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
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:

$L_P = \text{possible program executions}$

$L_\varphi = \text{valid executions}$

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

$L_P = \text{possible program executions}$

$L_\varphi = \text{valid executions}$
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 \( \mathcal{L}_P \)
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
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 [Podolski et al. since 2010]
CEGAR Loop

Define $\mathcal{L}_S := \mathcal{L}_\varphi$
CEGAR Loop

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

$\mathcal{L}_{CF} \subseteq \mathcal{L}_S$?
CEGAR Loop

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

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

\[
\text{Init } \mathcal{L}_S := \mathcal{L}_\varphi \\
\mathcal{L}_{CF} \subseteq \mathcal{L}_S ? \xrightarrow{\text{yes}} \text{return } P \models \varphi \\
\text{no } w \in \mathcal{L}_{CF} \setminus \mathcal{L}_S \\
w \in \mathcal{L}_P ?
\]
CEGAR Loop

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

$\mathcal{L}_{CF} \subseteq \mathcal{L}_S$ ?

- yes $\Rightarrow$ return $P \models \varphi$
- no $w \in \mathcal{L}_{CF} \setminus \mathcal{L}_S$

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

- yes $\Rightarrow$ return $P \not\models S$
CEGAR Loop

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

$\mathcal{L}_{CF} \subseteq \mathcal{L}_S$ ?

- yes $\rightarrow$ return $P \models \varphi$
- no $w \in \mathcal{L}_{CF} \setminus \mathcal{L}_S$

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

- no $w \in \mathcal{L}_P$
- yes $\rightarrow$ return $P \not\models S$
CEGAR Loop

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

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

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

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

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

$\quad \overset{yes}{\rightarrow} \quad \text{return } P \not\models S$

$\quad \overset{yes}{\rightarrow} \quad \text{return } P \models \varphi$
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 $w \in \mathcal{L}_{CF} \setminus \mathcal{L}_S$

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

- no $w \in \mathcal{L}_P$?
  - yes $\rightarrow$ return $P \not\models S$
  - no $\rightarrow$ return $P \not\models S$

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
CEGAR Illustration
CEGAR Illustration
CEGAR Illustration
Language Synthetic Synthesis
Synthesis

Why write a bad program and check it’s ok?
  - 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 $\varphi$?
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 $\varphi$?

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

\[ F = rd(x, 0) \land rd(x, 1) H \]

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

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

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

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

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

no  yes

return

$\forall s. P@s \not\models S$

Algorithmic challenges:
Game $\exists s. \mathcal{L}_{CF@s} \subseteq \mathcal{L}_S$
Membership $w \in \mathcal{L}_P$
Extrapolation $w \leadsto \mathcal{L}_w$
Higher-Order Inclusion Games
Given a

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

\[
\begin{align*}
  q_0 \quad & \quad \xrightarrow{a} \quad q_0 \\
  q_0 \quad & \quad \xrightarrow{b} \quad q_1 \\
  q_1 \quad & \quad \xrightarrow{a} \quad q_1
\end{align*}
\]
Safety Games

We study safety games:

Can Player avoid generating a word $w \notin \mathcal{L}_A$?
$S_\circ = F_\circ G_\square$

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

$G_\square x = x \land b x$
Example Play

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

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

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

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

\[ S_\circ = F_\circ \text{G} \]

\[
\begin{align*}
F_\circ f &= a (F_\circ f) \lor a (f b) \\
G \quad x &= x \wedge b x
\end{align*}
\]
Example Play

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

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

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

\[ a \]

\[ F_\circ \]

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

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

$G x = x \land b \ x$

a

| F \circ | G □ |
Example Play

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

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

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

\[ a \]

\[ G \square \]

\[ b \]
Example Play

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

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

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

\[ G_\Box \]

\[ b \]
Example Play

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

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

\[ G_{\square} x = x \land b x \]

\[ \begin{array}{c}
  a \\
  \downarrow \\
  a \\
  \downarrow \\
  b \\
  \downarrow \\
  b 
\end{array} \]
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

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

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

Theorem
Given a higher-order game $G$ and regular specification $A$, determining the winning of $G$ wrt $A$ is $k$-EXPTIME-complete for an order-$k$ scheme.

Such a result is already known

- Determinize $A$
- 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
Related Work

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
Boolean formula representing game

\[ S = a \lor b \]

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

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

$$S = a \lor b$$

$$\llbracket S \rrbracket = a \lor b$$

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

Formulas may be infinite:

$$\llbracket S \rrbracket = (w_1 \lor w_2) \land (w_3 \lor w_4 \lor (w_4 \land \cdots$$
Boolean formula representing game

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

A proposition \( w \) is true iff \( w \not\in \mathcal{L}_A \).
Formulas may be infinite:

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

The semantics of a function is given as a function

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

We compute the semantics via recursive equations

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

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

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!
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:

- \( w \text{ true iff } w \notin L_A \)
- \( \alpha(w) \text{ true iff } q_0 \notin \alpha(w) \)

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 $k$-EXPTIME-complete for an order-$k$ scheme
- We need a second abstraction into an optimised domain

Winning region and strategy
- For any term $[t]$ 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 $k$-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