CEGIS(T)
CounterExample Guided Inductive Synthesis
modulo Theories
CAV 2018

Alessandro Abate\textsuperscript{1}, Cristina David\textsuperscript{2}, Pascal Kesseli\textsuperscript{3}, Daniel Kroening\textsuperscript{13},
Elizabeth Polgreen\textsuperscript{0}

\textsuperscript{0}University of Edinburgh, \textsuperscript{1}University of Oxford, \textsuperscript{2}University of Bristol, \textsuperscript{3}DiffBlue Ltd
CounterExample Guided Inductive Synthesis modulo Theories

CEGIS(T)

Alessandro Abate¹, Cristina David², Pascal Kesseli³, Daniel Kroening⁴, Elizabeth Polgreen⁰

¹University of Edinburgh, ²University of Oxford, ³University of Bristol, ⁴DiffBlue Ltd

CAV 2018
CEGIS(T)

Program synthesis is hard
CEGIS(T)

Program synthesis is hard

CEGIS framework that uses a theory solver to
• verify generalized candidate solutions
• return more general counterexamples

CEGIS(T) is able to synthesize programs containing arbitrary constants that elude other solvers.
Outline

• Overview of CEGIS and motivation for CEGIS(T)

• CEGIS(T): algorithm in detail

• Evaluation

• CEGIS(T) in CVC4

• Ongoing work: beyond constants
∃P ∀x . σ(P, x)
\[ \exists P \cdot \forall x_i \cdot \sigma(P, x) \]

CEGIS
CEGIS

![Diagram](attachment:image.png)
$\exists x . \neg \sigma(P^*, x)$
CEGIS
∃P . ∀x_i . σ(P, x)
Is it a plant?

No
Is it a plant?

Yes

No

Does it have legs?

Yes

No
Can I eat it?

Does it have legs?

Is it a plant?

SYNTHESIZE

No

Yes

eerrm..

VERIFY
No
Safety invariant

```c
int x = 5;
while (x < 1000)
    x++;
assert(5 < x && x < 1005)
```

\[ init(x) \iff x = 0 \]
\[ trans(x, x') \iff x' = x + 1 \]

**find \( inv(x) \) such that:**

\[ init(x) \implies inv(x) \]
\[ inv(x) \land (x < 1000) \land trans(x, x') \implies inv(x') \]
\[ inv(x) \land \neg(x < 1000) \implies (x < 1005) \land (x > 5) \]
Safety invariant

```c
int x = 5;
while ( x < 1000 )
    x++;
assert( 5 < x && x < 1005 )
```

\[
\text{init}(x) \iff x = 0 \quad \text{trans}(x, x') \iff x' = x + 1
\]

\[
\text{inv}(x) = (4 < x) \land (x < 1003)
\]
Possible solution:

\[ inv(x) = (4 < x) \land (x < 1003) \]

And so on ..
Can we ask more general questions?
Is it a plant? Yes

Does it have legs? Yes

Can I eat it? errm..
No, it’s not a plant
No, it’s not a plant

No, it has < 4 legs
No, it’s not a plant
No, it has < 8 legs
No, I can’t sit on it
Can we give more general answers?
More general questions

More general answers

CEGIS(T)
Outline

• Overview of CEGIS and motivation for CEGIS(T)

• **CEGIS(T): algorithm in detail**

• Evaluation

• CEGIS(T) in CVC4

• Ongoing work: beyond constants
DPLL(T)

Diagram showing:
- PROPOSITIONAL SAT SOLVER
- ADD CLAUSES
- DEDUCTION

Connections indicate the flow of processes with
- A red 'X' and green tick indicating some processes are not applicable or correct

DPLL Theory Solver
CEGIS(T)

SYNTHESIZE

VERIFY

DEDUCTION

Generalize candidate

first order solver
CEGIS(T)

CEGIS

SYNTHESIZE

VERIFY

$P^*$

Generalize candidate

DEDUCTION

first order solver
Generalize

Candidate

\[ P^* \]

\( x < 95 \)

Generalized candidate

\[ P^*[v] \]

\( x < v \)
CEGIS(T)

SYNTHESIZE

VERIFY

DEDUCTION

$P^*[v]$

Generalize candidate

first order solver
Deduction

$$\exists \nu \forall x. \sigma(P^*[\nu], x)$$

is there a value for $\nu$ that makes $(x < \nu)$ a valid invariant
CEGIS(T)

