Domains for Higher-Order Games

Matthew Hague, Roland Meyer, Sebastian Muskalla
Royal Holloway University of London, and TU Braunschweig

MFCS 2017
Decide the winning region / strategy of inclusion games

- Played over higher-order recursion schemes.
  - (Higher-order control-flow)
- A play generates a program trace.
- The program trace must belong to a regular specification.
Overview – Our Solution

We use

- Concrete semantics of terms $t$
  - Pointed $\omega$-complete partial order (CPPO)
  - Fixed point semantics via Kleene iteration
  - Infinite formula evaluates to “true” iff Player $\circ$ can win from $t$

- Abstract-interpretation framework
  - Into a finite CPPO – fixed point computable
  - Fixed-point transfer ensures exact abstraction
  - Gives a decision procedure for determining winner
Background
Verification Problem

The verification problem:

Given: Source code of a program $P$ and a specification $\varphi$

Question: Does runtime behaviour of $P$ satisfy $\varphi$?
The verification problem:

Given: Source code of a program $P$ and a specification $\varphi$

Question: Does runtime behaviour of $P$ satisfy $\varphi$?

Language-theoretic approach:

$L_P =$ possible program executions
$L_\varphi =$ valid executions

Decide: $L_P \subseteq L_\varphi$
The Good and Bad

\[ \mathcal{L}_P = \text{possible program executions} \]

\[ \mathcal{L}_\varphi = \text{valid executions} \]
The Good and Bad

\[ \mathcal{L}_P = \text{possible program executions} \]
\[ \mathcal{L}_\varphi = \text{valid executions} \]

Good: \( \mathcal{L}_\varphi \) usually regular (easy)

Bad: \( \mathcal{L}_P \) usually complicated...
The Good and Bad

\[ \mathcal{L}_P = \text{possible program executions} \]
\[ \mathcal{L}_\varphi = \text{valid executions} \]

Good: \( \mathcal{L}_\varphi \) usually regular (easy)

Bad: \( \mathcal{L}_P \) usually complicated...

Because of the bad:

- Problem is undecidable
- We need to approximate \( \mathcal{L}_P \)
Breaking down $\mathcal{L}_P$

A program has control-flow and data

$$\mathcal{L}_P = \mathcal{L}_{CF} \cap \mathcal{L}_{Data}$$

We know

- $\mathcal{L}_{CF}$ may have many manageable representations
  - Regular, context-free, higher-order...
- $\mathcal{L}_{Data}$ can be arbitrary
  - Best handled using techniques from logic
Breaking down $\mathcal{L}_P$

A program has control-flow and data

$$\mathcal{L}_P = \mathcal{L}_{CF} \cap \mathcal{L}_{Data}$$

We know

- $\mathcal{L}_{CF}$ may have many manageable representations
  - Regular, context-free, higher-order...
- $\mathcal{L}_{Data}$ can be arbitrary
  - Best handled using techniques from logic

How to combine the two?

- CEGAR loop [Podelski et al. since 2010]
CEGAR Loop

\[ \text{Init } \mathcal{L}_S := \mathcal{L}_\varphi \]
CEGAR Loop

\[
\text{Init } \mathcal{L}_S := \mathcal{L}_\varphi \\
\mathcal{L}_{CF} \subseteq \mathcal{L}_S ?
\]
CEGAR Loop

Init $\mathcal{L}_S := \mathcal{L}_\varphi$

$\mathcal{L}_{CF} \subseteq \mathcal{L}_S$ ? yes return $P \models \varphi$
CEGAR Loop

Init $\mathcal{L}_S := \mathcal{L}_\varphi$

$\mathcal{L}_{CF} \subseteq \mathcal{L}_S$ ?  yes  return $P \models \varphi$

no $w \in \mathcal{L}_{CF} \setminus \mathcal{L}_S$

$w \in \mathcal{L}_P$ ?
CEGAR Loop

\[
\text{Init } \mathcal{L}_S := \mathcal{L}_\varphi
\]

\[
\mathcal{L}_{CF} \subseteq \mathcal{L}_S \quad \text{?} \quad \text{yes} \quad \rightarrow \quad \text{return } P \models \varphi
\]

\[
\text{no } w \in \mathcal{L}_{CF} \setminus \mathcal{L}_S
\]

\[
w \in \mathcal{L}_P \quad ?
\]

\[
\text{yes} \quad \rightarrow \quad \text{return } P \not\models S
\]
CEGAR Loop

