Template-Free Symbolic Performance Modeling of Analog Circuits Via Canonical Form Functions and Genetic Programming

Trent McConaghy, *Member, IEEE*, and Georges G.E. Gielen, *Fellow, IEEE*

***Abstract*—This paper presents CAFFEINE, a method to automatically generate compact, interpretable symbolic performance models of analog circuits with no prior specification of an equation template. CAFFEINE uses SPICE simulation data, to model arbitrary nonlinear circuits and circuit characteristics. CAFFEINE expressions are canonical form functions: product-of-sum layers alternating with sum-of-product layers, as defined by a grammar. Multi-objective genetic programming trades off error with model complexity. On test problems, CAFFEINE models demonstrate lower prediction error than posynomials, splines, neural networks, kriging, and support vector machines. This paper also demonstrates techniques to scale CAFFEINE to larger problems.**

***Index Terms*—Macromodeling, Yield Modeling, Performance Optimization, Transistor Sizing, Posynomial**

I. Introduction

BOTH *symbolic analysis* and *symbolic modeling* aim to derive human-interpretable expressions of analog circuit behavior [1]. Symbolic analysis extracts the expressions via topological analysis of the circuit, whereas symbolic modeling extracts the expressions by using SPICE simulation data. These expressions have the same applications: knowledge acquisition and educational / training purposes, analytic model generation for automated circuit sizing, design space exploration, repetitive formula evaluation including statistical analysis, analog fault diagnosis and testability analysis, and analog behavioral model generation [2]. In particular, a tool that can help a designer improve his understanding of a circuit is highly valuable, because it leads to better decision-making in circuit sizing, layout, verification, and topology design, regardless of the degree of automation [27]. Therefore, approaches to generate symbolic expressions are of great interest.

Historically, symbolic analysis came first, starting with ISAAC [3] and followed by several other techniques; see [2] for a review. Until recently, the main weakness was their limitation to linearized and weakly nonlinear circuits. This was overcome via piecewise-linear/polynomial modeling approaches [4]–[6], but at the cost of interpretability.

Leveraging SPICE simulations in modeling is promising because simulators readily handle nonlinear circuits, environmental effects (e.g. temperature, power supply voltage, loads), manufacturing effects, different technologies, new effects (e.g. proximity [7]), and more. From simulation data, a model $y = f(\boldsymbol{x})$ is constructed, where y is typically a performance metric, $\boldsymbol{x}$ includes design, process, or environmental variables, and f is an approximation of the SPICE mapping. Models used include linear models [8], [9], [25], posynomials [10]–[12], polynomials [13], [14], [25], splines [15], [25], neural networks [16], [17], [25], boosted neural networks [18], [25], support vector machines [19]–[21], [25], latent variable regression (LVR) [22], [23], kriging [24], [25], and stochastic gradient boosting [26]. However, such models either follow an overly restrictive functional template which limits their applicability, or they are opaque and thus provide no insight to the designer. Less opaque flows exist, such as visualizing CART trees [26], nonlinear sensitivity analysis [26], or plotting the mapping from an LVR's first affine transform $\boldsymbol{w_1} * \boldsymbol{x}$ to the output $y = f_1(\boldsymbol{w_1} * \boldsymbol{x})$ [22], [23]. While useful, these approaches do not give the functional relations that symbolic models provide.

The aim of *symbolic modeling* as defined in this paper is to use simulation data to generate *interpretable mathematical expressions* for circuit applications, typically relating the circuit performances to the design variables. Symbolic modeling has similar goals to symbolic analysis, but a different core approach to solving the problem. In [10]–[12], posynomial-based symbolic models are constructed. The main problem is that the models are constrained to a predefined template, which restricts the functional form. Also, the models have dozens of terms, limiting their interpretability for designers. Finally, the approach assumes posynomials can fit the data; in analog circuits there is no guarantee of this. There have also been advances in building quadratic polynomial models [13], [14], but polynomials also have a restrictive structure.

The problem we address in this paper is how to generate symbolic models with more *open-ended* functional forms (i.e. without a pre-defined template), for arbitrary nonlinear circuits and circuit characteristics, and at the same time ensure that the models are *interpretable*. Figure 1 shows a target flow that reflects these goals. Note that symbolic modeling is most suited to properly-biased design regions (smaller changes in design variables), because models covering incorrectly-biased regions too would be too complex for manual inspection.

We approach the task by posing it as a search problem in the space of possible functional form trees. An appropriate search algorithm is then genetic programming (GP) [28]. GP generates symbolic expressions without using a template, but

Manuscript received March 28, 2008; revised December 23, 2008.

T. McConaghy was with ESAT-MICAS, Katholieke Universiteit, Leuven, Belgium. He is now with Solido Design Automation Inc., Canada.

G.G.E. Gielen is with ESAT-MICAS, Katholieke Universiteit, Belgium.

Funding for the reported research results is acknowledged from IWT/Medea+ Uppermost, Solido Design Automation Inc. and FWO Flanders..

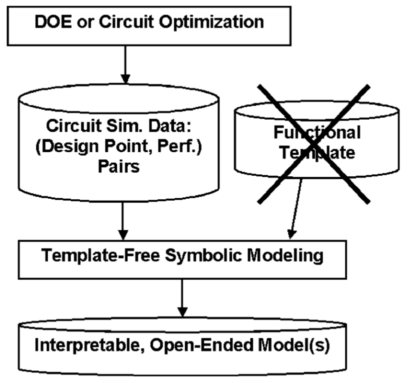
DOE or Circuit Optimization
Circuit Sim. Data: (Design Point, Perf.) Pairs
Functional Template
Template-Free Symbolic Modeling
Interpretable, Open-Ended Model(s)

Fig. 1. Template-free symbolic modeling flow.

those functions are overly complex. So, we extend GP via a *grammar* [29] specifically designed to have simpler but accurate, *interpretable* symbolic models. We name the approach CAFFEINE: Canonical Functional Form Expressions in Evolution[1].

The contributions of this paper are as follows:

- To the best of our knowledge, the first-ever tool for *template-free symbolic modeling*. Because it uses SPICE simulation data, it allows modeling of any nonlinear circuit characteristic, or analysis (including transient, noise, and more).
- The approach returns models that are compact and understandable, yet with good accuracy. In fact, it returns a *set* of possible models that *trade off* accuracy and complexity by using multi-objective search [30].
- A specially-designed grammar to give functions a *canonical form*, which enhances interpretability. The grammar plugs into any grammatical-GP engine, e.g. [29], [31].
- Techniques to *scale* symbolic modeling to problems with more than 100 input variables. The techniques are: subtree caching [32], gradient-directed regularization [33] to simultaneously prune basis functions and learn remaining coefficients, a pre-evolution step of filtering single-variable expressions, and always considering all the linear basis functions.

This paper is organized as follows. Section II presents the problem formulation. Section III presents background on genetic programming. Section IV introduces the heart of CAFFEINE: canonical form functions. Section V describes the reference search algorithm, which uses multiobjective genetic programming and a grammar to constrain to canonical form functions. Section VI describes the first round of experiments. Section VII describes how to scale up CAFFEINE to larger problems, with corresponding experiments in section VIII. Section IX describes other applications of CAFFEINE. Section X discusses the sensitivity of canonical form functions to the search algorithm employed. Section X concludes.

[1]An earlier version of CAFFEINE appeared in [27].

II. Problem Formulation

The modeling problem that we address has the flow of Figure 1. Its inputs and outputs are as follows.
Given:

- $\boldsymbol{X}$ and $\boldsymbol{y}$: A set of $\{\boldsymbol{x_j}, y_j\}, j = 1..N$ data samples where $\boldsymbol{x_j}$ is a N_d-dimensional design point j and y_j is a corresponding circuit performance value measured from SPICE simulation of that design. Design of experiments (DOE) [34] *or* circuit optimization can be used to generate the data samples.
- **No** model template

Determine:

- A set of symbolic models M that provide the Pareto-optimal tradeoff between minimizing model complexity f_1 and minimizing future model prediction error f_2.

The formulation is a constrained optimization problem:

$$M = minimize \begin{Bmatrix} f_1 & = & complexity(m) \\ f_2 & = & E_{x,y}L(y, m(\boldsymbol{x})) \end{Bmatrix} s.t.\ m \in \Psi \quad (1)$$

where Ψ is the space of template-free symbolic models. The algorithm will traverse Ψ to return a Pareto-optimal set $M = \{m_1^*, m_2^*, \ldots, m_{N_M}^*\}$. Each model m maps an N_d-dimensional input $\boldsymbol{x}$ to a scalar circuit performance approximation $\hat{y}$, i.e. $\hat{y} = m(\boldsymbol{x})$. Complexity is *some* measure that differentiates the degrees of freedom between different models (details are in eqn. (5)). $E_{x,y}L$ is the expected loss for a given m over future predictions in the distribution $pdf(\boldsymbol{x})$, where L is the squared-error loss function [33]:

$$L(y, m(\boldsymbol{x})) = (y - m(\boldsymbol{x}))^2)/2 \quad (2)$$

Section V-A describes how an approximation for $L()$ is computed. By definition, no model in the Pareto-optimal set M dominates any other model. A model m_a "dominates" another model m_b if $\{f_j(m_a) \leq f_j(m_b)\}\forall j$, and $\{f_j(m_a) < f_j(m_b)\}\exists j$; $j = \{1, 2\}$ in our case. That is, to be Pareto-optimal, a model must be at least as good as any model on both objectives, and better than any model in one objective.

III. Background: Genetic Programming

Genetic Programming (GP) [28] is an evolutionary algorithm, with the distinguishing characteristic that GP individuals (points in the design space) are *trees*. Since a symbolic model is a function and can be represented as a tree, the search for the above models can be accomplished by GP search.

The functional form of results from canonical GP is completely unrestricted. While this sounds promising compared to the restrictions of fixed-template regression, it actually goes a little too far: an unrestricted form is almost always difficult to analyze. GP-evolved functions can be notoriously *complex* and *un-interpretable*. For example, [28] showed functions so bloated [35] that they take up a full page of dense text. A recent paper complains: "[GP-evolved] expressions can get, as we have seen, quite complex, and it is often extremely difficult to understand them without a fair bit of interaction with a tool such as *Mathematica*" [36].

We can see for ourselves. Using a dataset from section VI, canonical GP evolution returned the following expression[1]:

```
- 1.40 * ( vsg1 + max( vsg5, max( max( max( vsg5,
max( vsg3 + vgs2, min( vsg3, abs( 1/vds2 ) ) ) -
log10(vsd5) ), min( ib2, abs( sqrt( abs(id1) ) ) ) )
- log10(vsd5), max( id2, min( vsg3, abs( sqrt( abs(
log10(id2) ) ) ) ) ) + log10(vsd5) ) - min( vsg3,
abs( sqrt( abs(id1) ) ) ) - log10(vsd5) ) )
```

Improvements in interpretability are clearly needed. The next section presents CAFFEINE to handle this issue.

