Towards Parallel Boolean Function Synthesis

S. Akshay[1], Supratik Chakraborty[1], Ajith John[2], Shetal Shah[1]

[1] IIT Bombay, India
[2] BARC, India

presented earlier this year at TACAS

Dec 6-8, SAT-SMT Workshop, 2017

Boolean Function Synthesis

- Boolean Functions: fundamental building blocks in computing
- Easy to specify them *declaratively*; as a relation between input and output values.
- But we often need them *constructively*
  - output specified as a *function* of the inputs
- Deriving a boolean function from a boolean relation - Boolean Function Synthesis

Boolean Function Synthesis

Problem Statement

- *Given*: a boolean relation $R(x_1, \ldots x_n, y_1, \ldots y_m)$
  where each x_i is an *input* variable
  and each y_i is an *output* variable

Boolean Function Synthesis

Problem Statement

- *Given*: a boolean relation $R(x_1, \ldots x_n, y_1, \ldots y_m)$
  where each x_i is an *input* variable
  and each y_i is an *output* variable
- Synthesize: functions $F_i(x_1, \ldots, x_n)$, for each y_i such that

  $\exists y_1, \ldots, y_m R(x_1, \ldots x_n, y_1, \ldots y_m) \equiv R(x_1, \ldots x_n, F_1, \ldots, F_m)$

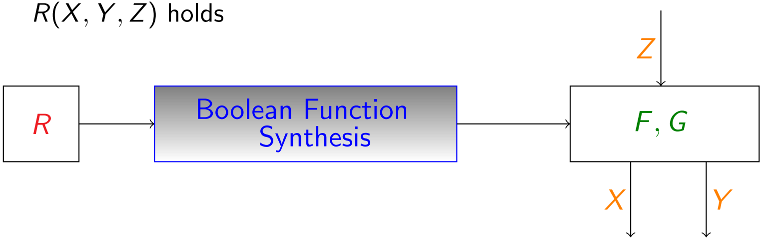
Boolean Function Synthesis

Problem Statement

- *Given*: a boolean relation $R(x_1, \ldots x_n, y_1, \ldots y_m)$
  where each x_i is an *input* variable
  and each y_i is an *output* variable
- Synthesize: functions $F_i(x_1, \ldots, x_n)$, for each y_i such that
  $\exists y_1, \ldots, y_m R(x_1, \ldots x_n, y_1, \ldots y_m) \equiv R(x_1, \ldots x_n, F_1, \ldots, F_m)$
- F_i is also called a *Skolem Function*
- R need not be true for every combination of $x_1, \ldots x_n$

Factorization using functional synthesis

- Given a relation $R(X, Y, Z)$, set of all triples (X, Y, Z) s.t., $Z = X * Y, X \neq 1, Y \neq 1$.
- Our goal: To synthesize functions F, G, s,t., $F(Z) = X$, $G(Z) = Y$, $R(X, Y, Z)$ holds



Applications of Boolean Function Synthesis

1. Factorization: Useful as a motivating example but a hard problem for boolean function synthesis!
2. Synthesizing Arithmetic Functions: from specifications of arithmetic relations
   - Example: floor, min, max, avg, ceil
3. Quantifier Elimination in Model Checking
4. Certifying Solvers: certificates for satisfiable quantified Boolean formulas (QBF)
5. Disjunctive Decomposition: compute a disjunctive decomposition of implicitly specified state transition graphs of sequential circuits.
6. Circuit Synthesis: automatically synthesizing circuits from specifications – New area: Reactive Circuit Synthesis.

Existing Approaches

1. Extract Skolem function from the proof of validity of $\forall X \exists Y F(X, Y)$
   - succinct Skolem functions [Ben05], [JB11], [JBS$^+$07], [HSB14], [RS16]
   - not applicable when $\forall X \exists Y F(X, Y)$ is not valid
   - [RS16] latest version works for formulae which are not valid
2. Generate Skolem functions matching a given template.
   - Template-based program verification and program synthesis by Srivastava, Gulwani, and Foster [SGF13]
   - effective when the set of candidate Skolem functions is known and small
   - it is not always reasonable assumption
3. Composition based approaches [Jia09], [Tri03]
   - Work well for small-sized formulas
   - Compositions cause formula blow up and memory out

Existing Approaches Contd...