SYNTHESIZE

VERIFY

\neg P^*[v]

DEDUCTION

Generalize candidate

first order solver
First order solver

Solves 1st order formula with:
• Arbitrary propositional structure
• 1 quantifier alternation

Paper presents 2 versions:
• SMT
• FM
\[ \exists v \forall x. \sigma(P^*[v], x) \]
CEGIS(T) - SMT

\[ \exists v \forall x. \sigma(P^*[v], x) \land (v < c) \]

\[ \exists v \forall x. \sigma(P^*[v], x) \land (v > c) \]

\[ \neg P^*[v] \]

\[ v > c \]

\[ v < c \]

\[ P^*[v] \]

BLOCK

CONSTRAINT

CONSTRAINT

SOLUTION
Target:
\[ \text{inv}(x) = (4 < x) \land (x < 1003) \]

\[ \exists v \forall x. \sigma(P^*[v], x) \land (v < c) \]

\[ \exists v \forall x. \sigma(P^*[v], x) \land (v > c) \]

\[ P^* = (x < 95) \]

\[ P^*[v] = (x < v) \]

\[ \neg P^*[v] \] BLOCK

\[ v > c \] CONSTRAINT

\[ v < c \] CONSTRAINT

\[ P^*[v] \] SOLUTION
Target:
\[ \text{inv}(x) = (4 < x) \land (x < 1003) \]

\[ \exists \nu \forall x. \sigma(P^*[\nu], x) \land (\nu < 95) \]

\[ \exists \nu \forall x. \sigma(P^*[\nu], x) \land (\nu > 95) \]

\[ P^* = (x < 95) \]

\[ P^*[\nu] = (x < \nu) \]

\[ \neg P^*[\nu] \quad \text{BLOCK} \]

\[ \nu > 95 \quad \text{CONSTRAINT} \]

\[ \nu < 95 \quad \text{CONSTRAINT} \]

\[ P^*[\nu] \quad \text{SOLUTION} \]
Target:
\( \text{inv}(x) = (4 < x) \land (x < 1003) \)
Target:
\[ \text{inv}(x) = (4 < x) \land (x < 1003) \]
Target:
\[ \text{inv}(x) = (4 < x) \land (x < 1003) \]
Target:
\[ \text{inv}(x) = (4 < x) \land (x < 1003) \]

\[ \exists v \forall x. \sigma(P^*[v], x) \land (v_1 < 95) \]

\[ \exists v \forall x. \sigma(P^*[v], x) \land (v_1 > 95) \]

\[ \neg P^*[v] \]

\[ v > 95 \]

\[ v < 95 \]

\[ P^*[v] \]

SOLUTION

BLOCK

CONSTRAINT

CONSTRAINT
Target:
\[ \text{inv}(x) = (4 < x) \land (x < 1003) \]

\[ \exists \forall x. \sigma(P^*[v], x) \land (v_1 < 95) \]

\[ \exists \forall x. \sigma(P^*[v], x) \land (v_1 > 95) \]

\[ \neg P^*[v] \]

\[ v > 95 \]

\[ v < 95 \]

\[ P^*[v] \]

BLOCK

CONSTRAINT

CONSTRAINT

SOLUTION
Outline

• Overview of CEGIS and motivation for CEGIS(T)

• CEGIS(T): algorithm in detail

• Evaluation

• CEGIS(T) in CVC4

• Ongoing work: beyond constants
Experiments

Benchmarks:
• Bitvectors
• Syntax-guided Synthesis competition \(\text{(without the syntax)}\)
• Loop invariants
• Danger invariants

Solvers:
• CVC4
• EUSolver, E3Solver, LoopInvGen – bitvectors with no grammar unsupported
Experiments

SOLVED
CVC4 - 29
CEGIS(T) - 49

TIME-OUT

<table>
<thead>
<tr>
<th>TIME (s)</th>
<th>CVC4</th>
<th>CEGIS(T)</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;1s</td>
<td>38</td>
<td></td>
</tr>
<tr>
<td>[1,10]</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>[10,100]</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>[100,3600]</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>T/O</td>
<td>24</td>
<td>10</td>
</tr>
</tbody>
</table>

Supported by ERC project 280053 (CPROVER) and the H2020 FET OPEN 712689 SC².
Cristina David is supported by the Royal Society University Research Fellowship UF160079.