IV. CAFFEINE Canonical Form Functions

The design of CAFFEINE follows two guidelines: ensure maximum expressiveness per node, and make all candidate functions directly interpretable. Figure 2 shows the general structure of a CAFFEINE function. It alternates between levels of *sum-of-product* expressions and *product-of-sum* expressions. Each sum-of-product expression is a weighted linear add of an overall offset term plus weighted basis functions. A basis function is a combination of product terms, where each product term is a polynomial/rational, zero or more nonlinear operators, and zero or more unity operators. Each product term acts as a "gate" to the next sum-of-products layer.

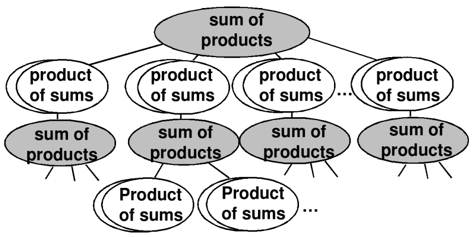

Fig. 2. CAFFEINE evolves functions of this canonical form. While it can go deeper indefinitely, it is typically only as deep as shown in order to retain human interpretability.

Figure 3 shows an example function and its corresponding tree. In the "$7.1/x_3$" part of the function, the 7.1 is the tree's top left "w_0" and the "$1/x_3$" is its neighboring "poly/rat'l of vars". The "1.8" corresponds to top "w_1", and the "x_1" is the its neighboring "poly/rat'l of vars". The function's "log" corresponds to "nonlinear func", which in the tree holds the "weighted linear add" term "$-1.9+8.0/x_1+1.4*x_2^2/x_3$". That term itself breaks down: function's the "-1.9" is the tree's lower "w_{offset}"; "$8.0/x_1$" corresponds to the tree's lower left "w_0" * "poly/rat'l of vars"; and "$1.4*x_2^2/x_3$" corresponds to the tree's lower right "w_1" * "poly/rat'l of vars". Note how CAFFEINE places coefficients only where they are needed, and nowhere else.

Figure 4 gives an example which has unity functions for product terms. Note how there is *no* nonlinear function that gates one layer of linear adds to the next – this is how CAFFEINE supports a product-of-sums formulation.

Typical usage of CAFFEINE would restrict the number of product term layers to just one or two, therefore ensuring that there is not an excessive compounding of nonlinear

[1]The expression font and style are presented like [28].

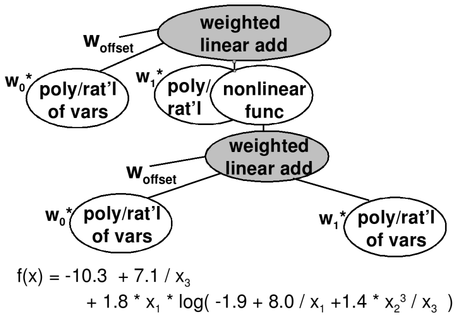
weighted linear add
w_{offset}
w_0* poly/rat'l of vars
w_1* poly/ rat'l
nonlinear func
weighted linear add
w_{offset}
w_0* poly/rat'l of vars
w_1* poly/rat'l of vars
f(x) = -10.3 + 7.1 / x_3
+ 1.8 * x_1 * log(-1.9 + 8.0 / x_1 +1.4 * x_2^3 / x_3)

Fig. 3. Example of a function in text form, and its corresponding CAFFEINE tree form.

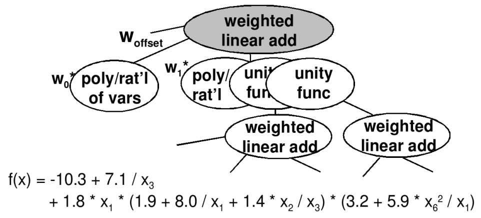
weighted linear add
w_{offset}
w_0* poly/rat'l of vars
w_1* poly/ rat'l
unity func
unity func
weighted linear add
weighted linear add
f(x) = -10.3 + 7.1 / x_3
+ 1.8 * x_1 * (1.9 + 8.0 / x_1 + 1.4 * x_2 / x_3) * (3.2 + 5.9 * x_6^2 / x_1)

Fig. 4. Example where CAFFEINE product terms include unity functions.

components such as $log(sin(exp(x)))$. There can also be a limit on the maximum number of basis functions. Due to the use of a canonical form, all evolved functions are immediately interpretable, with no symbolic manipulation needed.

V. CAFFEINE Search Algorithm

This section describes the search algorithm used on CAFFEINE functions. CAFFEINE search uses GP as a starting point, but extends it in order to properly address template-free symbolic modeling. It attacks the issues of complexity and interpretability in two main ways: a multi-objective approach that provides a tradeoff between error and complexity, a specially designed grammar and operators to constrain the search to specific functional forms without cutting out good solutions. As described in the previous section, in CAFFEINE the overall expression is a linear function of N_B basis functions $B_i; i = 1, 2, ..., N_B$:

$$\hat{y} = m(\boldsymbol{x}) = a_0 + \sum_{i=1}^{N_B} a_i * B_i(\boldsymbol{x}) \qquad (3)$$

A CAFFEINE individual m has one GP tree to define each basis function: $m = \{B_1, B_2, ..., B_{N_B}\}$. The linear coefficients $\boldsymbol{a} \in \Re^{N_B+1}$ are determined on-the-fly using linear regression on the least-squares cost function (2).

A. Multi-Objective Approach

CAFFEINE uses a state-of-the-art *multi-objective* evolutionary algorithm, namely NSGA-II [30]. NSGA-II returns a set of individuals that, collectively, trade off model error and complexity. Error and complexity are objectives f_1 and f_2 in eqn. (1). Error (expected loss $E_{x,y}L$) is approximated by the "training error" ϵ_{tr}, which is is the normalized root mean squared error of individual m on the training data:

$$\epsilon_{tr}(m) = \sqrt{\frac{1}{N_{tr}} * \sum_{i=1}^{N_{tr}} \left(\frac{\widehat{y_{tr,i}} - y_{tr,i}}{max(\boldsymbol{y}) - min(\boldsymbol{y})} \right)^2} \quad (4)$$

where N_{tr} is the number of training samples, $y_{tr,i}$ is sample i of training outputs $\boldsymbol{y_{tr}}$, $\widehat{y_{tr,i}} = F(\boldsymbol{x_{tr,i}}; m)$, and $\boldsymbol{x_{tr,i}}$ is sample i of training inputs $\boldsymbol{X_{tr}}$. Note that the y-values are scaled by $\boldsymbol{y}$, not $\boldsymbol{y_{tr}}$. ϵ_{test} has a similar formula, except the N_{tr} training points $\{\boldsymbol{y_{tr}}, \boldsymbol{X_{tr}}\}$ are replaced by the N_{test} testing points $\{\boldsymbol{y_{test}}, \boldsymbol{X_{test}}\}$.

Complexity is measured from the number of basis functions, the number of nodes in each tree, and the exponents of "variable combos" (VCs), according to:

$$complexity(m) = \sum_{j=1}^{N_B} (w_b + nnodes(j) + \sum_{k=1}^{nvc(j)} vccost(vc_{k,j})) \quad (5)$$

where w_b is a constant to give a minimum cost to each basis function, $nnodes(j)$ is the number of tree nodes of basis function j, and $nvc(j)$ is the number of VCs of basis function j, with cost:

$$vccost(vc) = w_{vc} * \sum_{i=1}^{N_d} |vc(i)| \quad (6)$$

The approach accomplishes *simplification during generation* [37] by maintaining evolutionary pressure towards lower complexity. The user avoids an *a priori* decision on error or complexity because the algorithm generates a *set* of models that provide tradeoffs of alternatives, rather than producing just *one* model.

Note that specific parameter settings are given in the experiments (section VI).

B. Grammar Implementation of Canonical Form Functions

In GP, a means of constraining search is via a grammar, as in [29]. Tree-based evolutionary operators such as crossover and mutation must respect the derivation rules of the grammar.

Even though grammars can usefully constrain search, none have yet been carefully designed for functional forms. In designing such a grammar, it is important to allow all functional combinations (even if just in one canonical form).

The CAFFEINE grammar, shown in Table I is explicitly designed to create separate layers of linear and nonlinear functions and to place coefficients and variables carefully, in adherence with Figure 2

TABLE I
CAFFEINE GRAMMAR.

REPVC $\mapsto$	VC | **REPVC** * **REPOP** | **REPOP**
REPOP $\mapsto$	**REPOP** * **REPOP** | **OP_1ARG** (W + **REPADD**) |
	OP_2ARG (**2ARGS**) | ... **3OP**, **4OP**, *etc*
2ARGS $\mapsto$	W + **REPADD**, **MAYBEW** | **MAYBEW**, W + **REPADD**
MAYBEW $\mapsto$	W | W + **REPADD**
REPADD $\mapsto$	W * **REPVC** | **REPADD** + **REPADD**
OP_2ARG $\mapsto$	DIVIDE | POW | MAX | *etc*
OP_1ARG $\mapsto$	INV | LOG10 | *etc*

First, we describe the notation of Table I. The nonterminal symbols are in bold-case; terminal symbols are not. Each line (or two) shows the possible expressions that a nonterminal symbol on the left can map ($\mapsto$) into. The possible expressions, i.e. "derivation rules" are separated by the OR operator "|".

We now explain how the grammar implements canonical form functions. REP is short for "repeating", such as "repeating operators" REPOP and "repeating variable combo" REPVC, which are explained further. The start symbol is REPVC, which expands into one basis function (remember that an individual has several root-level basis functions). Note the strong distinction among operators. The root is a product of variables (REPVC) and / or nonlinear functions (REPOP). Within each nonlinear function is REPADD, the weighted sum of next-level basis functions.

A VC is a "variable combo", intended to maintain a compact representation of polynomials/rationals. Its expansion could have been implemented directly within the grammar; though in our baseline system we store a vector holding an integer value per design variable as the variable's exponent. An example vector is [1,0,-2,1], which means $(x_1 * x_4)/(x_3)^2$, and according to eqn. (6) has cost $|1| + |0| + |-2| + |1| = 4$. This approach guarantees compactness and allows for special operators on the vector.

In determining coefficient values, we distinguish between linear and nonlinear coefficients. As described, a CAFFEINE individual is a set of basis functions which are linearly added. Each basis function is a tree of grammatical derivations. Linear coefficients are found by evaluating each tree across all input samples to get a matrix of basis function outputs, then to apply least-squares regression with that matrix and the target output vector to find the optimal linear weights.

With each nonlinear coefficient W in the tree (i.e. ones that are not found via linear regression), a real value will accompany it, taking a value in the range $[-2 * B, +2 * B]$. During interpretation of the tree the value is transformed into $[-1e+B, -1e-B] \cup [0.0] \cup [1e-B, 1e+B]$. B is user-set; see section VI-A.

POW(a,b) is a^b. When the symbol 2ARGS expands to include MAYBEW, either the base or the exponent (but not both) can be constants.

