Boolean Functional Synthesis via Self-Substitution

Lucas M. Tabajara

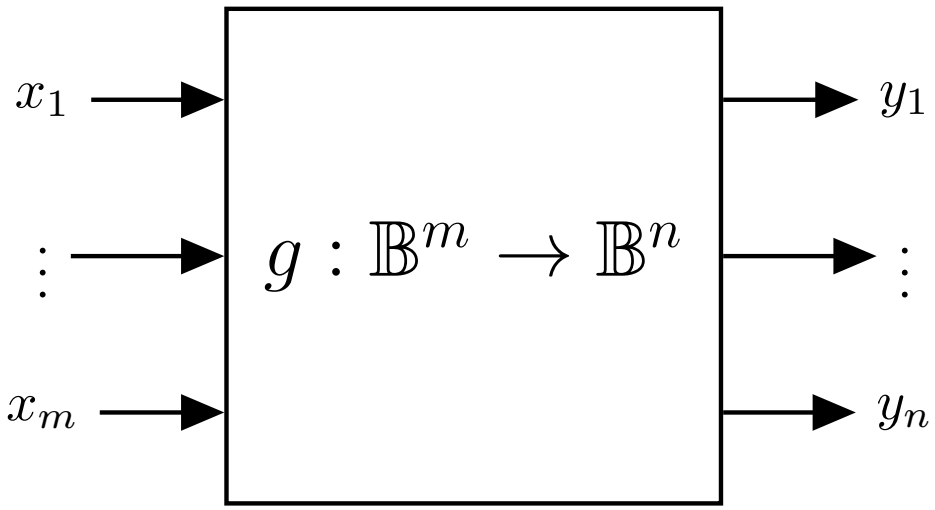
Rice University

lucasmt@rice.edu

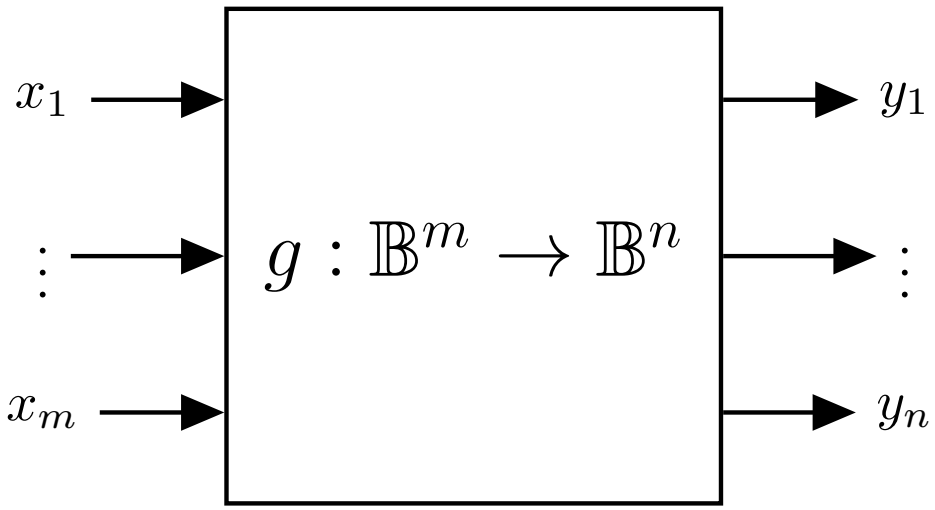
May 9, 2016

Joint work with Dror Fried and Moshe Vardi

Motivation



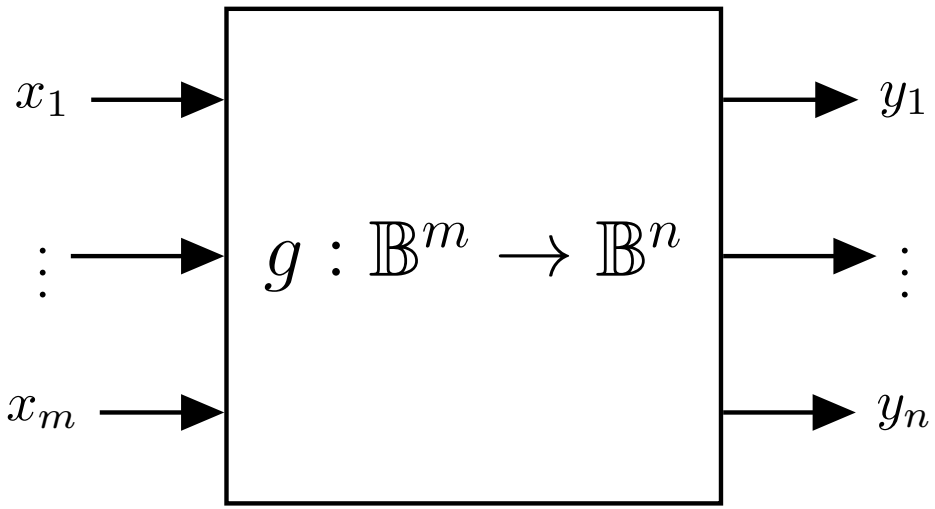
Motivation



$$f : \mathbb{B}^m \times \mathbb{B}^n$$

$$f(\vec{x}, \vec{y})$$

Motivation



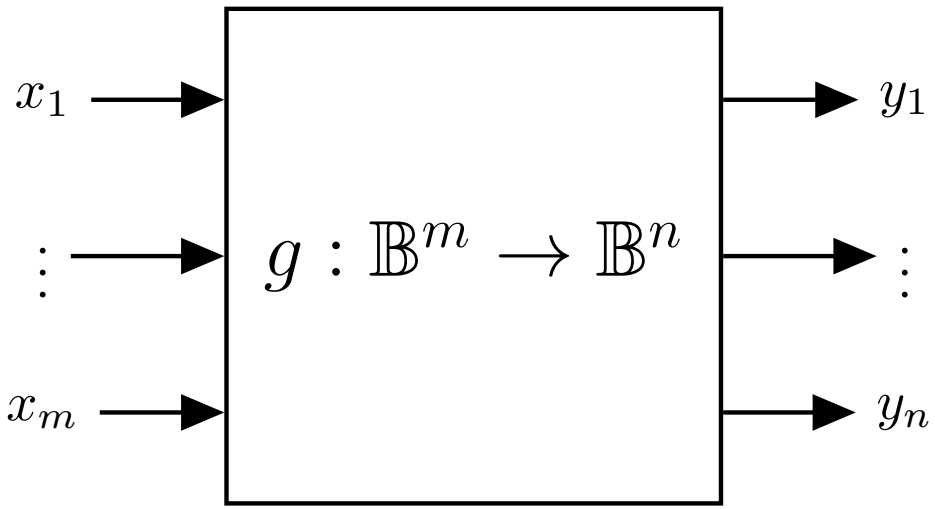