Init $\mathcal{L}_S := \mathcal{L}_\varphi$

$\mathcal{L}_{CF} \subseteq \mathcal{L}_S$ ? yes return $P \models \varphi$

no $w \in \mathcal{L}_{CF} \setminus \mathcal{L}_S$

$w \not\Rightarrow \mathcal{L}_w$, $\mathcal{L}_w \cap \mathcal{L}_P = \emptyset$ no $w \in \mathcal{L}_P$ ?

yes return $P \not\models S$

no
CEGAR Loop

Init $\mathcal{L}_S := \mathcal{L}_\varphi$

$\mathcal{L}_S := \mathcal{L}_S \cup \mathcal{L}_w \quad \rightarrow \quad \mathcal{L}_{CF} \subseteq \mathcal{L}_S \quad ? \quad \rightarrow \quad \text{return } P \models \varphi$

$w \leadsto \mathcal{L}_w$, $\mathcal{L}_w \cap \mathcal{L}_P = \emptyset$

no $w \in \mathcal{L}_{CF} \setminus \mathcal{L}_S$

no $w \in \mathcal{L}_P \quad ?$

yes

return $P \not\models S$
CEGAR Loop

Init $\mathcal{L}_S := \mathcal{L}_\varphi$

$\mathcal{L}_S := \mathcal{L}_S \cup \mathcal{L}_w \rightarrow \mathcal{L}_{CF} \subseteq \mathcal{L}_S ?$

yes $\rightarrow$ return $P \models \varphi$

no $\rightarrow$ $w \in \mathcal{L}_{CF} \setminus \mathcal{L}_S$

$w \leadsto \mathcal{L}_w$, $\mathcal{L}_w \cap \mathcal{L}_P = \emptyset$

$w \in \mathcal{L}_P ?$

no $\rightarrow$ return $P \not\models S$

yes $\rightarrow$ $w \in \mathcal{L}_P$

Algorithmic challenges:
- Inclusion $\mathcal{L}_{CF} \subseteq \mathcal{L}_S$
- Recursion schemes!
- Membership $w \in \mathcal{L}_P$
- Hoare Logic
- Extrapolation $w \leadsto \mathcal{L}_w$
CEGAR Illustration

\[ \mathcal{L}_\varphi \]
\[ \mathcal{L}_{CF} \]
\[ \mathcal{L}_P \]
$L_{\varphi}$

$L_{CF}$

$w_1$

$L_P$
CEGAR Illustration
CEGAR Illustration
CEGAR Illustration
Language Synthetic Synthesis
Why write a bad program and check it’s ok?
  o Better to generate a correct program!

The synthesis problem:
  Given: Template of a program $P$ and a specification $\varphi$
  Question: Is there an instantiation $P'$ that satisfies $\varphi$?
Why write a bad program and check it’s ok?
  o Better to generate a correct program!

The synthesis problem:
  Given: Template of a program $P$ and a specification $\varphi$
  Question: Is there an instantiation $P'$ that satisfies $\varphi$?

Approach:
  o Language-theoretic synthesis
  o CEGAR loop
Types of Non-determinism

Model the control-flow as a Higher-order Recursion Scheme

**Demonic**

Program input:
- handle all possibilities.

```python
def F():
    x = read()
    if x == 0:
        G()
    else:
        H()
```

becomes

\[
F = \text{rd}(x, 0) \ G \land \text{rd}(x, 1) \ H
\]

**Angelic**

Program branch:
- choose best.

```python
def F():
    if ???:
        G()
    else:
        H()
```

becomes

\[
F = G \lor H
\]
Language-Theoretic Synthesis

Model as a higher-order two player perfect information game

- Player □ – uncontrollable non-determinism
- Player ◦ – controllable non-determinism

Is there a strategy $s$ for ◦ such that

$$L_{G@s} \subseteq L_{\varphi}$$

I.e. when Player ◦ uses $s$ all generated words are in $L_{\varphi}$
Model as a higher-order two player perfect information game

○ Player □ – uncontrollable non-determinism
○ Player ◦ – controllable non-determinism

Is there a strategy \( s \) for ◦ such that

\[
\mathcal{L}_{G@s} \subseteq \mathcal{L}_\varphi
\]

I.e. when Player ◦ uses \( s \) all generated words are in \( \mathcal{L}_\varphi \)

Replace the inclusion check

\[
\mathcal{L}_G \subseteq \mathcal{L}_S
\]

with strategy synthesis.
CEGAR Loop

Init $\mathcal{L}_S := \mathcal{L}_\varphi$

$\mathcal{L}_S := \mathcal{L}_\varphi \cup \mathcal{L}_w \rightarrow \exists s. \mathcal{L}_{CF@s} \subseteq \mathcal{L}_S \ ? \stackrel{yes}{\rightarrow} \text{return } P@s \models \varphi$