The designer can turn off any of the rules in the grammar of Table I, if they are considered unwanted or unneeded. For example, he could easily restrict the search to polynomials or rationals, or remove potentially difficult-to-interpret functions such as *sin* and *cos*. He could also change or extend the operators or inputs, e.g. include W_i, L_i, and W_i/L_i.

C. High-Level CAFFEINE Algorithm

Table II gives the algorithm *ExtractSymbolicCaffeineModels()*. It takes in the training inputs $\boldsymbol{X}$ and training outputs $\boldsymbol{y}$. It will output a Pareto-optimal set of models, M. Line 1 initializes M, the current set of parents P, and current set of children Q, all to empty sets. Lines 2-3 loops across the population size N_{pop} to randomly draw each individual P_i from the space of possible canonical form functions Ψ. Line 4 begins the EA's generational loop of lines 5 and 6. The loop stops when the target number of generations $N_{gen,max}$ is hit. Line 5 does the main EA work, which here is a single generation of the NSGA-II multi-objective EA (see [30] for details). Line 6 updates the external archive of Pareto-optimal individuals, M, by nondominated-filtering on the existing M with the recently updated parents P and children Q. Line 7 of Table II concludes the *ExtractSymbolicCaffeineModels()* routine, by returning the Pareto-optimal symbolic models, M.

TABLE II
PROCEDURE EXTRACTSYMBOLICCAFFEINEMODELS()

```
Inputs: $\boldsymbol{X}$, $\boldsymbol{y}$
Outputs: $M$
1. $M = \emptyset$; $P = \emptyset$; $Q = \emptyset$
2. for $i = 1..N_{pop}$:
3.     $P_i \sim \Psi$
4. for $N_{gen} = 1..N_{gen,max}$:
5.     $\{P, Q\}$ = OneNsgaiiGeneration($P$, $Q$)
6.     $M$ = nondominatedFilter($M \cup P \cup Q$)
7. return $M$
```

D. Evolutionary Search Operators

We now describe how trees are randomly generated, and explain the search operators on the trees. The search operators are grouped by the aspect of search representation that they concern: grammar, real-valued coefficient, variable combos (`VCs`), and basis functions.

Random generation of trees and subtrees from a given symbol involves merely randomly picking one of the derivations of one of the symbols, and recursing the (sub)tree until terminal symbols are encountered (subject to tree depth limits).

Grammatical restrictions on the trees lead to a natural grammar-obeying crossover operator and mutation operator, as described by Whigham [29]. Whigham-style crossover works as follows: it randomly picks a node on the first parent, then randomly picks a node on the second parent with the constraint that it must be the same grammatical symbol (e.g. `REPOP`) as the first node, and finally swaps the subtrees corresponding to each node. Whigham-style mutation involves randomly picking a node, then replacing its subtree with a randomly-generated subtree (as in the generation of initial trees).

Real-valued coefficients are mutated according to a Cauchy distribution [38], which cleanly combines aggressive local tuning with the occasional large change.

The specialized structure of `VCs` get appropriate operators, which include: one-point crossover, and randomly adding or subtracting to an exponent value.

Each individual has a list of basis functions, which also leads to special operators: creating a new individual by randomly choosing >0 basis function from each of 2 parents; deleting a random basis function; adding a randomly generated tree as a basis function; copying a subtree from one individual to make a new basis function for another.

VI. EXPERIMENTAL RESULTS

This section describes the application of CAFFEINE to building symbolic models for analog circuits that map design variables to performances, for problems with 13 input variables. It shows the actual symbolic models generated, measured error vs. complexity tradeoffs, how prediction error and complexity compare to posynomials, and how prediction error compares to other state-of-the-art (blackbox) regression approaches. The extension to larger problems is described in section VII.

A. Experimental Setup

Unary operators allowed are: $\sqrt(x)$, $log_{10}(x)$, $1/x$, x^2, $sin(x)$, $cos(x)$, $tan(x)$, $max(0, x)$, $min(0, x)$, 2^x, and 10^x, where x is an expression. Binary operators allowed are $x_1 + x_2$, $x_1 * x_2$, $max(x_1, x_2)$, $min(x_1, x_2)$, $power(x_1, x_2)$, and x_1/x_2. Conditional operators included $\leq$ $(testExpr$, $condExpr$, $exprIfLessThanCond$, $elseExpr)$ and $\leq$ $(testExpr$, 0, $exprIfLessThanCond$, $elseExpr)$. Any input variable could have an exponent in the range $\{\ldots$, -1, 1, 2, $\ldots\}$. While real-valued exponents could have been used, that would have harmed interpretability.

The circuit being modeled in this example is a high-speed CMOS OTA as shown in Figure 5. The goal is to discover expressions for the low-frequency gain (A_{LF}), unity-gain frequency (FU), phase margin (PM), input-referred offset voltage ($VOFF$), and the positive and negative slew rate (SR_p, SR_n). To allow a direct comparison to the posynomial approach [10], an almost-identical problem setup was used, as well as identical simulation data. The only difference is that, because scaling makes the model less interpretable, neither the inputs nor the outputs were scaled. The one exception is that FU is log-scaled so that the mean-squared error calculations and linear learning are not wrongly biased towards high-magnitude samples of FU. The technology is 0.7μm CMOS. The supply voltage is 5V. $V_{th,nom}$ is 0.76V and -0.75V for the NMOS and PMOS devices, respectively. The load capacitance is 10 pF.

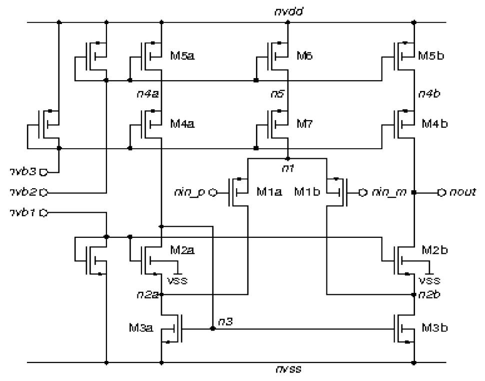

Fig. 5. CMOS high-speed OTA.

Good training data is essential to the methodology. The choice of design variables and sampling methodology determines the extent to which the designer can make inferences about the physical basis, and what regions of design space the model is valid in. We used an operating-point driven formulation [39], where currents and transistor gate drive voltages comprise design variables (13 variables in our case). Device sizings could have been used as design variables instead; it all depends on designer preference. Full orthogonal-hypercube Design-Of-Experiments (DOE) [34] sampling of design points was used, with scaled dx=0.1 (where dx is % change in variable value from center value)[1] to have 243 samples. Simulation time for one sample was about 1 s, or 4 min for all samples; this is fully dependent on the circuit, analyses, and experimental design method being used. These samples, otherwise unfiltered, were used as training data inputs. Testing data inputs were also sampled with full orthogonal-hypercube DOE and 243 samples, but with dx=0.03. Thus, in this experiment we are creating a somewhat localized model; one could just as readily model a broader design space, but this allows us to compare the results to [10].

The run settings were: N_B = maximum number of basis functions = 15 (any larger is definitely non-interpretable), N_{pop} = population size = 200 (like NSGA-II's default), $N_{gen,max}$ = 5000 generations (more than enough time to converge), maximum tree depth = 8 (so that each basis function has exactly one layer of nonlinear operators), and "W" coefficients range $[-1e+10, -1e-10] \cup [0.0] \cup [1e-10, 1e+10]$ (i.e. B=10; therefore coefficients can cover 20 orders of magnitude, both positive and negative). All operators had equal probability (a reliable setting), except parameter mutation was 5x more likely (to encourage tuning of a compact function). Complexity measure settings were w_b = 10, w_{vc} = 0.25. That is, the cost of adding a basis function is relatively high compared to the cost of adding another variable combo.

One run was done for each performance goal, for 6 runs total. The original non-optimized implementation was in Matlab, therefore using pass-by-value functions; a single run took approximately 12 hours. The optimized implementation was in Python using pass-by-reference functions, caching, and more as described in section VII; a single run on these problems took approximately 10 minutes.

We calculate normalized mean-squared error on the training data and on the separate testing data: ϵ_{tr} and ϵ_{test} as described in eqn. (4). These are standard measurements of model quality in regression literature. The testing error ϵ_{test} is ultimately the more important measure, because it measures the model's ability to generalize to unseen data. These measures are identical to two of the three posynomial "quality of fit" measures in [10]: its measure "worst-case quality" q_{wc} *is* the training error ϵ_{tr}, and its measure "typical case quality" q_{tc} *is* ϵ_{test} (as long as long as the constant 'c' in the denominator is set to zero, which [10] did.)

B. *Results: Whitebox Models and Tradeoffs*

Let us first examine some symbolic models generated by CAFFEINE. We ask: "which symbolic models have $<$ 10% training and testing error, with the lowest complexity?" Table III shows those functions. (Note that FU has been converted to its true form by putting the generated function to the power of 10). We see that each function has up to four basis functions, not including the constant. For $VOFF$, a constant was sufficient to keep the error within 10%. We see that a rational functional form was favored heavily; at these target errors only one nonlinear function, $ln()$, appears (for A_{LF}). The $ln()$ indicates that the order of magnitude of some input variables is meaningful.

[1]The simpler problem of dx=0.01 from [10] is ignored in this paper.

TABLE III
CAFFEINE-GENERATED SYMBOLIC CIRCUIT MODELS OF THE OTA OF FIGURE 5 WITH <10% TRAINING ERROR AND <10% TESTING ERROR.

Perf. Char.	**Expression**
ALF	$-10.3 + 7.08e\text{-}5/i_{d1}$ $+1.87 * ln(-1.95e{+}9{+}1.00e{+}10/(v_{sg1} * v_{sg3})$ $+1.42e{+}9{*}(v_{ds2} * v_{ds5})/(v_{sg1} * v_{gs2} * v_{gs5} * i_{d2}))$
FU	$10^{(5.68-0.03*v_{gs1}/v_{ds2}-55.43*i_{d1}+5.63e-6/i_{d1})}$
PM	$90.5 + 190.6 * i_{d1}/v_{gs1} + 22.2 * i_{d2}/v_{ds2}$
$VOFF$	$-2.0e\text{-}3$
SR_p	$2.36e{+}7{+}1.95e{+}4{*}i_{d2}/i_{d1} - 104.7/i_{d2} + 2.15e{+}9{*}i_{d2}$ $+4.63e{+}8{*}i_{d1}$
SR_n	$-5.72e{+}7{-}2.50e{+}11{*}(i_{d1} * i_{d2})/v_{gs2}$ $+5.53e{+}6{*}v_{ds2}/v_{gs2} + 109.7/i_{d1}$

One can examine the equations in more detail to gain an understanding of how design variables in the topology affect performance. For example, A_{LF} is inversely proportional to i_{dl}, the current at the OTA's differential pair. Or, SR_p is solely dependent on i_{d1} and i_{d2} and the ratio i_{d1}/i_{d2}. Or, within the design region sampled, the nonlinear coupling among the design variables is quite weak, typically only as ratios for variables of the same transistor. Or, that each expression only contains a (sometimes small) subset of design variables. Or, that transistor pairs $M1$ and $M2$ are the only devices affecting five of the six performances (within 10% error).