$$f : \mathbb{B}^m \times \mathbb{B}^n$$

$$f(\vec{x}, \vec{y})$$

$$g : \mathbb{B}^m \rightarrow \mathbb{B}^n$$

$$\vec{y} = g(\vec{x})$$

Motivation

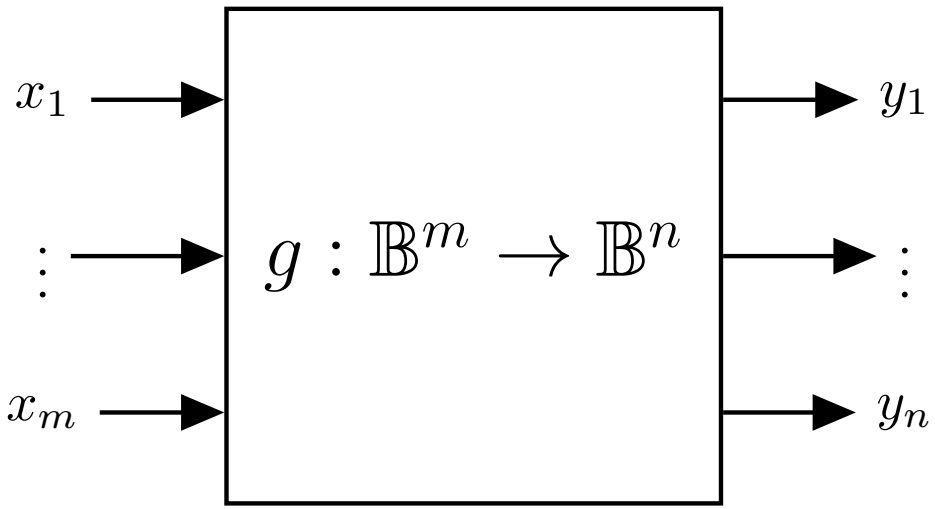


$$f : \mathbb{B}^m \times \mathbb{B}^n \qquad g : \mathbb{B}^m \rightarrow \mathbb{B}^n$$

$$f(\vec{x}, \vec{y}) \qquad \vec{y} = g(\vec{x})$$

Given $f(\vec{x}, \vec{y})$, how to obtain $g(\vec{x})$?

