 0.04 \cdot 0.1}{0.0190476190476} \cdot \text{ES}$$

Here Arco materializes 1 and derives 0.0190476190476.

Benchmark	Number of Equations	Synthesis Time
menten	4	0m58.216s
gtoggle	5	1m11.002s
repri	9	3m34.984s
osc	25	13m13.876s
apop	75	53m41.902s

Table 5. Circuit Synthesis Times

osc: ∂MA/∂t = −MA with mm: Arco uses a Michaelis-Menten (`mm`) component (Table 2) with materialized constants $K_I = 0$ and $R_I = 1$. These constants eliminate the term $K_I \cdot X_V \cdot Y_V$ in the `mm` component. Arco leaves X_V and Y_V unassigned since the term disappears when $K_I = 0$. The resulting analog circuit implements $\partial MA/\partial t$ as:

$$\partial MA/\partial t = 0 * X_V * Y_V - 1 * MA$$

This specialized configuration enables Arco to use the multifunctional Michaelis-Menten component without relation entanglement.

5.4 Circuit Synthesis Times

Table 5 presents the time required to synthesize configurations for the five benchmarks. We performed a linear regression on these times and found that the running time (in minutes) = 0.753 × the number of relations in the dynamical system + 3.62. The R squared value for the regression is 0.996573, indicating that, with high confidence, the time varies linearly with the number of equations.

Table 5 presents the running times for synthesizing configurations for the five benchmarks. We performed a linear regression on these runtimes and found that the running time (in minutes) = 0.753 × the number of relations in the dynamical system + 3.62. The R squared value for the regression is 0.996573, indicating with high confidence the time varies linearly with the number of equations.

6. Related Work

Much early work in analog computing focused on manually building analog circuits from simple components with the goal of performing numerical computations for simple dynamical systems [10, 27, 32, 38]. This work was based on manual development of the mapping between the problem and the analog circuit. The research presented in this paper, in contrast, focuses on automatic synthesis of analog circuits that use complex components to implement dynamical systems specified in our specification language.

Analog computation has been experiencing a renaissance in the hardware community — modern day programmable analog devices provide complex, domain specific primitives whose behavior is analogous to their physical counterparts [5, 7, 8, 31, 33, 34, 37, 41]. These systems accurately model the dynamical systems they target and minimize discretization errors. The hardware is typically programmed directly at a low level with little or no automation. The techniques presented in this paper, in contrast, automatically synthesize analog circuits from a dynamical system specification.

Recent work in synthesis for analog computing has focused on transistor-level techniques to aid designers in designing specialized analog logic circuits [4, 20, 26]. Our synthesis techniques, in contrast, use the complex multifunctional analog building blocks that these techniques provide to synthesize analog implementations of specified dynamical systems.

Researchers have developed techniques that use analog neural network accelerators to improve the performance of applications written in standard imperative programming languages [16, 36]. These techniques approximately map subcomputations onto trained analog neural network accelerators. Arco, in contrast, maps complete computations onto analog hardware without approximation — the resulting analog configurations are algebraically identical to the specified dynamical system. The goal is not to accelerate an approximable subcomputation expressed in a standard digital programming language — the goal is instead to obtain an algebraically equivalent implementation of the specified dynamical system. To this end, Arco works with a dynamical system specification language (and not a standard digital programming language). The target hardware platform contains specialized components optimized for implementing dynamical systems (and not analog neural networks). There is no training step — Arco synthesizes an exact analog implementation of the dynamical system without training. Any inaccuracy in the solution comes from the inherent analog noise, not from the translation of the computation onto the analog hardware platform. And for our target class of biological dynamical systems, the analog noise is directly analogous to the noise present in the biological system.

The Arco synthesis algorithms use a tableau to organize a search for a configuration of the target analog hardware platform that is algebraically equivalent to the specified dynamical system. The broad concept of a tableau has been widely used in theorem proving [2] and for the synthesis of functional programs [24, 25]. In this context logical deduction rules transform a set of assertions and goals into a proof, potentially with output entries that make it possible to extract a program from the proof. Unlike these previous approaches, Arco works with complex multifunctional analog components, not standard programming language primitives. To correctly utilize these components, Arco uses algebraic unification and must deal successfully with the cascading relation entanglement inherent in the use of such powerful but complex analog components.