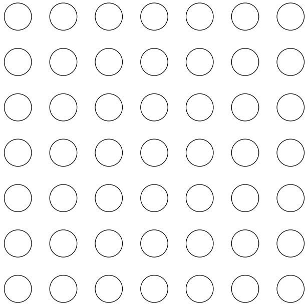
4. Boolean Function Synthesis Using BDDs [FTV16]
    - scales for a class of benchmarks with predetermined orders
    - Without prior knowledge of benchmark classes; good variable orders, performance can degrade considerably
    - Recent work in FMCAD 2017 address factored formulae
5. CEGARSKOLEMGEN [JSC+15]
    - Considers *factored formulas* wherein a formula is represented by a conjunction of factors
    - Scales well if each factor contains a small subset of variables
    - Does not perform well on large benchmarks which are not a conjunction of factors

Our Contributions

- Extend [JSC$^+$15] to arbitrary boolean formulae
- A new compositional approach to synthesize functions.
- Capitalize on compositionality to enable parallelism
- Outperforms existing techniques in terms of the number of benchmarks solved and the time taken to synthesize boolean functions

Some Basic Ideas

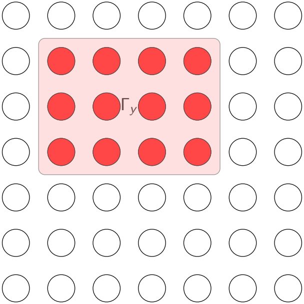
Find $F(X)$ such that $\exists y \varphi(X, y) \equiv \varphi(X, F(X))$.



— Set of all valuations to X.

Some Basic Ideas

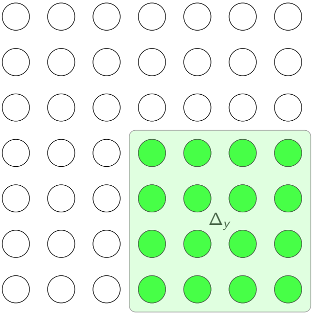
Find $F(X)$ such that $\exists y \varphi(X, y) \equiv \varphi(X, F(X))$.



—Can't set y to 1 to satisfy φ: $\Gamma_y(X) = \neg\varphi(X, y)[y \mapsto 1]$

Some Basic Ideas

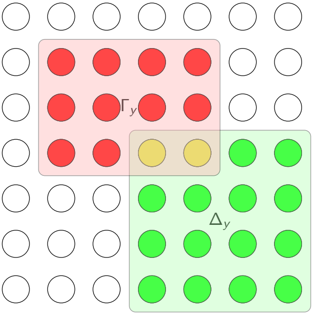
Find $F(X)$ such that $\exists y \varphi(X, y) \equiv \varphi(X, F(X))$.



— Can't set y to 0 to satisfy φ: $\Delta_y(X) = \neg\varphi(X, y)[y \mapsto 0]$

Some Basic Ideas

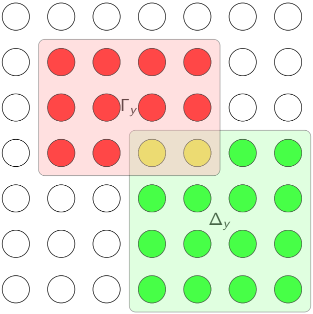
Find $F(X)$ such that $\exists y \varphi(X, y) \equiv \varphi(X, F(X))$.



— Can't set y to 1 to satisfy φ: $\Gamma_y(X) = \neg\varphi(X, y)[y \mapsto 1]$
— Can't set y to 0 to satisfy φ: $\Delta_y(X) = \neg\varphi(X, y)[y \mapsto 0]$

Some Basic Ideas

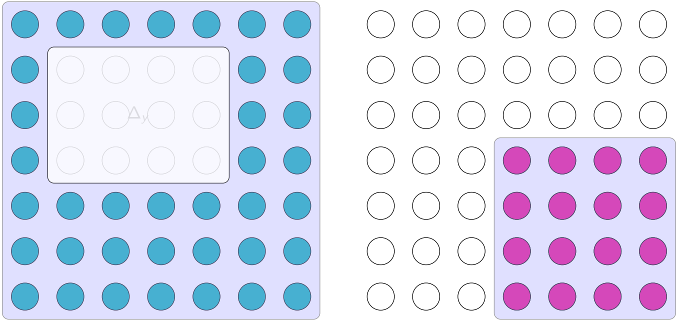
Find $F(X)$ such that $\exists y \varphi(X, y) \equiv \varphi(X, F(X))$.