© The Author(s) 2018
https://doi.org/10.1007/978-3-319-96145-3_15

Author Proof
Experiments - update

SOLVED
CVC4 - 59
CEGIS(T) - 44

CVC4 v1.7
CEGIS(T) updated
more benchmarks
Outline

• Overview of CEGIS and motivation for CEGIS(T)

• CEGIS(T): algorithm in detail

• Evaluation

• **CEGIS(T) in CVC4**

• Ongoing work: beyond constants
CEGIS(T) in CVC4

CVC4 implementation of CEGIS(T)
• CVC4 version 1.7
• Makes self-call to CVC4 SMT solver
• Supports CEGIS(T) with a syntactic template
CEGIS(T) in CVC4

SYNTHESIZE

[\(P^*[\nu]\)]

VERIFY

DEDUCTION

CEGIS

first order solver
CEGIS(T) in CVC4

- BV-invertibility benchmarks
- CEGIS(T) solves 8 unique benchmarks
- avg. ~20s faster on mutually solved benchmarks
- VBS solver gains avg. 48s on mutually solved benchmarks
CEGIS(T) solves program synthesis via 1\textsuperscript{st} order solvers that support quantifiers:
- Enables use of existing solvers

Algorithmic insights:
- verify generalized candidate solutions
- return generalized counterexamples
CEGIS(T) solves program synthesis via 1st order solvers that support quantifiers:

- Enables use of existing solvers

**Algorithmic insights:**

- verify generalized candidate solutions
- return generalized counterexamples

CEGIS(T) SYNTHESIZE VERIFY
CEGIS(T) solves program synthesis via 1st order solvers that support quantifiers:

- Enables use of existing solvers

Algorithmic insights:

- Verify generalized candidate solutions
- Return generalized counterexamples
Outline

• Overview of CEGIS and motivation for CEGIS(T)

• CEGIS(T): algorithm in detail

• Evaluation

• CEGIS(T) in CVC4

• Ongoing work: beyond constants
Beyond constants?

SYNTHESIZE

VERIFY
Beyond constants?
Via Oracle Guided Synthesis

A Theory of Formal Synthesis via Inductive Learning

Susmit Bha
United Technologies Research Center, Berkeley
bhash@utrc.utc.com

Sanjiv A. Seshia
CSICS, UC Berkeley
seshia@cs.berkeley.edu

Formal synthesis is the process of generating a program satisfying a high-level formal specification. In recent years, effective formal synthesis methods have been proposed based on the use of inductive learning. We refer to this class of methods that learn programs from examples as formal inductive synthesis. In this paper, we present a theoretical framework for formal inductive synthesis. We show that new formal inductive synthesis differs from traditional machine learning. We also create a family of synthesizers that operate by iteratively querying an oracle. An instance of OGIS that has had such practical impact is counterexample-guided inductive synthesis (CEGIS). We present a theoretical characterization of CEGIS for learning any program that computes a recursive language. In particular, we analyze the relative power of CEGIS variants, where the types of counterexamples generated by the learned algorithm. We also consider the impact of bounded versus unbounded memory available to the learning algorithm. In the special case where the universe of candidate programs is finite, we reduce the speed of convergence of the notion of teaching dimension studied in machine learning theory. The results of the paper take a first step towards a theoretical foundation for the emerging field of formal inductive synthesis.

1 Introduction
Beyond constants: general queries and responses

SYNTHESIZE

QUERY

ORACLE

RESPONSE

Is my program correct?

No, but I can tell you that on input 7 it should return 73.
CEGIS(T) as oracle guided synthesis

Is my program correct?

no, but if I replace 7 with 9 it is.

no, and all programs of this structure are wrong

no, and if you try another program like this, make the constant $< 12$.

yes
Beyond constants: general queries and responses

- Future direction for SyGuS
- Syntax extension in SyGuS-IF
- What type of queries/responses? Any! (Provided the response can be expressed as a logical constraint)
Conclusions

CEGIS(T) solves program synthesis via 1st order solvers that support quantifiers

Algorithmic insights:
• verify generalized candidate solutions
• return generalized counterexamples

Broader communication is good!
Conclusions

CEGIS(T) solves program synthesis via 1st order solvers that support quantifiers

Algorithmic insights:
• verify generalized candidate solutions
• return generalized counterexamples

Broader communication is good!

What is it?

It’s an owl!!