Also unlike these previous approaches, the Arco synthesis algorithms operate in the presence of resource constraints — they synthesize the dynamical system onto a hardware platform with finite resources. The Arco synthesis algorithms therefore must track the resources that have been consumed in the synthesis (including complex partially consumed components), with the synthesis failing if it consumes too many resources.

Code generators use tree pattern matching to translate code trees into sequences of machine instructions [3, 12, 17]. Arco, in contrast, works with dynamical systems that may contain feedback loops and circular dependencies. The Denali superoptimizer [21] combines mathematical and machine rewrite rules to generate search spaces of instruction sequences that implement a given computation. It then searches this space to find efficient instruction sequences. Arco, in contrast, produces inherently parallel analog hardware configurations with no concept of sequencing. As noted above, the target components include a finite set of complex, potentially partially utilized analog building blocks optimized for analog efficiency, not digital machine instructions. And the relevant reasoning involves continuous, nonlinear, potentially entangled transcendental functions, not digital logic.

7. Conclusion

Programmable analog devices have emerged as a powerful new computing substrate for scientific computations such as neuromorphic and cytomorphic computations[5, 7, 8, 31, 33, 34, 37, 41]. Arco provides the sophisticated compiler support required to effectively exploit this class of new, powerful, but complex computational platforms.

Acknowledgements

We would like to thank Sung Sik Woo for helpful discussions about programmable analog hardware platforms in general and the specific programmable analog devices that he is building. We would like to thank our shepherd Adrian Sampson and the anonymous reviewers for their very helpful feedback.

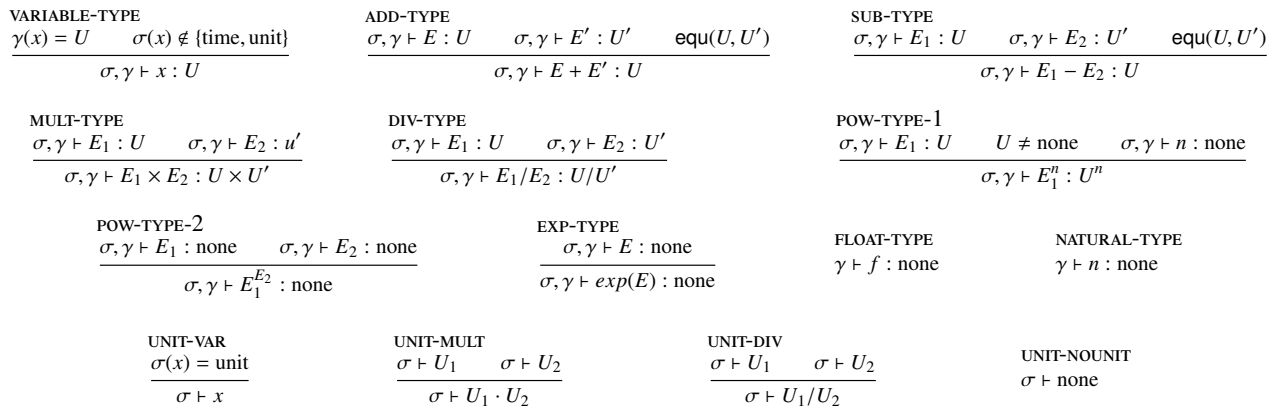


Figure 10. Type Checking Rules for DSSL Expressions E and Unit Expressions U.

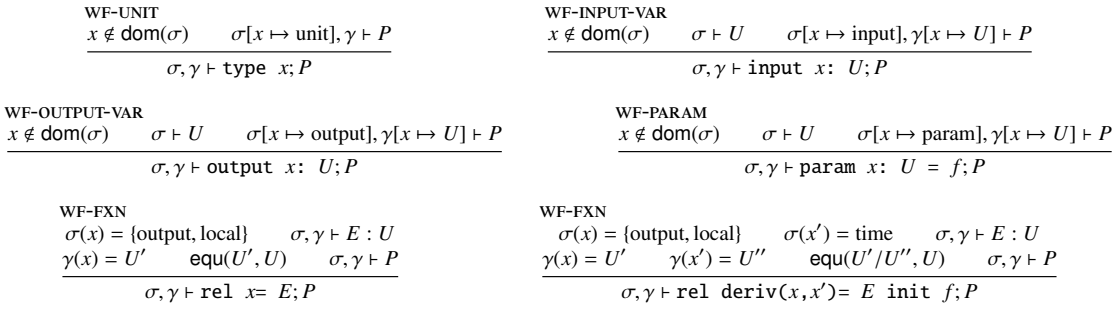


Figure 11. Type Checking Rules for DSSL Programs P.