— Can't set y to 1 to satisfy φ: $\Gamma_y(X) = \neg\varphi(X, y)[y \mapsto 1]$
— Can't set y to 0 to satisfy φ: $\Delta_y(X) = \neg\varphi(X, y)[y \mapsto 0]$
— A Skolem function for y in φ is any Interpolant of $(\Delta_y \setminus \Gamma_y)$ and $(\Gamma_y \setminus \Delta_y)$

Some Basic Ideas

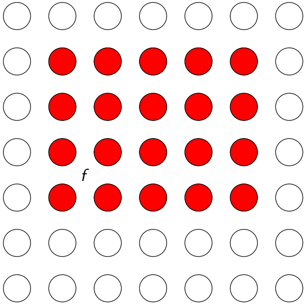
Find $F(X)$ such that $\exists y \varphi(X, y) \equiv \varphi(X, F(X))$.



— Can't set y to 1 to satisfy φ: $\Gamma_y(X) = \neg\varphi(X, y)[y \mapsto 1]$
— Can't set y to 0 to satisfy φ: $\Delta_y(X) = \neg\varphi(X, y)[y \mapsto 0]$
— A Skolem function any Interpolant of $(\Delta_y \setminus \Gamma_y)$ and $(\Gamma_y \setminus \Delta_y)$
— E.g. $\neg\Gamma_y = \varphi(X, y)[y \mapsto 1] = \varphi(X, 1)$
— and $\Delta_y = \neg\varphi(X, y)[y \mapsto 0] = \neg\varphi(X, 0)$

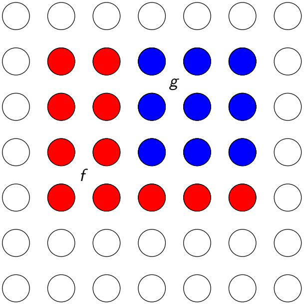
Notation: Abstraction Refinement

- Given propositional functions $f(X)$ and $g(X)$



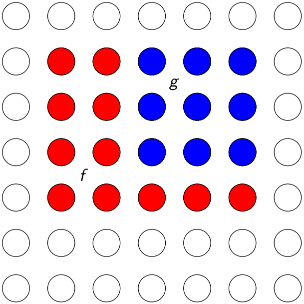
Notation: Abstraction Refinement

- Given propositional functions $f(X)$ and $g(X)$



Notation: Abstraction Refinement

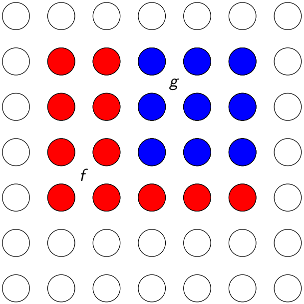
- Given propositional functions $f(X)$ and $g(X)$



If $g \implies f$ then f is an abstraction of g and that g is a refinement of f

Notation: Abstraction Refinement

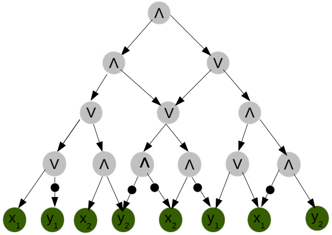
- Given propositional functions $f(X)$ and $g(X)$



If $g \implies f$ then f is an abstraction of g and that g is a refinement of f

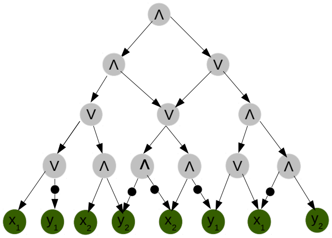
- Similarly, γ_i is a refinement of Γ_i, i.e., $\gamma_i \implies \Gamma_i$
- and δ_i is a refinement of Δ_i, i.e., $\delta_i \implies \Delta_i$

Using Compositionality



- *Input*: DAG representing $\varphi(X, Y)$ in NNF Form
  - Internal nodes tagged as AND/OR; negations pushed to leaves

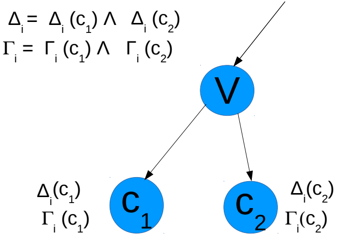
Using Compositionality



- Can the DAG representation be exploited to obtain compositionality?
- i.e., can we compose Δ_i (δ_i) and Γ_i (γ_i) sets of a (operator) in terms of Δ_i (δ_i), Γ_i (γ_i) of its children?