Motivation



$$f : \mathbb{B}^m \times \mathbb{B}^n \qquad g : \mathbb{B}^m \rightarrow \mathbb{B}^n$$

$$f(\vec{x}, \vec{y}) \qquad \vec{y} = g(\vec{x})$$

Given $f(\vec{x}, \vec{y})$, how to obtain $g(\vec{x})$? How to identify when an input $\vec{x}$ has no corresponding output?

Overview

1. Problem formulation
2. Framework
   - Introducing Self-Substitution
   - Synthesis in two phases
3. Experimental evaluation

Problem formulation

Given:

Specification: Boolean formula $f(\vec{x}, \vec{y})$

- Input variables: $x_1, \ldots, x_m$
- Output variables: $y_1, \ldots, y_n$

Problem formulation

Given:

Specification: Boolean formula $f(\vec{x}, \vec{y})$

- Input variables: $x_1, \ldots, x_m$
- Output variables: $y_1, \ldots, y_n$

Obtain:

Precondition: Boolean formula $p(\vec{x})$

- $p(\vec{x})$ is true exactly for those inputs for which there is a valid output
- Formally, $p(\vec{x}) \leftrightarrow \exists \vec{y}.f(\vec{x}, \vec{y})$

Problem formulation

Given:

Specification: Boolean formula $f(\vec{x}, \vec{y})$

- Input variables: $x_1, \ldots, x_m$
- Output variables: $y_1, \ldots, y_n$

Obtain:

Precondition: Boolean formula $p(\vec{x})$

- $p(\vec{x})$ is true exactly for those inputs for which there is a valid output
- Formally, $p(\vec{x}) \leftrightarrow \exists \vec{y}.f(\vec{x}, \vec{y})$

Implementation: Boolean function $g(\vec{x}) = (g_1(\vec{x}), \ldots, g_n(\vec{x}))$

- for every input that satisfies the precondition, g satisfies f
- Formally, $p(\vec{x}) \rightarrow f(\vec{x}, g(\vec{x}))$

Self-Substitution

Lemma (Self-Substitution)

Let $f(\vec{x}, y)$ be a Boolean formula. Then,

- $\exists y.f(\vec{x}, y) \equiv f(\vec{x}, f(\vec{x}, 1))$
- $\forall y.f(\vec{x}, y) \equiv f(\vec{x}, f(\vec{x}, 0))$

Self-Substitution

Lemma (Self-Substitution)

Let $f(\vec{x}, y)$ be a Boolean formula. Then,

- $\exists y.f(\vec{x}, y) \equiv f(\vec{x}, f(\vec{x}, 1))$
- $\forall y.f(\vec{x}, y) \equiv f(\vec{x}, f(\vec{x}, 0))$

Self-Substitution provides a novel way to perform Quantifier Elimination.

Self-Substitution for Synthesis

For a specification $f(\vec{x}, y)$ of a single output variable, Self-Substitution directly defines an implementation for y:

Self-Substitution for Synthesis

For a specification $f(\vec{x}, y)$ of a single output variable, Self-Substitution directly defines an implementation for y:

- $\exists y.f(\vec{x}, y) \equiv f(\vec{x}, \boldsymbol{f}(\vec{\boldsymbol{x}}, \mathbf{1}))$

Self-Substitution for Synthesis

For a specification $f(\vec{x}, y)$ of a single output variable, Self-Substitution directly defines an implementation for y:

- $\exists y.f(\vec{x}, y) \equiv f(\vec{x}, \boldsymbol{f}(\vec{\boldsymbol{x}}, \mathbf{1}))$
- $g(\vec{x}) = f(\vec{x}, 1)$

Self-Substitution for Synthesis

For a specification $f(\vec{x}, y)$ of a single output variable, Self-Substitution directly defines an implementation for y:

- $\exists y.f(\vec{x}, y) \equiv f(\vec{x}, \boldsymbol{f}(\vec{\boldsymbol{x}}, \mathbf{1}))$
- $g(\vec{x}) = f(\vec{x}, 1)$

$f(\vec{x}, 1)$ is a *default 1* implementation of y.

General framework

Given a specification $f(\vec{x}, \vec{y})$, we perform synthesis in two phases:

Phase 1: Quantifier elimination

Phase 2: Function construction

Phase 1: Quantifier Elimination

Using Self-Substitution, eliminate each y_i in sequence.