A. DSSL Type Checker

A type judgment $\sigma, \gamma \vdash P$ states that the DSSL program P type checks in variable environment σ and unit environment γ.

- **The variable environment** σ: The variable environment $\sigma : \mathit{Ident} \rightarrow \mathit{Type} = \{\text{input, output, local, param, unit, time}\}$ tracks the binding of variables to types.
- **The unit environment** γ: The unit environment $\gamma : \mathit{Ident} \rightarrow \mathit{Unit}$ tracks the binding of variables to unit expressions.

Expressions: Figure 10 presents the type checking rules for expressions E. A type judgment $\sigma, \gamma \vdash E : U$ states that, with variable environment σ and unit environment γ, E type checks to produce a value measured in units U.

The type checking rules ensure only like units are added together and infer new unit expressions for products and quotients. The Add-Type and Sub-Type rules ensure that summands have like units and produce a value measured in the same units. The Mult-Type and Div-Type rules construct a unit expression that is the product or quotient of the unit expressions of the operands of the product or quotient.

Arco supports non-integer exponents only for unitless expressions. The Pow-Type1 rule ensures that if the base expression is not unitless, then the exponent expression must be a unitless integer constant n. The Pow-Type2 rule ensures that if the base expression E_1 is unitless, then the exponent expression E_2 must be a unitless integer.

Unit Expressions: Figure 10 also presents type checking rules for unit expressions U. A type judgment $\gamma \vdash U$ states that the unit expression U type checks in unit environment γ. The Unit-Var, Unit-Mult, Unit-Div, and Unit-None rules check that all of the variables x in the unit expression U are units.

Statements: Figure 11 presents the type checking rules for statements. These rules check that all variable declarations use defined units U, there are no variable redeclarations, that relation definitions `rel` only define values for output or local variables, and that the units of the defined variables x and the relevant expressions E are consistent. The $\text{equ}(U, U')$ function tests the equivalence of two or more unit expressions U and U'. It algebraically simplifies the unit expressions, interpreting none (no unit) as 1, and determines if the simplified expressions are equivalent.

$$\textsc{conn-comp}\ \frac{x \in \mathsf{dom}(\phi)}{\sigma,\gamma,\phi \vdash x : \langle x, \text{any}\rangle} \qquad \textsc{conn-port}\ \frac{\sigma,\gamma,\phi \vdash C : \langle x, k'\rangle \quad x' \neq \text{any} \quad \phi(x') = \langle \_,\_,\beta\rangle \quad \beta(x) = k \quad k = k' \vee k' = \text{any}}{\sigma,\gamma,\phi \vdash C\,.\,x : \langle x, k\rangle}$$

$$\textsc{conn-range}\ \frac{\sigma,\gamma,\phi,\vdash C : \langle x, k\rangle \quad x \neq \text{any} \quad \phi(x) = \langle n,\_,\_\rangle \quad 0 \leq n' \leq n \quad n' \leq n'' \leq n}{\sigma,\gamma,\phi \vdash C[n' \texttt{:} n'']} : \langle x, k\rangle \qquad \textsc{conn-all}\ \sigma,\gamma,\phi \vdash * : \langle \text{any}, \text{any}\rangle$$

Figure 12. Type Checking Rules for Port Collections C.

$$\textsc{port-prop-type}\ \frac{x \in \mathsf{dom}(\beta) \quad \gamma(x') = U \quad \sigma(x') = \text{prop}}{\sigma,\gamma,\alpha,\beta \vdash x'(x) : U} \qquad \textsc{param-type}\ \frac{\alpha(x) = U}{\sigma,\gamma,\alpha,\beta \vdash x : U}$$