We now examine the CAFFEINE-generated tradeoffs between training error ϵ_{tr} (eqn. (4)) and complexity (eqn. (5)). Figure 6 illustrates. All models in the tradeoff of training error vs. complexity are shown: as complexity increases, the training error decreases. In each performance instance, CAFFEINE generates a tradeoff of about 50 different models. As expected, a zero-complexity model (i.e. a constant) has the highest training error of 10-25%. The highest-complexity models have the lowest training error, of 1-3%.

We can also examine the curves relating complexity to the number of basis functions. Recall that complexity is a function of both number of basis functions, and the complexity of each tree within each basis function. In the curves, we see that the number of basis functions usually increases with the complexity. However, sometimes complexity increases by having larger trees within existing basis functions, rather than adding more basis functions. This can be seen in the curves: as complexity increases, the number of bases temporarily levels off, or even decreases.

The testing error, ϵ_{test}, is also shown in Figure 6. We see that unlike the training error, it is not monotonically decreasing as complexity rises. This means that some less complex models are more predictive than more complex ones. However, we can prune the models down to the ones that give a tradeoff between testing error and complexity, as shown in Figure 7. These are the most interesting and useful.

It is notable that the testing error is lower than the training error in almost all cases. This sounds promising, but such behavior is rare in the regression literature, and made us

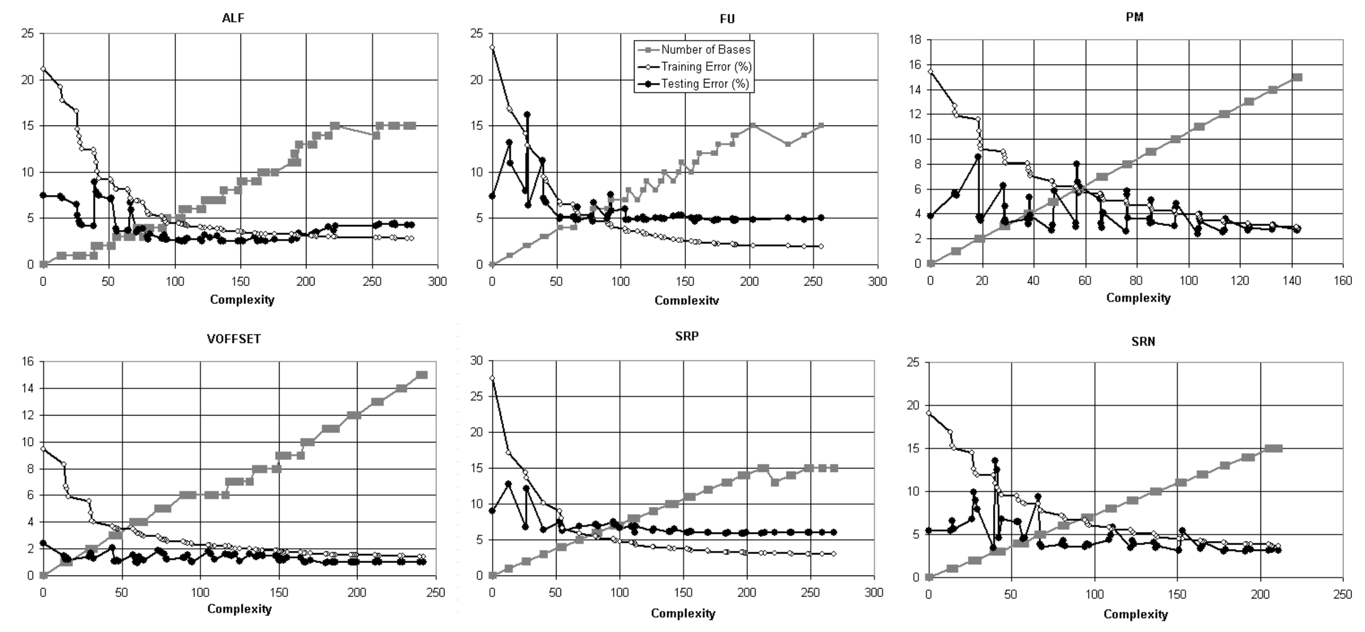

Fig. 6. Plots of models' training error, testing error, and number of bases vs. the complexity for each performance goal for the opamp of Figure 5. Every $(diamond, triangle, square)$ triplet corresponds to a symbolic model at a given complexity.

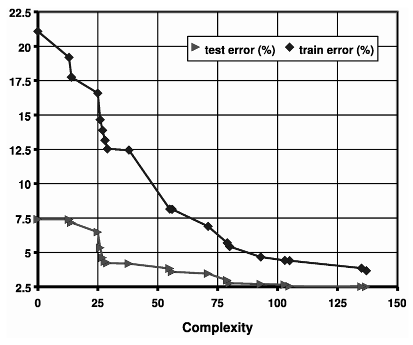

Fig. 7. Every $(diamond, triangle)$ is a symbolic model for A_{LF} like Figure 6, except filtered to only keep models on the tradeoff of *testing* error vs. complexity.

question what was happening. It turns out that there is a valid reason: recall that the training data is from extreme points of the sampling hypercube (scaled dx=0.10), and the testing data is internal to the hypercube (dx=0.03). This testing data tests the *interpolation* ability. Thus, models that really *are* predictive should be able to interpolate well, even at the cost of a perfect fit to the extreme points. In any case, validly having the testing error lower than the training error demonstrates the strength of the CAFFEINE approach.

By only putting the relevant variables into a model, the approach demonstrates the potential to provide expressions for circuits with significantly more variables (see next section).

One may improve the understanding of the basic dependencies in a circuit in another fashion: by examining expressions of varying complexity for a single performance characteristic. Low-complexity models will show the macro-effects; alterations to get improved error show how the model is refined to handle second-order effects. Table IV shows models generated for the phase margin (PM) for decreasing training and testing error. A constant of 90.2, while giving 15% training error, had only 4% test error. For better prediction, CAFFEINE injected two more basis functions; one basis being the current into the differential pair i_{d1}, the other basis, i_{d2}/v_{ds2}, being the ratio of the current to the drain-source voltage of $M2$; i.e. $M2$'s small-signal output conductance ($1/r_{out2}$). The next model turns the input current term into a ratio i_{d1}/v_{gs1}; i.e. M1's transconductance, inverted ($1/g_{m1}$). Interestingly, and reassuringly, almost all ratios use the same transistor in the numerator and denominator.

Such analyses demonstrate the core aim of CAFFEINE symbolic modeling: gaining insight into the design-performance relationship.

C. Results: Comparison to Posynomial Symbolic Models

We also compared CAFFEINE to the posynomial approach using the posynomial results in [10]. We first compare model complexity. To pick the models to compare, we first choose the CAFFEINE model which meets the reported posynomial training and test error of [10], then we compare the number of posynomial coefficients to the number of coefficients appearing in the CAFFEINE expressions (this is reasonable when the CAFFEINE expressions are largely rationals; more complex symbolic models would be less appropriate). As Figure 8 shows, the CAFFEINE models are 1.3 to 6.4 times more compact than the posynomial models. And, in $VOFF$, the only performance that the posynomials had slightly better prediction error than CAFFEINE (see Figure 9), the CAFFEINE model is 6.2x more compact.

We can also compare the prediction abilities of CAFFEINE to posynomials. To pick a model from a CAFFEINE-generated tradeoff for comparison, we fixed the training error to what the posynomial achieved, then compared the testing errors.

TABLE IV
CAFFEINE-GENERATED MODELS FOR PM OF THE OTA OF FIGURE 5, IN ORDER OF DECREASING ERROR AND INCREASING COMPLEXITY.

Test error (%)	**Train error** (%)	PM **Expression**
3.98	15.4	90.2
3.71	10.6	$90.5 + 186.6 * i_{d1} + 22.1 * i_{d2}/v_{ds2}$
3.68	10.0	$90.5 + 190.6 * i_{d1}/v_{gs1} + 22.2 * i_{d2}/v_{ds2}$
3.39	8.8	$90.1 + 156.85 * i_{d1}/v_{gs1} - 2.06e\text{-}3 * i_{d2}/i_{d1} + 0.04 * vgs2/v_{ds2}$
3.31	8.0	$91.1 - 2.05e\text{-}3 * i_{d2}/i_{d1} + 145.8 * i_{d1} + 0.04 * v_{gs2}/v_{ds2} - 1.14/v_{gs1}$
3.20	7.7	$90.7 - 2.13e\text{-}3 * i_{d2}/i_{d1} + 144.2 * i_{d1} + 0.04 * v_{gs2}/v_{ds2} - 1.00/(v_{gs1} * v_{gs3})$
2.65	6.7	$90.8 - 2.08e\text{-}3 * i_{d2}/i_{d1} + 136.2 * i_{d1} + 0.04 * v_{gs2}/v_{ds2} - 1.14/v_{gs1} + 0.04 * v_{gs3}/v_{ds5}$
2.41	3.9	$91.1 - 5.91e\text{-}4 * (v_{gs1} * i_{d2})/i_{d1} + 119.79 * i_{d1} + 0.03 * v_{gs2}/v_{ds2} - 0.78/v_{gs1} + 0.03 * v_{gs1}/v_{ds5}$ $-2.72e\text{-}7/(v_{ds2} * v_{ds5} * i_{d1}) + 7.11 * (v_{gs2} * v_{gs4} * i_{d2}) - 0.37/vsg5 - 0.58/v_{gs3} - 3.75e\text{-}6/i_{d2} - 5.52e\text{-}6/i_{d1}$

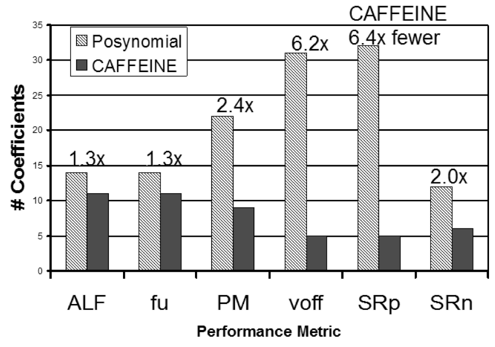

Fig. 8. Comparison of the complexity of CAFFEINE models to posynomial models [10]. Method: (1) choose CAFFEINE model that meets posynomial training and test error, then (2) compare number of coefficients.

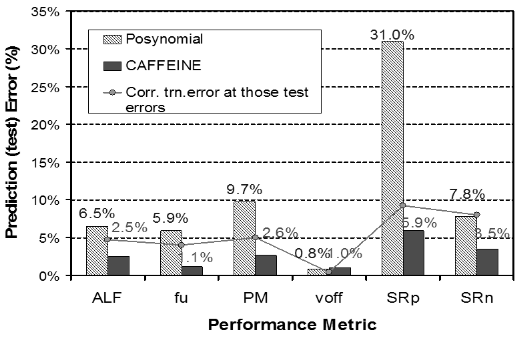

Fig. 9. Comparison of CAFFEINE testing error to posynomial testing error. The CAFFEINE model training error corresponding to the given test errors is also shown.