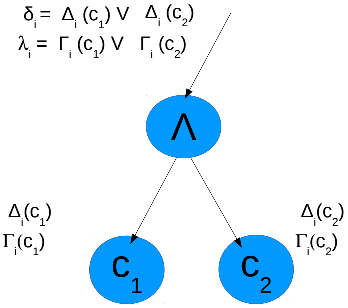
Compositionality for an OR node

For an OR node N, with children c_1 and c_2, for each y_i,



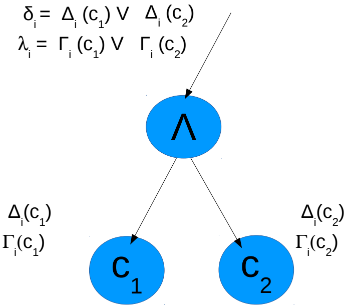
Compositionality for an AND node

For an AND node N, with children c_1 and c_2, for each y_i,

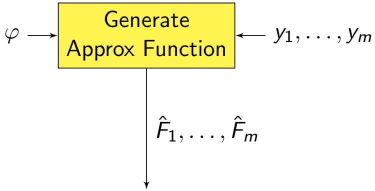


Compositionality for an AND node

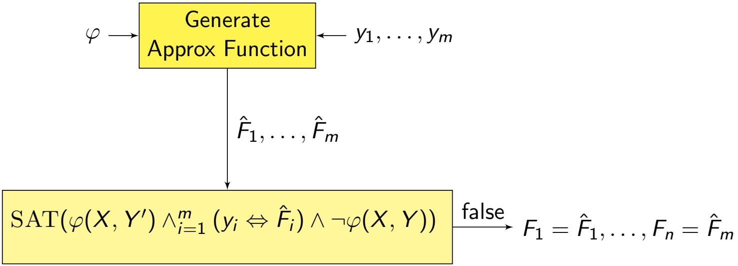
For an AND node N, with children c_1 and c_2, for each y_i,
To obtain $\Delta_i(N)$ and $\Gamma_i(N)$, we may need to perform CEGAR



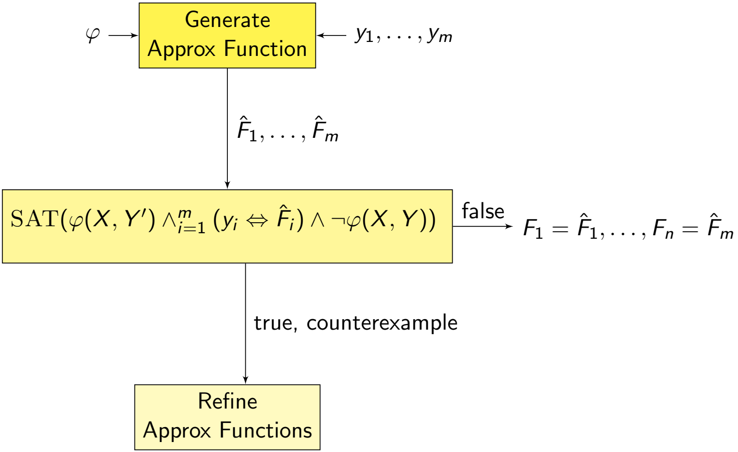
Counter-Example Guided Abstraction Refinement (CEGAR)