$w \rightsquigarrow \mathcal{L}_w$, $\mathcal{L}_w \cap \mathcal{L}_P = \emptyset$

no $\exists s_{op}. w \in \mathcal{L}_{CF@s_{op}} \setminus \mathcal{L}_S$

no $w \in \mathcal{L}_P \ ?$

yes return \text{Algorithmic challenges:}\n
Game $\exists s. \mathcal{L}_{CF@s} \subseteq \mathcal{L}_S$

Membership $w \in \mathcal{L}_P$

Extrapolation $w \rightsquigarrow \mathcal{L}_w$

yes return $\forall s. P@s \not\models S$
Higher-Order Inclusion Games
Given a

- Higher-Order Recursion Scheme
- Ownership partition of non-terminals
  \[ S \circ = F \circ G \square \]
  \[ F \circ f = a (F \circ f) \lor a (f b) \]
  \[ G \square x = x \land b x \]
- Finite automaton \( A \) over terminals \( \{a, b\} \)
We study safety games:

Can Player $\circ$ avoid generating a word $w \notin \mathcal{L}_A$?
Example Play

\[
S = F \circ G
\]

\[
F \circ f = a (F \circ f) \lor a (f b)
\]

\[
G x = x \land b x
\]
\[ S \circ = F \circ G \]
\[ F \circ f = a (F \circ f) \lor a (f b) \]
\[ G \Box x = x \land b x \]
Example Play

\[ S_\circ = F_\circ \ G_\square \]

\[ F_\circ f = a (F_\circ f) \lor a (f b) \]

\[ G_\square x = x \land b x \]
Example Play

\[ S \odot = F \odot G \]

\[ F \odot f = a (F \odot f) \lor a (f \ b) \]

\[ G \square x = x \land b \ x \]

[Diagram]

\[ F \odot \]

\[ G \square \]
Example Play

\[ S \circ = F \circ G \square \]

\[ F \circ f = a (F \circ f) \lor a (f \ b) \]

\[ G \square x = x \land b \ x \]

\[ a \]

\[ F \circ \]

\[ G \square \]
Example Play

\[ S_\circ = F_\circ G_\square \]

\[ F_\circ f = a \ (F_\circ f) \lor a \ (f \ b) \]

\[ G_\square x = x \land b \ x \]
Example Play

\[ S_\circ = F_\circ G \Box \]

\[ F_\circ f = a (F_\circ f) \lor a (f b) \]

\[ G \Box x = x \land b x \]

\[ \begin{array}{c}
    a \\
    a \\
    G \Box \\
    b
\end{array} \]
$S \circ = F \circ G$

$F \circ f = a \ (F \circ f) \lor a \ (f \ b)$

$G \ x = x \land b \ x$

\[ \begin{align*}
\text{Diagram:} & \quad a \\
& \quad a \\
& \quad b \\
& \quad b \\
& \quad a \\
& \quad G \\
\end{align*} \]
Example Play

\[ S_\circ = F_\circ G_\square \]
\[ F_\circ f = a (F_\circ f) \lor a (f b) \]
\[ G_\square x = x \land b x \]

\[ \begin{array}{c}
  a \\
  a \\
  b \\
  b \\
\end{array} \]
Example Play

\[ S_\circ = F_\circ G_\Box \]
\[ F_\circ f = a (F_\circ f) \lor a (f b) \]
\[ G_\Box x = x \land b x \]
Example Play

\[
S_\circ = F_\circ G_{\square}
\]
\[
F_\circ f = a (F_\circ f) \vee a (f b)
\]
\[
G_{\square} x = x \land b x
\]

Since \( aabb \notin \Sigma^*bb\Sigma^* \) Player \( \circ \) loses this play.
Results

**Theorem**
Given a higher-order game \( G \) and regular specification \( \mathcal{A} \), determining the winning of \( G \) wrt \( \mathcal{A} \) is \((k+1)\)-EXPTIME-complete for an order-\( k \) scheme.

Such a result is already known

- Determinize \( \mathcal{A} \)
- Product with \( G \)
- \( \Rightarrow \) standard safety game over higher-order recursion schemes.
  - Solvable by e.g. [Serre]
Our Approach

We provide a new approach

- Develop a concrete semantics $[S]$ of $G$ wrt $\mathcal{A}$
  - infinite CPPO: monotone boolean formulas and continuous (higher-order) functions between them.
- Give a framework for exact abstract interpretation
- Abstract into an abstract semantics over a finite CPPO
- Compute the abstract semantics by simple Kleene iteration
Similar approaches have been studied in the literature.