The results are in Figure 9. In one case, $VOFF$, CAFFEINE did not meet the posynomial training error (0.4%), although it probably could have with more basis functions; we instead picked an expression which very nearly matched the posynomial approach's testing error of 0.8%. What we saw in the previous data, and what we see again here, is that CAFFEINE has a lower testing error than training error, which provides great confidence to the models. In contrast, in all cases but $VOFF$, the posynomials had a higher testing error than training error, even on this interpolative data set. The CAFFEINE models' testing errors were *2x to 5x* lower than ones from the posynomial models. The exception is $VOFF$, where the posynomial achieves 0.8% testing error compared to 0.95% for CAFFEINE. In short, posynomials have poor prediction ability even in interpolation. CAFFEINE models predict far better, and with more compact models. Given this, one can reasonably question the trustworthiness of constraining analog circuit performance models to posynomials.

D. Results: Comparison to State-of-the-Art Blackbox Regression Approaches

While other modeling techniques may produce models that are opaque (and therefore not interpretable), it is still instructive to see how well CAFFEINE compares to them in terms of prediction ability. So, on the 6 problems already described in section VI-A, we tested the following regression techniques: a constant, linear models with least-squares fit, full quadratic models with least-squares fit, projection-based quadratic (PROBE) [13], posynomial [10], state-of-the-art feedforward neural networks (FFNN) [40], boosting the FFNNs, multivariate adaptive regression splines (MARS) (i.e. piecewise polynomial with stepwise construction) [41], least-squares support vector machines (LS-SVM) [42], and kriging [43].

Model builders were coded and configured as follows. The code to build constant, linear, and full quadratic models was about 25 lines of Matlab. The model building time was a few seconds, at most. The code to build PROBE was about 100 lines of python, using Numeric / LAPACK for least-squares regression and maximum rank of 2. Model building time was a few seconds, at most. The posynomial results were taken directly from [10]; it reports that the model building time was 1-4 minutes (on a slower machine). The target training error for the other model builders was the posynomial's training error from [10]. The FFNN is trained via an adaptive Levenberg-Marquardt optimization scheme (OLMAM); we used the Matlab code of [40]. Settings were NumRestarts = 10, MaxEpochs = 5000. The time to build a single network was about 10 s. A suitable error was typically found in the first or second restart of about 3 hidden neurons. Therefore the total model building time was about (10 s) * (10 restarts) * (first 2 neurons) + (10 s) * (2 restarts) * (1 final neuron) = 10*10*2 + 10*2 = 220 s = 3.7 min. The boosted FFNN was Matlab code wrapping the OLMAM code. Settings were NumModels = 20. Model building time was about (220 s to discover NumHid) + (10 s)*(20 models) = 220 s + 200 s = 420 s = 7.0 min. A 10x speedup via a C implementation would make this 42 s. The MARS model builder was about 500 lines of Matlab code; model building time was about 5 minutes. The SVM is trained using the least-squares strategy (LS-SVM); we used the Matlab code from [42], with all settings at "fully automatic"; model building time was about 5 minutes. The kriging model builder was about 200 lines of Matlab code, with Θ_{min} = 0.0, Θ_{max} = 10.0, p_{min} = 0.0, p_{max} = 1.99. The model building time was about 5 minutes.

Figure 10 shows the resulting test errors for the 6 performances (adapted from [25]).

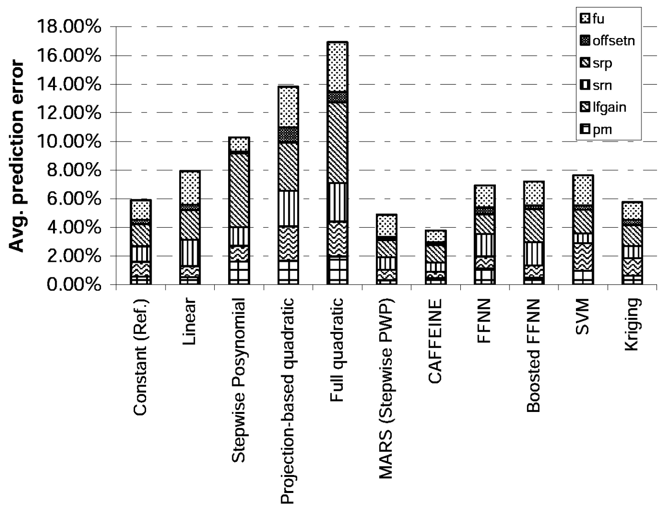

Fig. 10. Comparison of prediction ability of CAFFEINE to state-of-the-art modeling techniques.

On this dataset, CAFFEINE has the lowest average prediction error. MARS comes in very close. Kriging is the next-best. The FFNN, boosted FFNN, and SVM are all very close, and perform about the same as the linear model. The quadratic and posynomial approaches and posynomial approaches perform the worst.

The results on different regressors inform us about the nature of the data and models. Progressing across the spectrum of polynomial complexity – from the simplest linear models to posynomials to projection-based quadratic to full quadratic – the prediction error continually worsens. It turns out that the polynomials even capture the *training* error poorly; for example the projection-based quadratic had a training error of about 10% for each performance. Since the prediction error became lower the more constrained the polynomial model was, this indicates that where the models do attempt to use the added flexibility to predict better, it backfires. In general, this is indicative that a polynomial functional template is not appropriate for circuit performance mappings, even for this relatively simple OTA circuit.

CAFFEINE only selects input variables that really matter. It is biased towards the axes of the input variables rather than being affine-invariant. That is, CAFFEINE expressions and search operators work on one or a few input variables at a time, as opposed to using all variables in a weighted sum. MARS did similarly, because its stepwise-forward nature makes it also biased towards the axes and is selective of input variables. While CAFFEINE had the best or near-best prediction error on 5 of the 6 performance goals, MARS had the best or near-best on 3. As we shall see, the other approaches lose prediction performance because they have different biases.

Kriging performed fairly admirably in this setting. This is not surprising because it tends to perform well when the input samples have relatively uniform spacing, as they do here with the DOE sampling. Kriging, FFNNs, and boosted FFNNs did worse than CAFFEINE and MARS, most likely because they did not have the helpful (for this application) bias towards the input axes. The boosted FFNN did not have noticeably superior performance to the FFNN, which means that overfitting was likely not an issue with the FFNN. The SVM's performance was poor, probably because it treated the variables it selected too uniformly. Also, the support vector at the center of the sampling hypercube has to reconcile all the other samples, which it does not really have enough parameters to do properly. Because kriging did substantially better than SVMs, the choice of kernel distance function was likely not an issue. Interestingly, only three approaches, namely CAFFEINE, MARS, and kriging, did better at prediction than a constant. This is not because constants are good predictors *per se*, but because other predictors failed for the various reasons described. Put in another way, the other predictors' attempts to predict outputs from unseen (testing) inputs did poorly because the models generalized in poor directions that caused more extreme error values, whereas the constant never had extreme error values.

We can consider how well a model's structure can capture the mapping where no overfitting is present, by taking the number of training samples $N_{tr} \rightarrow \infty$. If the model can approximate arbitrary nonlinear functions, then with enough model building effort, training error $\epsilon_{tr} = 0$, and $\epsilon_{test} \rightarrow 0$ as $N_{tr} \rightarrow \infty$. The following approaches can approximate arbitrary nonlinear functions: MARS (with enough basis functions), CAFFEINE (with enough basis functions and depth), FFNN (with enough hidden nodes), boosted FFNN (with enough hidden nodes and ensemble entries), SVM (using all support vectors and a narrow kernel bandwidth), and kriging (using all training datapoints and a narrow kernel bandwidth).

VII. Scaling Up CAFFEINE: Algorithm

We ran the algorithm described in section V on larger circuits – problems with more than 100 input variables. The results were disappointing: despite good performance on smaller problems, CAFFEINE was too slow to return interesting results on these larger problems in reasonable time. That experience motivates this section. The aim is to alter the search algorithm so that it can scale to problems of 100 variables. The specific aims are to run faster (hours or minutes), yet maintain predictive and interpretable models. The improved CAFFEINE leverages four complementary techniques:

- Subtree caching [32]
- Gradient-directed regularization [33] to simultaneously prune basis functions and set coefficients for the remaining basis functions
- Filter single-variable expressions in a pre-evolution step
- Always consider all linear basis functions

We now describe each technique in detail.

A. Subtree Caching

In the original implementation of CAFFEINE, every time a tree was changed, it would have to be *fully* re-evaluated. The technique of sub-tree caching [32] sidesteps evaluations in some nodes of the tree. Given that the training dataset does not change, when a new tree is created from parent tree(s) via the search operators, only *part* of the new tree is different. Therefore, we evaluate just the nodes of the tree that have changed, and their parent nodes, and *cache* the results. The "evaluation" for other nodes merely uses the evaluated results that have been cached previously. Note that to do this cleanly, CAFFEINE was re-implemented in Python, whereas the previous implementation was in Matlab. This improved runtime further because Python passes function values by reference, whereas Matlab passes by value.

B. On-the-fly Pruning with Gradient-Directed Regularization

In previous subsections, the linear coefficients **a** of eqn. (3) were learned by minimizing the least-squares (LS) loss function on the training data. But for larger problems having potentially more basis functions, the LS predictions can be unstable because there is higher variance in the range of possible parameters. Furthermore, to keep the complexity down, it is desirable to have a more aggressive way to prune the basis functions. Regularization is promising because it explicitly accounts for parameter variance and can implicitly prune basis functions on-the-fly. Historically, the main regularization choices have been ridge regression [44] and the lasso [45]. Unfortunately, ridge regression does little pruning, and the lasso prunes *too* aggressively. Fortunately, a new technique, gradient-directed regularization (GDR) [33], strikes a compromise. GDR does gradient-based optimization on the loss function f_2 in eqn. (1) according to **a**'s update rule:

$$\boldsymbol{a} = \boldsymbol{a} + \Delta\nu * \boldsymbol{h} \tag{7}$$

where $\Delta\nu$ is a small ("infinitesimal") value and $\boldsymbol{h}$ is the direction of the next step. The starting value of $\boldsymbol{a}$ is [0, 0, ..., 0]. The gradient to the loss function is:

$$\boldsymbol{g} = -\frac{\boldsymbol{d}}{\boldsymbol{da}} \frac{1}{N_{tr}} \sum_{i=1}^{N_{tr}} L(y_i, F(\boldsymbol{x_i}; \boldsymbol{a})) \tag{8}$$

where L is given in eqn. (2). One could directly optimize using $\boldsymbol{g}$ instead of $\boldsymbol{h}$ in (7), but little pruning would happen, and collinear or near-collinear bases get similar values (like ridge regression). Instead, GDR encourages diversity by selectively updating coefficients. Specifically, it changes a_j at a given step only if $|g_j|$ is sufficiently large:

$$\boldsymbol{h} = \{h_j\}\forall j = \{\gamma_j * g_j\}\forall j; j = 1, 2, \ldots, N_B \tag{9}$$

$$\gamma_j = I(|g_j|) \geq \tau * \underset{0 \leq k \leq N_B}{max} |g_k|) \tag{10}$$

where γ_j is an indicator function that returns 0 or 1, and h_j either outputs 0 or g_j as it combines the indicator function and the gradient. τ is a parameter which controls the degree of pruning: 0.0 is like ridge regression, 1.0 is like lasso, and values in between strike a compromise.