At each step of this process we obtain:

$$f_i(\vec{x}, y_1, \ldots, y_i) \equiv \exists y_{i+1} \ldots y_n . f(\vec{x}, y_1, \ldots, y_n)$$

Phase 1: Quantifier Elimination

Using Self-Substitution, eliminate each y_i in sequence.

At each step of this process we obtain:

$$f_i(\vec{x}, y_1, \ldots, y_i) \equiv \exists y_{i+1} \ldots y_n . f(\vec{x}, y_1, \ldots, y_n)$$

In the final step, we have:

$$f_0(\vec{x}) \equiv \exists y_1 \ldots y_n . f(\vec{x}, y_1, \ldots, y_n)$$

Phase 1: Quantifier Elimination

Using Self-Substitution, eliminate each y_i in sequence.

At each step of this process we obtain:

$$f_i(\vec{x}, y_1, \ldots, y_i) \equiv \exists y_{i+1} \ldots y_n . f(\vec{x}, y_1, \ldots, y_n)$$

In the final step, we have:

$$f_0(\vec{x}) \equiv \exists y_1 \ldots y_n . f(\vec{x}, y_1, \ldots, y_n)$$

$f_0(\vec{x})$ is exactly the precondition $p(\vec{x})$.

Phase 2: Function construction

For each output y_i, use $f_i(\vec{x}, y_1, \ldots, y_i)$ to construct $g_i(\vec{x})$.

Phase 2: Function construction

For each output y_i, use $f_i(\vec{x}, y_1, \ldots, y_i)$ to construct $g_i(\vec{x})$.

Base case: $f_1(\vec{x}, y_1)$

Phase 2: Function construction

For each output y_i, use $f_i(\vec{x}, y_1, \ldots, y_i)$ to construct $g_i(\vec{x})$.

Base case: $f_1(\vec{x}, y_1)$
$g_1(\vec{x}) = f_1(\vec{x}, 1)$

Phase 2: Function construction

For each output y_i, use $f_i(\vec{x}, y_1, \ldots, y_i)$ to construct $g_i(\vec{x})$.

Base case: $f_1(\vec{x}, y_1)$
$g_1(\vec{x}) = f_1(\vec{x}, 1)$

Induction step: $f_i(\vec{x}, y_1, \ldots, y_{i-1}, y_i)$

Phase 2: Function construction

For each output y_i, use $f_i(\vec{x}, y_1, \ldots, y_i)$ to construct $g_i(\vec{x})$.

Base case: $f_1(\vec{x}, y_1)$
$g_1(\vec{x}) = f_1(\vec{x}, 1)$

Induction step: $f_i(\vec{x}, y_1, \ldots, y_{i-1}, y_i)$
$g_i(\vec{x}) = f_i(\vec{x}, g_1(\vec{x}), \ldots, g_{i-1}(\vec{x}), 1)$

Phase 2: Function construction

For each output y_i, use $f_i(\vec{x}, y_1, \ldots, y_i)$ to construct $g_i(\vec{x})$.

Base case: $f_1(\vec{x}, y_1)$
$g_1(\vec{x}) = f_1(\vec{x}, 1)$

Induction step: $f_i(\vec{x}, y_1, \ldots, y_{i-1}, y_i)$
$g_i(\vec{x}) = f_i(\vec{x}, g_1(\vec{x}), \ldots, g_{i-1}(\vec{x}), 1)$