Figure 13. Type Checking Rules for Hardware Expressions E.

$$\textsc{wf-unit}\ \frac{x \notin \mathsf{dom}(\sigma) \quad \sigma[x \mapsto \text{unit}],\gamma,\phi \vdash P}{\sigma,\gamma,\phi \vdash \texttt{type}\ x; P} \qquad \textsc{wf-prop}\ \frac{x \notin \mathsf{dom}(\sigma) \quad \sigma(x') = \text{unit} \quad \sigma[x \mapsto \text{prop}],\gamma[x \mapsto x'],\phi \vdash P}{\sigma,\gamma,\phi \vdash \texttt{prop}\ x\texttt{:}\ x'; P}$$

$$\textsc{wf-time}\ \frac{x \notin \mathsf{dom}(\sigma) \quad \sigma(x') = \text{unit} \quad \sigma[x \mapsto \text{time}],\gamma[x \mapsto x'],\phi \vdash P}{\sigma,\gamma,\phi \vdash \texttt{time}\ x\texttt{:}\ x'; P} \qquad \textsc{wf-comp}\ \frac{\sigma,\gamma,\alpha,\beta \vdash Y \quad \sigma,\gamma,\phi[x \mapsto \langle 0,\alpha,\beta\rangle] \vdash P}{\sigma,\gamma,\phi \vdash \texttt{comp}\ x\ Y\ \texttt{end}; P}$$

$$\textsc{wf-comp-param}\ \frac{x \notin \mathsf{dom}(\alpha) \quad \sigma,\gamma \vdash U \quad \sigma,\gamma,\alpha[x \mapsto U],\beta \vdash Y}{\sigma,\gamma,\alpha,\beta \vdash \texttt{param}\ x\texttt{:}U\texttt{=}f; Y} \qquad \textsc{wf-comp-input-port}\ \frac{x \notin \mathsf{dom}(\beta) \quad \sigma,\gamma,\alpha,\beta[x \mapsto \text{input}] \vdash Y}{\sigma,\gamma,\alpha,\beta \vdash \texttt{input}\ x; Y}$$

$$\textsc{wf-comp-output-port}\ \frac{x \notin \mathsf{dom}(\beta) \quad \sigma,\gamma,\alpha,\beta[x \mapsto \text{output}] \vdash Y}{\sigma,\gamma,\alpha,\beta \vdash \texttt{output}\ x; Y}$$

$$\textsc{wf-comp-rel}\ \frac{\sigma,\gamma,\alpha,\beta \vdash x'(x) : U \quad \sigma,\gamma,\alpha,\beta \vdash E : U' \quad \mathsf{equ}(U,U') \quad \beta(x) = \text{output} \quad \sigma,\gamma,\alpha,\beta \vdash Y}{\sigma,\gamma,\alpha,\beta \vdash \texttt{rel}\ x'(x)\texttt{=}E; Y}$$

$$\textsc{wf-comp-deriv}\ \frac{\sigma,\gamma,\alpha,\beta \vdash x'(x) : U \quad \sigma,\gamma,\alpha,\beta \vdash x'' : U'' \quad \sigma,\gamma,\alpha,\beta \vdash E : U' \quad \mathsf{equ}(U/U'',U') \quad \beta(x) = \text{output} \quad \beta(y) = \text{input} \quad i\sigma,\gamma,\alpha,\beta \vdash y'(y) : U''' \quad \sigma,\gamma,\alpha,\beta \vdash Y}{\sigma,\gamma,\alpha,\beta \vdash \texttt{rel deriv(x'(x),x'')=E init y'(y)}; Y}$$

$$\textsc{wf-schem}\ \frac{\sigma,\gamma,\phi \vdash Z \quad \sigma,\gamma,\phi \vdash P}{\sigma,\gamma,\phi \vdash \texttt{schematic Z end}; P} \qquad \textsc{wf-inst}\ \frac{\phi(x) = \langle n',\alpha,\beta\rangle \quad n > 0 \quad \sigma,\gamma,\phi[x \mapsto \langle n,\alpha,\beta\rangle] \vdash Z}{\sigma,\gamma,\phi,\vdash \texttt{inst x : n}; Z}$$

$$\textsc{wf-conn}\ \frac{\sigma,\gamma,\phi \vdash C : \langle \_,k\rangle \quad k = \text{output} \vee k = \text{any} \quad \sigma,\gamma\phi \vdash C' : \langle \_,k'\rangle \quad k' = \text{input} \vee k' = \text{any} \quad \sigma,\gamma,\phi \vdash Z}{\sigma,\gamma,\phi \vdash \texttt{conn}\ C\texttt{->}C'; Z}$$

Figure 14. Type Checking Rules for Analog Hardware Specifications P.