We employ GDR here (with settings given in section VIII-A). The result is that we can have CAFFEINE individuals with a large number of basis functions, and in a single pass GDR will drive many linear coefficients to zero (i.e. prune the basis functions), and set robust values for the remaining linear coefficients. GDR is fast too: our 300-line python implementation of GDR has about the same runtime as the highly-optimized LAPACK linear LS solver [46].

C. Pre-Evolution Filtering of Single-Variable Expressions

The third scalability-improving technique focuses the search towards the most promising single-variable nonlinear expressions. It determines those expressions with the routine *ExtractUsefulExpressions()* shown in Table V, prior to the evolutionary run (i.e. right before line 2 in the procedure of Table II). *ExtractUsefulExpressions()* considers a large set of possible *single-variable* expressions at once, and extracts the most promising ones.

TABLE V
PROCEDURE EXTRACTUSEFULEXPRESSIONS()

```
Inputs: $\boldsymbol{X}$, $\boldsymbol{y}$, $\iota_{thr}$
Outputs: $\boldsymbol{B_{useful}}$
1.  $\boldsymbol{B}$ = {}; $i$ = 1
2.  for each input variable $v$ = $\{x_1, x_2, \ldots\}$
3.      for each operator $op$ = $\{unity(), log_{10}, \ldots\}$
4.          for each exponent $exp$ = {-2, -1.5, ...}
5.              define $B_i$ as $op(v)^{exp}$
6.              $\boldsymbol{B}$ = $\boldsymbol{B} \cup B_i$; $i$ = $i$ + 1
7.  $\boldsymbol{X_B}$ = simulate $\boldsymbol{X}$ on each $B_i$
8.  $\boldsymbol{a}$ = GDR linear learning on $\boldsymbol{X_B} \mapsto \boldsymbol{y}$.
9.  $\iota_i$ = compute influence of $B_i$ according to (11); for each $B_i$
10. $\boldsymbol{B}$ = sort $\boldsymbol{B}$ in descending order of $\iota_i$
11. $\boldsymbol{B_{useful}}$ = $\emptyset$; $\iota_{tot}$ = 0; $i$ = 1
12. while $\iota_{tot} < \iota_{thr}$:
13.     $\boldsymbol{B_{useful}}$ = $\boldsymbol{B_{useful}} \cup B_i$
14.     $\iota_{tot}$ = $\iota_{tot} + \iota_i$
15.     $i$ = $i$ + 1
16. return $\boldsymbol{B_{useful}}$
```

We now describe *ExtractUsefulExpressions()* in detail. It inputs the target training inputs and outputs $\{\boldsymbol{X}, \boldsymbol{y}\}$, and ι_{thr}, which we discuss later. It returns a set of chosen expressions, $\boldsymbol{B_{useful}}$. Lines 1-6 construct the candidate expressions $\boldsymbol{B}$, by enumerating through all combinations of input variables (line 2), operators (line 3), and exponents (line 4). Line 7 simulates each candidate expression on each of the training input vectors in $\boldsymbol{X}$. Each row of the resulting matrix $\boldsymbol{X_B}$ has the values of each training input vector as input to a given expression B_i. Line 8 computes the influence of each B_i via linear learning on $\boldsymbol{X_B} \mapsto \boldsymbol{y}$. Since the number of expressions may exceed the number of training samples, GDR is used because it can handle underdetermined linear systems. GDR assigns a linear coefficient a_i to each expression B_i. Line 9 computes the influence, ι_i, of expression B_i:

$$\iota_i = |a_i| * (\underset{1 \leq j \leq N}{max}(B_i(\boldsymbol{x}_j)) - \underset{1 \leq j \leq N}{min}(B_i(\boldsymbol{x}_j))) \tag{11}$$

where $\boldsymbol{x_j}$ is the j^{th} training sample. $\underset{1 \le j \le N}{max}(B_i(\boldsymbol{x}_j))$ is the largest value that B_i computes to across the training data, and $\underset{1 \le j \le N}{min}(B_i(\boldsymbol{x}_j))$ is the smallest value. Influence ι_i is an absolute and normalized version of linear coefficient a_i. Lines 10-16 use the ι_i's to select basis functions. Line 10 sorts them such that B_1 has highest influence, B_2 has second-highest, etc. Line 11 initializes the loop. Line 12 loops until the total influence quota, $iota_{thr}$, is hit; e.g. $iota_{thr}$ = 0.95 means that the highest-influence expressions totaling 95% of total influence are returned. To implement, line 13 adds the next-most influencing expression, and lines 14-15 do bookkeeping. Line 16 returns $\boldsymbol{B_{useful}}$.

These $\boldsymbol{B_{useful}}$ get stored for use during the evolutionary run. During the run, whenever a sum of products expression is about to be randomly generated (as a basis function, or at a lower level in the CAFFEINE expression tree), then κ% of the time, only the useful expressions are considered. There has to be enough opportunity to try other expressions to avoid over-constraining the search, but the majority of search effort can be focused on known-promising expressions. We set κ = 80%. Note that variable interactions can easily be generated via crossover and mutation operations on single-variable expressions. This strategy is reminiscent of MARS [41], which builds up complex multi-variable expressions from a foundation of single-variable expressions.

D. Always Include All Linear Basis Functions

This scale-up technique is based on a few observations: (a) circuit problems with a larger number of input variables tend to have at least partially linear responses to some variables, (b) GDR was very effective at pruning bases, and (c) [47] showed improved prediction by combining linear basis functions and a (non-CAFFEINE) nonlinear model. So, we altered the search to always consider all linear basis functions (but not to evolve them). Specifically, when evaluating an individual, there is a step which does linear learning to find the best coefficients for the tree-based basis functions (and the offset). We altered that step to include more basis functions – one linear basis function for each input variable. This increases the number of basis functions for linear learning, but not for the evolutionary search itself which only sees the GP trees.

VIII. Scaling Up CAFFEINE: Experimental Results

A. Experimental Setup

In this section, the aim is to determine how well the scalability goals have been achieved with the improved CAFFEINE.

The tests are on three progressively larger circuits – the operational amplifiers shown in Figures 11. The simplest is the well-known Miller opamp, and the other two are larger fully-differential opamps with more complex compensation schemes. The circuit regression problems have been set up with the parameters of Table VI. Four output performances are modeled for each circuit, with the intent to represent a cross-section of analyses and measures: A_V (gain), THD (total harmonic distortion), SR (slew rate), and OS (overshoot). The technology is 0.13 μm CMOS. The design variables are widths W, lengths L, multipliers M, capacitances C, and resistances R. The samples were taken using Latin hypercube sampling [48] on a uniform distribution in the hypercube having its center at a "good" design, and variable ranges $\pm$10%. The training and test data were split apart by sorting the samples according to the output value, allocating every 4^{th} sample to the test data, and the rest to training (i.e. 25% test data).

TABLE VI
Parameters of Circuits for the CAFFEINE Scaling Experiments.

# Variables	# Devices	# Train Samples	# Test Samples	Performances Modeled
24	10	129	32	A_V, THD, SR, OS
59	30	330	82	A_V, THD, SR, OS
129	50	1050	262	A_V, THD, SR, OS

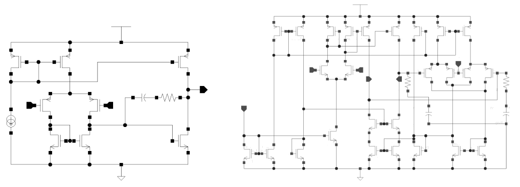
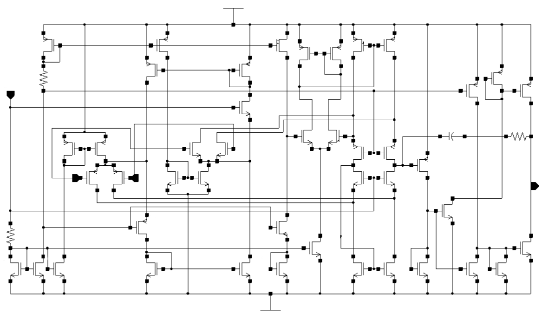

Fig. 11. Schematics of circuits for test problems. Top left: 10-device amplifier. Top right: 30-device amplifier. Bottom: 50-device amplifier.

The search strategy settings were as follows. For pre-evolution filtering: influence threshold ι_{thr} = 25%, bias to useful expressions κ = 80%. In GDR, pruning degree τ = 0.5. In CAFFEINE, all settings were like in section VI, except population size N_{pop} = 100, and maximum number of generations $N_{gen,max}$ = 50. Far fewer generations are now needed to get to reasonable results because the pre-evolution filtering picks highly useful expressions, and the linear bases are always available.

B. Experimental Results

This section aims to see how well the scalability goals were achieved on the above examples, in terms of runtime, model prediction abilities, and model interpretability. To assess the scalable CAFFEINE, we compare its models to a reference regression algorithm that has a good track record of predictive ability and of scalability: MARS [41]. To make the comparison as fair as possible, we used GDR for MARS' linear regression subroutine. A further motivation for MARS is that it was the most competitive to CAFFEINE in the experiments of section VI.

We first consider the interpretability of MARS-generated models versus CAFFEINE-generated models. We recognize that the judgement of interpretability is necessarily subjective, so here we aim to give the reader a feel. To do so, we must review MARS slightly further. Each MARS basis function is a product of "hockey stick" (HS) functions:

$$B_{MARS}(\boldsymbol{x}) = \prod_{i=1}^{N_{prod}} HS_{(i)}(x_{(i)}, t_i, q_i) \tag{12}$$

where $HS_{(i)}$ is the i^{th} HS function having either a $+$ or $-$ sign, and $x_{(i)}$, t_i, and q_i are the chosen input variable, split value, and power for $HS_{(i)}$, respectively. A HS function is:

$$HS_{\pm}(x,t,q) = \pm\begin{cases} 0 & if \quad x < t \\ (x-t)^q & if \quad x \geq t \end{cases} \tag{13}$$

To see how MARS basis functions look on real problems, we will use an arbitrarily chosen example OS, from the largest circuit (50T opamp). Table VII shows the equation for just a *single* MARS basis function. As we can see, the hockey stick functions translate to very hard-to-interpret functions.