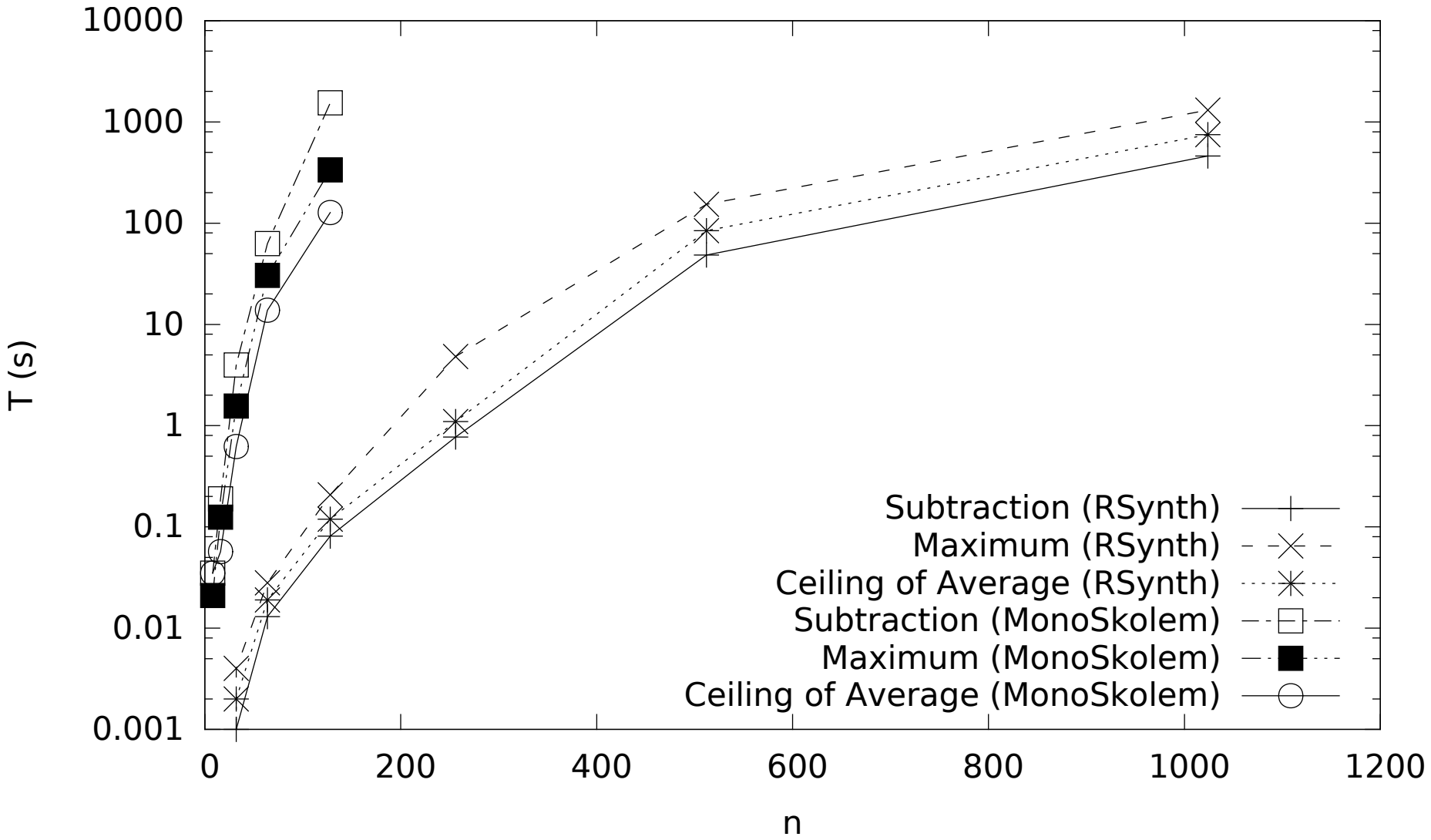
$g(\vec{x}) = (g_1(\vec{x}), \ldots, g_n(\vec{x}))$ is a correct implementation of $f(\vec{x}, \vec{y})$.

Experiments

- RSynth: implementation of the framework using Binary Decision Diagrams
- Comparison with MonoSkolem tool based on (Jiang et al., 2009)
- Evaluation on *scalable* benchmarks

	Function	Specification
Subtraction	$\vec{y} = \vec{x'} - \vec{x}$	$\vec{y} + \vec{x} = \vec{x'}$
Maximum	$\vec{y} = max(\vec{x}, \vec{x'})$	$(\vec{y} \geq \vec{x}) \wedge (\vec{y} \geq \vec{x'}) \wedge ((\vec{y} = \vec{x}) \vee (\vec{y} = \vec{x'}))$
Ceiling of Average	$\vec{y} = \lceil \frac{\vec{x} + \vec{x'}}{2} \rceil$	$(2\vec{y} = \vec{x} + \vec{x'}) \vee (2\vec{y} + 1 = \vec{x} + \vec{x'})$

Results


10000
1000
100
10
1
0.1
0.01
0.001
T (s)
0
200
400
600
800
1000
1200
n
Subtraction (RSynth)
Maximum (RSynth)
Ceiling of Average (RSynth)
Subtraction (MonoSkolem)
Maximum (MonoSkolem)
Ceiling of Average (MonoSkolem)

Conclusions

- When problem has an efficient variable ordering, performance scales well even for a large number of variables.
- RSynth outperforms previous approaches, even using a naive strategy for selecting implementations.

Future work

- Factored representation as a way to control formula size.
- More elaborate strategies for selecting the implementation for each variable.
- Alternative representations

Thank you! Questions?