B. Analog Hardware Type Checker

A type judgment $\sigma,\gamma,\phi \vdash P$ states that the hardware specification P type checks in variable environment σ, unit environment γ, and component environment ϕ.

- **variable environment** σ: The variable environment σ : *Ident* $\rightarrow$ *Type* maps variable identifiers to types, where *Type* = {time, prop, unit, comp}.
- **unit environment** γ: The unit environment σ : *Ident* $\rightarrow$ *Unit* maps variable identifiers to unit expressions.
- **component environment** ϕ: The component environment $\sigma = \{\langle n,\alpha,\beta\rangle \in \sigma(x) \mid x \in \textit{Ident}\}$ maps the component identifier to the component definition information. The parameter environment α : *Ident* $\rightarrow$ *Unit* maps parameter identifiers to units. The port environment β : *Ident* $\rightarrow$ *PType* maps the port identifier to the port types $k \in$ *PType*, where *PType* = {input, output}. The instance count of the component $n \in \mathbb{N}$ specifies the number of component instances.

Port Collection Type Checking Rules: The WF-CONN rule in Figure 12 and the rules in Figure 12 check port collections C. A conn $C->C'$ statement specifies that all of the output ports identified by C can be connected to any of the input ports identified by C'. The hardware specification language supports several kinds of port collections:

- x[l:u].x': Port x' of instances l to u of component x.
- x[l:u]: All ports of instances l to u of component x.
- $x.x'$: Port x' of all instances of component x.
- x: All ports of all instances of component x.
- *: All ports of all instances of all components.

The rules check that for a statement conn $C->C'$, C identifies at least one output port and C' identifies at least one input port. The rules also check that C and C' are well

formed, identified component instances are within the maximum number of instances, and all ports and components are defined.

Expression Type Checking Rules: Figure 13 presents the PORT-PROP-TYPE and PARAM-TYPE type checking rules for expressions E. The remaining rules include all of the DSSL expression type checking rules from Figure 10 except the VARIABLE-TYPE rule. The type judgments are $\sigma, \gamma, \alpha, \beta \vdash E : U$ instead of $\sigma, \gamma \vdash E : U$. A type judgment $\sigma, \gamma, \alpha, \beta \vdash E : U$ states that E type checks to produce a value measured in units U.

Statement Type Checking Rules: Figure 14 presents the type checking rules for programs P. Each rule implements the type checks for a different kind of statement S.

References

[1] Biomodel artifact database. `https://www.ebi.ac.uk/biomodels-main`.

[2] *Handbook of Tableau Methods*. Springer, 1999. ISBN 0792356276.

[3] A. V. Aho, M. Ganapathi, and S. W. Tjiang. Code generation using tree matching and dynamic programming. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 11(4):491–516, 1989.

[4] K. Antreich, J. Eckmueller, H. Graeb, M. Pronath, E. Schenkel, R. Schwencker, and S. Zizala. Wicked: Analog circuit synthesis incorporating mismatch. In *Custom Integrated Circuits Conference, 2000. CICC. Proceedings of the IEEE 2000*, pages 511–514. IEEE, 2000.

[5] B. V. Benjamin, P. Gao, E. McQuinn, S. Choudhary, A. R. Chandrasekaran, J.-M. Bussat, R. Alvarez-Icaza, J. V. Arthur, P. Merolla, K. Boahen, et al. Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations. *Proceedings of the IEEE*, 102(5):699–716, 2014.

[6] D. G. Buerk. Can we model nitric oxide biotransport? a survey of mathematical models for a simple diatomic molecule with surprisingly complex biological activities. *Annual review of biomedical engineering*, 3(1):109–143, 2001.