TABLE VII
A **SINGLE** BASIS FUNCTION IN MARS-GENERATED EQUATION FOR OS PERFORMANCE METRIC OF 50-TRANSISTOR OPAMP. EACH MARS MODEL TYPICALLY HAS 3-10 BASIS FUNCTIOMS.

$$\begin{Bmatrix} 0 & if\, L_{M2} < 2.13*10^{-6} \\ (L_{M2} - 2.13*10^{-6}) & if\, L_{M2} \geq 2.13*10^{-6} \end{Bmatrix}$$
$$* \begin{Bmatrix} 0 & if\, m_{DP1M2} < 8.416*10^{-6} \\ (m_{DP1M2} - 8.416*10^{-6}) & if\, m_{DP1M2} \geq 8.416*10^{-6} \end{Bmatrix}$$
$$* \begin{Bmatrix} 0 & if\, L_{M3} < 290 \\ (L_{M3} - 290) & if\, L_{M3} \geq 290 \end{Bmatrix}$$

TABLE VIII
CAFFEINE-GENERATED EQUATION OF OS FOR THE 50-TRANSISTOR OPERATIONAL AMPLIFIER CIRCUIT. (**ALL** BASIS FUNCTIONS.)

$$\begin{aligned} &-780.8 \\ &+9.90*10^8*L_{MDP3} + 5.23*10^8*L_{MCM1} + 4.18*10^8*L_{M9} \\ &-9.27*10^8*L_{MCMB2} - 4.24*10^8*L_{M4} - 4.20*10^8*L_{M13} \\ &+11.46*m_{M2} + 7.11*m_{M17} - 8.83*m_{CM1M2} \\ &+1.14*10^8*W_{MDP3} + 7.09*10^7*W_{MCM3} \\ &+2.39*10^7*W_{M11} - 2.45*10^7*W_{M4} \\ &-8.86*10^6*log_{10}(W_{MCM3})*m_{M10}^{3/2}*W_{MCM3}^{3/2} \\ &\quad * \ (0.655*m_{CM5M1}^2 + m_{CM2M1}^{5/2}) \end{aligned}$$

We saw that even a single basis function from MARS is extremely challenging to interpret. Table VIII shows the 50T opamp OS expression that CAFFEINE generated. The model is are not as interpretable as we have seen for smaller circuits, but *some* insights can be extracted. It is notable that of the 109 input variables, CAFFEINE pruned down to just use 17 variables, i.e. about 10% of the variables. The variables include widths W, lengths L, and multipliers m. Most of the basis functions have a linear relation to OS. To decrease OS, there are some L's which need to be decreased (e.g. L_{MDP3} and $5.23 * 10^8 * L_{MCM1}$), while other L's need their values increased (e.g. L_{MCMB2} and L_{M4}). Similarly, to decrease OS with m's, some m's need decreasing and others need increasing. And similarly for W's too. There is a single base with nonlinearity. It has interactions among the variables W_{MCM3}, m_{M10}, m_{CM5M1}, and m_{CM2M1}. It is very notable that of the 109 input variables, only 4 have significant interactions (in terms of affecting OS).

Table IX summarizes the interpretability results for CAFFEINE versus MARS. In short, MARS models are definitely not interpretable, and CAFFEINE models (arguably) are, at least enough to extract some insights.

Table IX also lists the CPU time that MARS and CAFFEINE each took to build each regression model. We see that the runtime is indeed reasonable, even for the largest problems. It is far faster than the original CAFFEINE on the smaller problems.

TABLE IX
MARS AND CAFFEINE BUILD TIMES AND INTERPRETABILITY, FOR DIFFERENT PROBLEM SIZES. CIRCUIT SIMULATION TIME NOT INCLUDED.

# Variables	**Can interpret MARS model?**	**MARS build time (min)**	**Can interpret CAFFEINE model?**	**CAFFEINE build time (min)**
24	No	7	≈ Yes	20
59	No	11	≈ Yes	40
129	No	25	≈ Yes	100

We have considered the interpretability and model construction times of MARS versus CAFFEINE. What about ability to predict on unseen inputs? Table X presents the results of the regressors' prediction performance. We see that MARS and CAFFEINE have similar performance: in some cases CAFFEINE is slightly better, in other cases MARS is. We see that some problems are quite difficult to model (e.g. THD of the 10-device circuit), while other problems are quite easy (e.g. OS of the 50-device circuit).

TABLE X
PREDICTION ERROR (TESTING ERROR) OF MARS VS. CAFFEINE ON LARGER CIRCUIT MODELING PROBLEMS.

# Variables	**# Devices**	**Output**	**MARS error (%)**	**CAFFEINE error (%)**
24	10	A_V	3.52	2.95
24	10	THD	24.98	24.90
24	10	SR	0.18	0.42
24	10	OS	4.31	5.21
59	30	A_V	6.19	5.54
59	30	THD	3.53	6.85
59	30	SR	0.32	1.23
59	30	OS	6.25	6.06
109	50	A_V	3.42	3.28
109	50	THD	4.47	4.51
109	50	SR	0.90	0.92
109	50	OS	0.08	0.08

IX. OTHER APPLICATIONS

This section briefly describes other problem types that CAFFEINE has been or could be applied to.

CAFFEINE was applied to statistical modeling, to give insight into the mapping from design variables to process capability (Cpk) [49] for the 50-device circuit of Figure 11. It followed the same methodology as performance modeling, except the Cpk is computed from SPICE simulation data having both local and global process variation. Table XI shows the extracted CAFFEINE equation, which has 6.3% testing error. Note that the technology variations are embedded in the numerical coefficients of the model – Cpk is not a function of these process parameters, only their aggregate effect on the design variables. We see that just 5 variables are needed to get 6.3% test error: C_c, W_{dp2}, W_{dp1}, W_{dp2}, W_{mt4}, W_{mt1}. The variables comprise one compensation capacitor and four widths, and no lengths nor multipliers. There are significant nonlinear interactions among the variables. An increase to W_{mt4} will increase Cpk, as will a decrease to W_{mt1}. Cpk is quite dependent on the square root of C_c. Cpk can also be increased by increasing W_{dp2} (big effect) or increasing W_{dp1} (much smaller effect).

TABLE XI
CAFFEINE-GENERATED EQUATION OF CPK FOR 50-DEVICE AMP.

$$\begin{array}{l}
+1231.4 + 4.21 * 10^6 * W_{mt4}^2 / W_{mt1} - 0.0012/\sqrt{C_c} \\
-9.39 * 10^8 * W_{dp2}^2 * \sqrt{W_{dp1}} * min(0.104, 6.60 * 10^7 - 76.9/\sqrt{C_c}) \\
+1.21 * 10^{12}/min(-4.96 * 10^6, 10^{10} - 2.48 * 10^5/(\sqrt{W_{dp2}} * C_c))
\end{array}$$

CAFFEINE has also used to extract behavioral models [50]. Despite much progress in automated behavioral modeling, manual design of models remains popular because humans can leverage their insights and vouch for the final model. CAFFEINE bridges manual and automated design by offering behavioral model "suggestions" to guide the modeling expert.

In [26], CAFFEINE extracted whitebox models relating performance-tradeoff objectives. The input dataset contained 1576 Pareto-optimal points in five objectives. One objective was set as the target output, and the other four became model input variables. The extracted model with 4.1% training error was: $GBW = 4.48 + 24.9/\sqrt{A_{LF}} - (8.60 * 10^6)/(A_{LF}^2 * \sqrt{SR})$.

Future CAFFEINE applications include the following. First is to extract the (nonlinear) sensitivity of the model output relative to each input variable, in a flow similar to [26]. Note that other sensitivity-extraction approaches have been proposed, but they were specific to the model template; e.g. quadratics [13] or LVR [22], [23]. A second application is based on how LVR approaches plot the mapping from an LVR's first affine transform $\boldsymbol{w_1} * \boldsymbol{x}$ to the output $y = f_1(\boldsymbol{w_1} * \boldsymbol{x})$. A special case of CAFFEINE could be put into an LVR framework for the same insight, via: (a) evolving just one basis function at a time, using the previous basis function's residuals as the target y, and (b) restricting the depth of CAFFEINE trees so that only linear models of the form $\boldsymbol{w} * \boldsymbol{x}$ are within each f_{nonlin} basis function.

X. CONCLUSION

This paper presented CAFFEINE, a tool that can generate interpretable symbolic models of nonlinear analog circuit performances as a function of the circuit's design variables, without *a priori* requiring a model template. The keys to CAFFEINE are: a flow using SPICE simulation data, multi-objective GP search to extract template-free functions from the simulation data, and canonical-form constraints on the functions for interpretability. In the first round of experiments, visual inspection of the models has demonstrated that the models are interpretable. The performance models were also shown to be significantly more compact than posynomials. The CAFFEINE models also had lower average prediction error than posynomials, projection-based polynomials, support vector machines, MARS splines, neural networks, and boosted neural networks. CAFFEINE has also demonstrated promise in applications like robustness modeling, behavioral modeling, and tradeoff modeling. This paper also described techniques to scale up CAFFEINE to handle more input variables: subtree caching, gradient-directed regularization to prune during linear learning, pre-filtering single-variable expressions, and always considering linear bases.

REFERENCES

[1] R.A. Rutenbar, G.G.E. Gielen, and J. Roychowdhury, "Hierarchical modeling, optimization and synthesis for system-level analog and RF designs," in *Proc. of the IEEE*, 95(3), March 2007, pp. 640-669.

[2] G.G.E. Gielen, "Techniques and applications of symbolic analysis for analog integrated circuits: a tutorial overview", in *Computer Aided Design of Analog Integrated Circuits And Systems,* R.A. Rutenbar et al., eds., IEEE, 2002, pp. 245–261.

[3] G.G.E. Gielen, H. Walscharts, and W.M.C. Sansen, "ISAAC: A symbolic simulator for analog integrated circuits," in *IEEE Journ. Solid-State Circuits* 24(6), Dec. 1989, pp. 1587–1597.

[4] A. Manthe, Z. Li, and C.-J. Richard Shi, "Symbolic analysis of analog circuits with hard nonlinearity", in *Proc. Design Autom. Conf.*, 2003.

[5] J. Yang et al, "Hierarchical symbolic piecewise-linear circuit analysis", in *Proc. Behavioral Modeling and Simulation*, 2005.

[6] N. Dong and J. Roychowdhury, "General-purpose nonlinear model order reduction using piecewise polynomial representations", in *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, 27(2), 2008, pp. 249–264.

[7] P. Drennan, M. Kniffin, and D. Locascio, "Implications of proximity effects for analog design", *Proc. Custom Integr. Circ. Conf.*, 2006.

[8] H.E. Graeb, *Analog design centering and sizing.* Springer, 2007.

[9] X. Li and H. Liu, "Statistical regression for efficient high-dimensional modeling of analog and mixed-signal performance variations," in *Proc. Design Autom. Conf.*, 2008, pp. 38–43.

[10] W. Daems, G.G.E. Gielen, and W.M.C. Sansen, "An efficient optimization-based technique to generate posynomial performance models for analog integrated circuits", in *Proc. Design Autom. Conf.*, 2002.