- Models/domains:
  - Walukiewicz & Salvati
  - Melliès & Grellois
  - Hofmann, Chen & Ledent

- Abstract interpretation:
  - Abramsky & Hankin
  - Ramsay
  - Hofmann, Chen & Ledent
Solution Sketch: Concrete Semantics

Boolean formula representing game

\[ S = a \lor b \]

\[ \llbracket S \rrbracket = a \lor b \]

A proposition \( w \) is true iff \( w \notin \mathcal{L}_A \).
Solution Sketch: Concrete Semantics

Boolean formula representing game

\[ S = a \vee b \]
\[ \llbracket S \rrbracket = a \vee b \]

A proposition \( w \) is true iff \( w \notin \mathcal{L}_A \).

Formulas may be infinite:

\[ \llbracket S \rrbracket = (w_1 \vee w_2) \land (w_3 \vee w_4 \land (w_4 \land \cdots) \]
Solution Sketch: Concrete Semantics

Boolean formula representing game

\[
S = a \lor b \\
\llbracket S \rrbracket = a \lor b
\]

A proposition \( w \) is true iff \( w \notin \mathcal{L}_\mathcal{A} \). Formulas may be infinite:

\[
\llbracket S \rrbracket = (w_1 \lor w_2) \land (w_3 \lor w_4 \lor (w_4 \land \cdots)
\]

The semantics of a function is given as a function

\[
F : \tau_1 \to \tau_2 \\
\llbracket F \rrbracket \in D_{\tau_1} \to D_{\tau_2}
\]
Solution Sketch: Fixed Points

We compute the semantics via recursive equations

\[ F = \lambda x. a\ (F\ x) \]

\[ [F] = [\lambda x. a\ (F\ x)] \]
\[ = \lambda x. [a] [F]\ x \]

The semantics \([F]\) is

- A function
- A fixed point of the above recursive equations

Once we know \([F]\), \([G]\), ..., computing the semantics of a term is easy

\[ [F\ a] = [F][a] \]
Solution Sketch: Concrete Semantics

**Theorem**

The following are equivalent

- Player ◦ wins from \( t : o \)
- \([t]\) is true under \( \mathcal{L}_A \)

“True under \( \mathcal{L}_A \)”

- A proposition \( w \) is true iff \( w \notin \mathcal{L}_A \).
Solution Sketch: Abstraction

We can’t compute infinite formulas.

- We need semantics in a finite domain
  - Semantics is computable via simple Kleene iteration
Solution Sketch: Abstraction

We can’t compute infinite formulas.
- We need semantics in a finite domain
  - Semantics is computable via simple Kleene iteration

The number of propositions \( w \in \Sigma^* \) is infinite
- We abstract \( \alpha(w) \) into a finite domain
- Therefore only finitely many boolean formulas
  - Fixed point computation terminates
Solution Sketch: Abstraction

We can’t compute infinite formulas.
- We need semantics in a finite domain
  - Semantics is computable via simple Kleene iteration

The number of propositions $w \in \Sigma^*$ is infinite
- We abstract $\alpha(w)$ into a finite domain
- Therefore only finitely many boolean formulas
  - Fixed point computation terminates
- We show the abstraction is precise
  - This involves defining what precise means
  - Our abstraction is exact not approximate
  - No false positives!
Solution Sketch: Abstraction

We abstract $w$ by the set of states of $A$ from which $w$ is accepted

$$\alpha(w) = \{q \mid q \xrightarrow{w} q_f\}$$

Here

$$\alpha(aba) = \{q_0, q_1\}$$
Solution Sketch: Correctness

Truth of propositions:

- $w$ true iff $w \notin \mathcal{L}_A$
- $\alpha(w)$ true iff $q_0 \notin \alpha(w)$

The abstraction is precise:

**Theorem**

$$\alpha(\text{Concrete semantics}) = \text{Abstract semantics}$$

We can compute in the finite domain!
Solution Sketch: Finishing

Complexity:
- The complexity is \((k+1)\)-EXPTIME-complete for an order-\(k\) scheme
- We need a second abstraction into an optimised domain

Winning region and strategy
- For any term \([\tau]\) is computable in “linear time”.
- Winning strategy for Player \(\circ\)
  - Always choose moves that stay in the winning region
Conclusion

We have

- Defined and motivated higher-order inclusion games
- Shown \((k + 1)\)-EXPTIME-completeness
- Given a solution based on semantics in CPPOs
- Used exact abstract interpretation to obtain an effective (and optimised) solution

Future work

- Categories?
- More powerful winning conditions