[7] G. Cowan, R. Melville, and Y. Tsividis. A VLSI analog computer/digital computer accelerator. *Solid-State Circuits, IEEE Journal of*, 41(1):42–53, Jan 2006. ISSN 0018-9200. doi: 10.1109/JSSC.2005.858618.

[8] R. Daniel, S. S. Woo, L. Turicchia, and R. Sarpeshkar. Analog transistor models of bacterial genetic circuits. In *Biomedical Circuits and Systems Conference (BioCAS), 2011 IEEE*, pages 333–336. IEEE, 2011.

[9] L. De Moura and N. Bjørner. Z3: An efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.

[10] J. Douce and H. Wilson. The automatic synthesis of control systems with constraints. *Mathematics and Computers in Simulation*, 7(1):18 – 22, 1965.

[11] M. B. Elowitz and S. Leibler. A synthetic oscillatory network of transcriptional regulators. *Nature*, 403(6767):335–338, 2000.

[12] H. Emmelmann, F.-W. Schröer, and R. Landwehr. Beg: a generator for efficient back ends. In *ACM Sigplan Notices*, volume 24, pages 227–237. ACM, 1989.

[13] K. Erguler and M. P. Stumpf. Practical limits for reverse engineering of dynamical systems: a statistical analysis of sensitivity and parameter inferability in systems biology models. *Molecular BioSystems*, 7(5):1593–1602, 2011.

[14] K. Erguler, M. Pieri, and C. Deltas. A mathematical model of the unfolded protein stress response reveals the decision mechanism for recovery, adaptation and apoptosis. *BMC systems biology*, 7(1):16, 2013.

[15] J. Ernst and M. Kellis. ChromHMM: automating chromatin-state discovery and characterization. *Nature Methods*, 9(3): 215–6, mar 2012. ISSN 1548-7105. doi: 10.1038/nmeth.1906.

[16] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger. Neural acceleration for general-purpose approximate programs. In *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 449–460. IEEE Computer Society, 2012.

[17] C. W. Fraser, D. R. Hanson, and T. A. Proebsting. Engineering a simple, efficient code-generator generator. *ACM Letters on Programming Languages and Systems (LOPLAS)*, 1(3):213–226, 1992.

[18] T. S. Gardner, C. R. Cantor, and J. J. Collins. Construction of a genetic toggle switch in escherichia coli. *Nature*, 403(6767): 339–342, 2000.

[19] T. S. Hall, C. M. Twigg, J. D. Gray, P. Hasler, and D. V. Anderson. Large-scale field-programmable analog arrays for analog signal processing. *Circuits and Systems I: Regular Papers, IEEE Transactions on*, 52(11):2298–2307, 2005.

[20] R. Harjani, L. R. Carley, et al. Oasys: A framework for analog circuit synthesis. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 8(12):1247–1266, 1989.

[21] R. Joshi, G. Nelson, and K. Randall. *Denali: a goal-directed superoptimizer*, volume 37. ACM, 2002.

[22] D. Joyner, O. Čertík, A. Meurer, and B. E. Granger. Open source computer algebra systems: Sympy. *ACM Communications in Computer Algebra*, 45(3/4):225–234, 2012.

[23] E. Lalonde, A. S. Ishkanian, J. Sykes, M. Fraser, H. Ross-Adams, N. Erho, M. J. Dunning, S. Halim, A. D. Lamb, N. C. Moon, G. Zafarana, A. Y. Warren, X. Meng, J. Thoms, M. R. Grzadkowski, A. Berlin, C. L. Have, V. R. Ramnarine, C. Q. Yao, C. A. Malloff, L. L. Lam, H. Xie, N. J. Harding, D. Y. Mak, K. C. Chu, L. C. Chong, D. H. Sendorek, C. P'ng, C. C. Collins, J. A. Squire, I. Jurisica, C. Cooper, R. Eeles, M. Pintilie, A. Dal Pra, E. Davicioni, W. L. Lam, M. Milosevic, D. E. Neal, T. van der Kwast, P. C. Boutros, and R. G. Bristow. Tumour genomic and microenvironmental heterogeneity for integrated prediction of 5-year biochemical recurrence of prostate cancer: a retrospective cohort study. *Lancet Oncol*, 15(13):1521–1532, 2014. ISSN 1474-5488. doi: 10.1016/s1470-2045(14)71021-6.