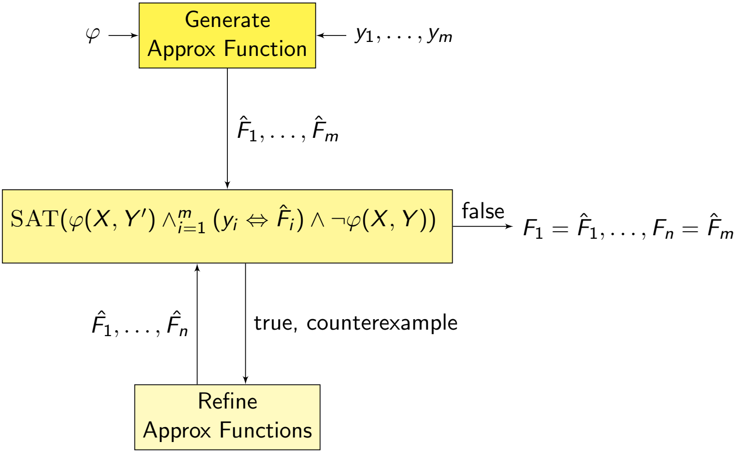
Counter-Example Guided Abstraction Refinement (CEGAR)



Counter-Example Guided Abstraction Refinement (CEGAR)



Counter-Example Guided Abstraction Refinement (CEGAR)



Generalized Compositional Lemma and Parallelism

Generalized Compositional Lemma (Details in the paper)

For any boolean operator op with children $c_1, \ldots, c_k$

- Generalized Compositional Lemma indicates how δ_i's and γ_i's of $c_1, \ldots, c_k$ can be combined to obtain $\delta_{\mathsf{op}}, \gamma_{\mathsf{op}}$
- Allows us to work with refinements(δ_i and γ_i); exact Δ_i and Γ_i not necessary
- Computation of exact Δ_i and Γ_i required at the root
- Allows us to compute better refinements

Generalized Compositional Lemma and Parallelism

Generalized Compositional Lemma (Details in the paper)

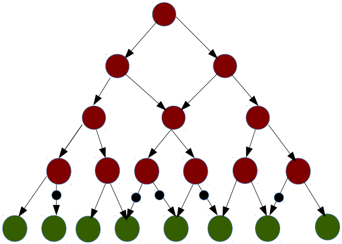
For any boolean operator op with children $c_1, \ldots, c_k$

- Generalized Compositional Lemma indicates how δ_i's and γ_i's of $c_1, \ldots, c_k$ can be combined to obtain $\delta_{\text{op}}, \gamma_{\text{op}}$
- Allows us to work with refinements(δ_i and γ_i); exact Δ_i and Γ_i not necessary
- Computation of exact Δ_i and Γ_i required at the root
- Allows us to compute better refinements

Exploiting Compositionality to enable Parallelism

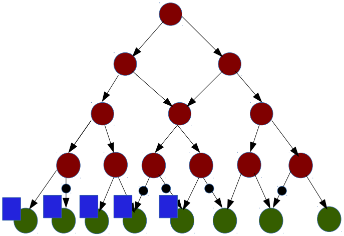
All nodes whose children's Δ_i and Γ_i sets are computed are candidates for processing in parallel

Exploiting Compositionality to enable Parallelism



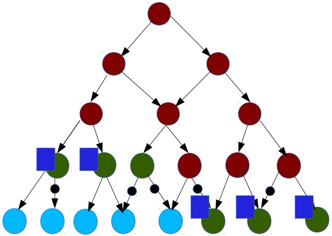
At the beginning: Identify nodes that can be processed in parallel, namely, leaf nodes

Exploiting Compositionality to enable Parallelism



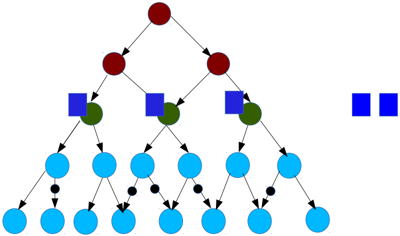
Execution of nodes in parallel; #cores = 5

Exploiting Compositionality to enable Parallelism



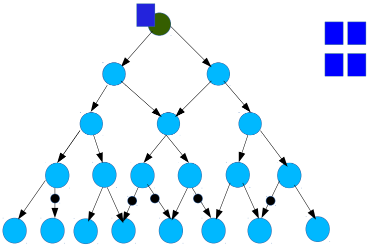
Once all children of a node are processed, it becomes candidate for processing