[11] W. Daems, G.G.E. Gielen, and W.M.C. Sansen, "Simulation-based generation of posynomial performance models for the sizing of analog integrated circuits", in *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, 22(5), May 2003, pp. 517–534.

[12] V. Aggarwal and U.-M. O'Reilly, "Simulation-based reusable posynomial models for MOS transistor parameters," in *Proc. Des. Autom. Test Europe Conf.*, 2007, pp. 69–74.

[13] X. Li, P. Gopalakrishnan, Y. Xu, and L. Pileggi, "Robust analog/RF circuit design with projection-based performance modeling", in *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, January 2007.

[14] Z. Feng and P. Li, "Performance-oriented statistical parameter reduction of parameterized systems via reduced rank regression," in *Proc. Intern. Conf. Comput.-Aided Design*, 2006, pp. 868–875.

[15] G. Wolfe and R.Vemuri, "Adaptive sampling and modeling of analog circuit performance parameters with pseudo-cubic splines," in *Proc. Intern. Conf. Comput.-Aided Design*, 2004

[16] P. Vancorenland, G. Van der Plas, M. Steyaert, G.G.E. Gielen, and W.M.C. Sansen, "A layout-aware synthesis methodology for RF circuits", in *Proc. Intern. Conf. Comput.-Aided Des.*, Nov. 2001, p.358.

[17] G. Wolfe and R.Vemuri, "Extraction and use of neural network models in automated synthesis of operational amplifiers," in *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, 22(2), Feb. 2003, pp. 198–212.

[18] H. Liu, A. Singhee, R.A. Rutenbar, and L.R. Carley, "Remembrance of circuits past: macromodeling by data mining in large analog design spaces", in *Proc. Design Autom. Conf.*, 2002, pp. 437–442.

[19] F. De Bernardinis, M.I. Jordan, and A. L. Sangiovanni-Vincentelli, "Support vector machines for analog circuit performance representation", in *Proc. Design Autom. Conf.*, 2003, pp. 964–969.

[20] T. Kiely and G.G.E. Gielen, "Performance modeling of analog integrated circuits using least-squares support vector machines", in *Proc. Des. Autom. and Test Europe Conf.*, 2004, pp. 448–453.

[21] M. Ding and R. Vemuri, "A two-level modeling approach to analog circuit performance macromodeling", in *Proc. Des. Autom. and Test Europe Conf.*, 2005, pp. 1088–1089.

[22] A. Singhee and R.A. Rutenbar, "Beyond low-order statistical response surfaces: latent variable regression for efficient, highly nonlinear fitting," in *Proc. Design Autom. Conf.*, 2007.

[23] X. Li and Y. Cao, "Projection-based piecewise-linear response surface modeling for strongly nonlinear VLSI performance variations," in *Proc. Intern. Symp. Quality Electronic Design*, 2008, pp. 108–113.

[24] G. Yu and P. Li, "Yield-aware analog integrated circuit optimization using geostatistics motivated performance modeling," in *Proc. of the Intern. Conf. on Comput. Aided Design*, pp. 464–469, 2007.

[25] T. McConaghy and G.G.E. Gielen, "Analysis of simulation-driven numerical performance modeling techniques for application to analog circuit optimization", in *Proc. Intern. Symp. Circ. and Syst.*, May 2005.

[26] T. McConaghy, P. Palmers, G.G.E. Gielen, and M. Steyaert, "Automated extraction of expert knowledge in analog topology selection and sizing," in *Proc. Intern. Conf. Comput.-Aided Design*, San Jose, November 2008.

[27] T. McConaghy, T. Eeckelaert, and G.G.E. Gielen, "CAFFEINE: template-free symbolic model generation of analog circuits via canonical form functions and genetic programming", in *Proc. Des. Autom. and Test Europe Conf.*, March 2005.

[28] John R. Koza, *Genetic Programming* MIT Press, 1992.

[29] P. A. Whigham, "Grammatically-based genetic programming", in *Proc. Workshop on Genetic Progr.*, J.R. Rosca, ed., 1995.

[30] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, 6(2), 2002, pp. 182–197.

[31] M. O'Neill and C. Ryan. *Grammatical evolution: evolutionary automatic programming in an arbitrary language*. Kluwer, Norwell, 2003.

[32] M. Keijzer, "Alternatives in subtree caching for genetic programming", in *Proc. European Conf. in Genetic Programming*, 2004, pp. 328–337.

[33] J.H. Friedman and B.E. Popescu, "Gradient directed regularization for linear regression and classification", in *Stanford Univ. Dept. Statistics*, Tech. Report, Feb. 2004.

[34] D.C. Montgomery, *Design and analysis of experiments*, 6^{th} edition John Wiley & Sons, NY, NY, USA. ISBN: 047148735X, 2004.

[35] T. Soule and R.B. Heckendom, "An analysis of the causes of code growth in genetic programming", in *Genetic Programming and Evolvable Machines*, 3(3), September 2002, pp. 283–309.

[36] E. Kirshenbaum and H.J. Suermondt, "Using genetic programming to obtain a closed-form approximation to a recursive function", in *Proc. Genetic and Evol. Comput. Conf.*, 2005, pp. 543–556.

[37] P. Wambacq et al, "Efficient symbolic computation of approximate small-signal characteristics", in *IEEE Journ. Solid-State Circuits*, 30(3), March 1995, pp. 327–330.

[38] X. Yao, Y. Liu, and G. Lin, "Evolutionary programming made faster," *IEEE Trans. Evol. Comput.*, 3(2), July 1999, pp. 82–102.

[39] F. Leyn, G.G.E. Gielen, and W.M.C. Sansen, "An efficient dc root solving algorithm with guaranteed convergence for analog integrated CMOS circuits", in *Proc. Intern. Conf. Comput.-Aided Des.*, 1998.

[40] N. Ampazis and S.J. Perantonis, "Two highly efficient second order algorithms for training feedforward networks," in *IEEE Trans. Neural Networks*, 13(5), Sept. 2002, pp. 1064–1074.

[41] J.H. Friedman, "Multivariate adaptive regression splines", in *Annals of Statistics*, 19, March 1991, pp. 1–141.

[42] J.A.K. Suykens and J.Vandewalle, *Least Squares Support Vector Machines*. World Scientific Pub. Co., Singapore, 2002.

[43] D.R. Jones, M. Schonlau, and W.J. Welch, "Efficient global optimization of expensive black-box functions", in *Journ. Glob. Optim.*, 13(4), 1998.

[44] A.E. Horel and R.W. Kennard, "Ridge regression: biased estimation for nonorthogonal problems", in *Technometrics*, 12, 1970, pp. 56–67.

[45] R. Tibshirani, "Regression shrinkage and selection via the lasso", in *J. Royal. Statist. Soc B*, 58 (1), 1997, pp. 267–288.

[46] "LAPACK": Linear Algebra PACKage, http://netlib.org/lapack, last accessed Dec 22, 2008.

[47] J. H. Friedman and B.E. Popescu, "Predictive learning via rule ensembles", in *Stanford Univ. Dept. Statistics*, Tech. Report, Feb. 2005.

[48] M.D. McKay, W.J. Conover, and R.J. Beckman, "A comparison of three methods for selecting values of input variables in the analysis of output from a computer code", *Technometrics*, 21, 1979, pp. 239–245.

[49] National Institute of Standards and Technology, "What is process capability?" in *NIST / SEMATECH e-Handbook of Statistical Methods*, sec 6.1.6. http://www.itl.nist.gov/div898/handbook/pmc/section1/pmc16.htm, last accessed Dec 22, 2008.

[50] T. McConaghy and G.G.E. Gielen, "Automation in mixed-signal design: challenges and solutions in the wake of the nano era", in *Proc. Inter. Conf. on Computer-Aided Design*, 2006.

[50] T. McConaghy and G.G.E. Gielen, "IBMG: Interpretable behavioral model generator for nonlinear analog circuits via canonical form functions and genetic programming", *Proc. Intern. Symp. Circ. Syst.*, 2005.

Trent McConaghy (S'95-M'99) is co-founder and Chief Scientific Officer of Solido Design Automation Inc. He was a co-founder and Chief Scientist of Analog Design Automation Inc., which was acquired by Synopsys Inc. in 2004. Prior to that, he did research for the Canadian Department of National Defense. He received his PhD degree in Electrical Engineering from the Katholieke Universiteit Leuven, Belgium, in 2008. He received a Bachelor's in Engineering (with great distinction), and a Bachelor's in Computer Science (with great distinction), both from the University of Saskatchewan, Canada, in 1999. He has about 40 peer-reviewed technical papers and patents granted / pending. He has given invited talks / tutorials at many labs, universities, and conferences such as JPL, MIT, ICCAD, and DAC. He is regularly a technical program committee member and reviewer in both the CAD and intelligent systems fields, such as IEEE Trans CAD, ACM TODAES, Electronics Letters, to IEEE Trans. Evol. Comp, the Journal of GP and Evolvable Machines, GPTP, GECCO, ICES, etc. His research interests are statistical machine learning and intelligent systems, with transistor-level CAD applications of variation-aware design, analog topology design, automated sizing, and knowledge extraction.

Georges G.E. Gielen (S'87-M'92-SM-'99-F'02) received the MSc and PhD degrees in Electrical Engineering from the Katholieke Universiteit Leuven, Belgium, in 1986 and 1990, respectively. He currently is a Full Professor at the Katholieke Universiteit Leuven. His research interests are in the design of analog and mixed-signal integrated circuits, and especially in analog and mixed-signal CAD tools and design automation (modeling, simulation and symbolic analysis, analog synthesis, analog layout generation, analog and mixed-signal testing). He is coordinator or partner of several (industrial) research projects in this area, including several European projects (EU, MEDEA, ESA). He has authored or coauthored five books and more than 300 papers in edited books, international journals and conference proceedings. He regularly is a member of the Program Committees of international conferences (DAC, ICCAD, ISCAS, DATE, CICC...), and served as General Chair of the DATE conference in 2006 and of the International Conference on Computer-Aided Design in 2007. He serves regularly as member of editorial boards of international journals (IEEE Transactions on Circuits and Systems, Springer international journal on Analog Integrated Circuits and Signal Processing, Elsevier Integration). He received the 1995 Best Paper Award in the John Wiley international journal on Circuit Theory and Applications, and was the 1997 Laureate of the Belgian Royal Academy on Sciences, Literature and Arts in the discipline of Engineering. He received the 2000 Alcatel Award from the Belgian National Fund of Scientific Research for his innovative research in telecommunications, and won the DATE 2004 Best Paper Award. He is a Fellow of the IEEE, served as elected member of the Board of Governors of the IEEE Circuits And Systems (CAS) society and as chairman of the IEEE Benelux CAS chapter. He served as the President of the IEEE Circuits And Systems (CAS) Society in 2005. He was elected DATE Fellow in 2007, and received the IEEE Computer Society Outstanding Contribution Award and the IEEE Circuits and Systems Society Meritorious Service Award in 2007.