[24] Z. Manna and R. Waldinger. A deductive approach to program synthesis. *ACM Transactions on Programming Languages*

and Systems (TOPLAS), 2(1):90–121, 1980.

[25] Z. Manna and R. Waldinger. Fundamentals of deductive program synthesis. *Software Engineering, IEEE Transactions on*, 18(8):674–704, 1992.

[26] E. Ochotta, R. Rutenbar, and L. Carley. Synthesis of high-performance analog circuits in ASTRX/OBLX. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 15(3):273–294, Mar 1996. ISSN 0278-0070. doi: 10.1109/43.489099.

[27] Y. Paker and S. H. Unger. {ADAC} — a programmed direct analog computer. *Mathematics and Computers in Simulation*, 9(1):16 – 23, 1967.

[28] B. Pankiewicz, M. Wojcikowski, S. Szczepanski, and Y. Sun. A field programmable analog array for CMOS continuous-time OTA-C filter applications. *Solid-State Circuits, IEEE Journal of*, 37(2):125–136, 2002.

[29] L. Petzold. Automatic selection of methods for solving stiff and nonstiff systems of ordinary differential equations. *SIAM journal on scientific and statistical computing*, 4(1):136–148, 1983.

[30] D. R. F. PhD. *Biochemistry (Lippincott Illustrated Reviews Series)*. LWW, 2013. ISBN 1451175620.

[31] S. Saighi, Y. Bornat, J. Tomas, G. Le Masson, and S. Renaud. A library of analog operators based on the Hodgkin-Huxley formalism for the design of tunable, real-time, silicon neurons. *Biomedical Circuits and Systems, IEEE Transactions on*, 5(1): 3–19, 2011.

[32] Sams. Arrangement and scaling of equations. *Mathematics and Computers in Simulation*, 6(3):179 – 182, 1964.

[33] R. Sarpeshkar. *Ultra Low Power Bioelectronics: Fundamentals, Biomedical Applications, and Bio-Inspired Systems*. Cambridge University Press, 2010. ISBN 0521857279.

[34] C. Schneider and H. Card. Analog CMOS synaptic learning circuits adapted from invertebrate biology. *Circuits and Systems, IEEE Transactions on*, 38(12):1430–1438, 1991.

[35] R. I. Sherwood, T. Hashimoto, C. W. O'Donnell, S. Lewis, A. a. Barkal, J. P. van Hoff, V. Karun, T. Jaakkola, and D. K. Gifford. Discovery of directional and nondirectional pioneer transcription factors by modeling DNase profile magnitude and shape. *Nature Biotechnology*, 32(2):171–8, mar 2014. ISSN 1546-1696. doi: 10.1038/nbt.2798. URL `http://www.ncbi.nlm.nih.gov/pubmed/24441470`.

[36] R. St Amant, A. Yazdanbakhsh, J. Park, B. Thwaites, H. Esmaeilzadeh, A. Hassibi, L. Ceze, and D. Burger. General-purpose code acceleration with limited-precision analog computation. *ACM SIGARCH Computer Architecture News*, 42 (3):505–516, 2014.

[37] J. J. Y. Teo, S. S. Woo, and R. Sarpeshkar. Synthetic biology: A unifying view and review using analog circuits. *IEEE Trans. Biomed. Circuits and Systems*, 9(4):453–474, 2015.

[38] R. Tomovic. Proceedings of the international association for analog computation method of iteration and analog computation. *Mathematics and Computers in Simulation*, 1(2):60 – 63, 1958.

[39] J. M. Vilar, H. Y. Kueh, N. Barkai, and S. Leibler. Mechanisms of noise-resistance in genetic oscillators. *Proceedings of the National Academy of Sciences*, 99(9):5988–5992, 2002.

[40] H. Weiner. The illusion of simplicity: the medical model revisited. *The American journal of psychiatry*, 1978.

[41] S. S. Woo, J. Kim, and R. Sarpeshkar. A cytomorphic chip for quantitative modeling of fundamental bio-molecular circuits. *IEEE Trans. Biomed. Circuits and Systems*, 9(4):527–542, 2015.