Exploiting Compositionality to enable Parallelism



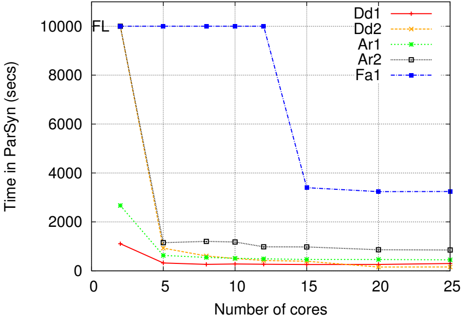
As we near the root of the DAG, fewer nodes can be processed in parallel

Exploiting Compositionality to enable Parallelism



Once root, R, is processed, return $\neg\Gamma_1(R), \ldots, \neg\Gamma_m(R)$

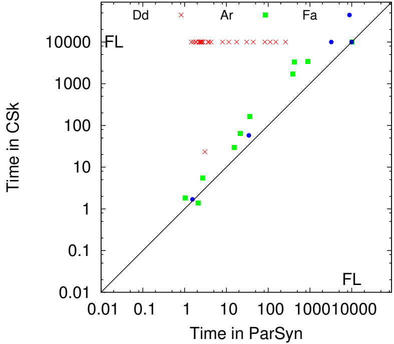
Experiment Results: ParSyn with different cores



- As #cores increases to 10-15, computation time decreases
- After a point, performance does not improve further
- Need to improve the parallelism of the algorithm

Experiment Results: ParSyn Vs Csk (based on [JSC+15])

\# Disjunctive Decomposition Benchmarks = 27;
\# Arithmetic Benchmarks = 15; #Factorization Benchmarks: 4;
Total Benchmarks = 46

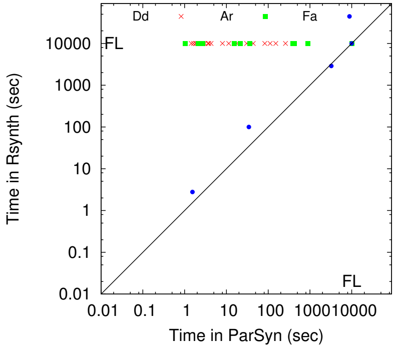


\#cores used for ParySyn = 20

- Csk was successful on only 12 of the 46 benchmarks; Most of the benchmarks with Csk was successful on were conjunctions of factors
- ParSyn was successful on 39 benchmarks

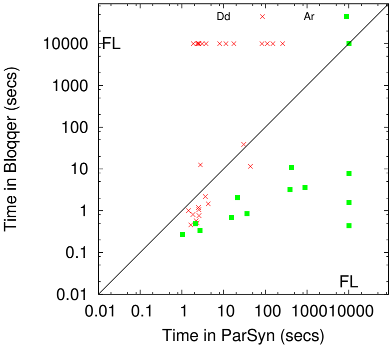
ParSyn Vs RSynth

Variable ordering: variable which occurs in the least number of transitive fan-ins



- ParSyn was on successful 39 of the 46 benchmarks
- RSynth was successful on only 3 benchmarks

Experiment Results: ParSyn Vs Bloqqer



- Compared only instances where $\exists Y F(X, Y)$ is valid - 42 benchmarks
- Bloqqer successfully synthesized functions for 25, gave a Not Verified message for 17
- ParSyn was successful on 36 benchmarks

Conclusions and List of Publications

- A CEGAR approach for boolean function synthesis for arbitrary boolean formulae
- A first step towards parallelization: can we do better?

Marco Benedetti.
sKizzo: A Suite to Evaluate and Certify QBFs.
In *Proc. of CADE*, pages 369–376. Springer-Verlag, 2005.

Dror Fried, Lucas M. Tabajara, and Moshe Y. Vardi.
BDD-based boolean functional synthesis.
In *CAV*, 2016.

Marijn Heule, Martina Seidl, and Armin Biere.
Efficient Extraction of Skolem Functions from QRAT Proofs.
In *Proc. of FMCAD*, 2014.

J.-H. R. Jiang and V Balabanov.
Resolution proofs and Skolem functions in QBF evaluation and applications.
In *Proc. of CAV*, pages 149–164. Springer, 2011.

T. Jussila, A. Biere, C. Sinz, D. KrÃ¶ning, and C. Wintersteiger.
A First Step Towards a Unified Proof Checker for QBF.
In *Proc. of SAT*, volume 4501 of *LNCS*, pages 201–214