Domains for Higher-Order Games

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

MFCS 2017

Overview – The Problem

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 ω-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 φ

Question: Does runtime behaviour of P satisfy φ?

Verification Problem

The verification problem:

Given: Source code of a program P and a specification φ

Question: Does runtime behaviour of P satisfy φ?

Language-theoretic approach:

$\mathcal{L}_P$ = possible program executions

$\mathcal{L}_\varphi$ = valid executions

Decide: $\mathcal{L}_P \subseteq \mathcal{L}_\varphi$

The Good and Bad

$\mathcal{L}_P =$ possible program executions

$\mathcal{L}_\varphi =$ valid executions

The Good and Bad

$\mathcal{L}_P$ = possible program executions

$\mathcal{L}_\varphi$ = valid executions

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

Bad: $\mathcal{L}_P$ usually complicated...

The Good and Bad

$\mathcal{L}_P$ = possible program executions

$\mathcal{L}_\varphi$ = 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

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

$$\downarrow$$

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

CEGAR Loop

$$
\begin{tikzcd}
\text{Init } \mathcal{L}_S := \mathcal{L}_\varphi \arrow[d] & \\
\mathcal{L}_{CF} \subseteq \mathcal{L}_S \ ? \arrow[r, "\text{yes}"] & \text{return } P \models \varphi
\end{tikzcd}
$$

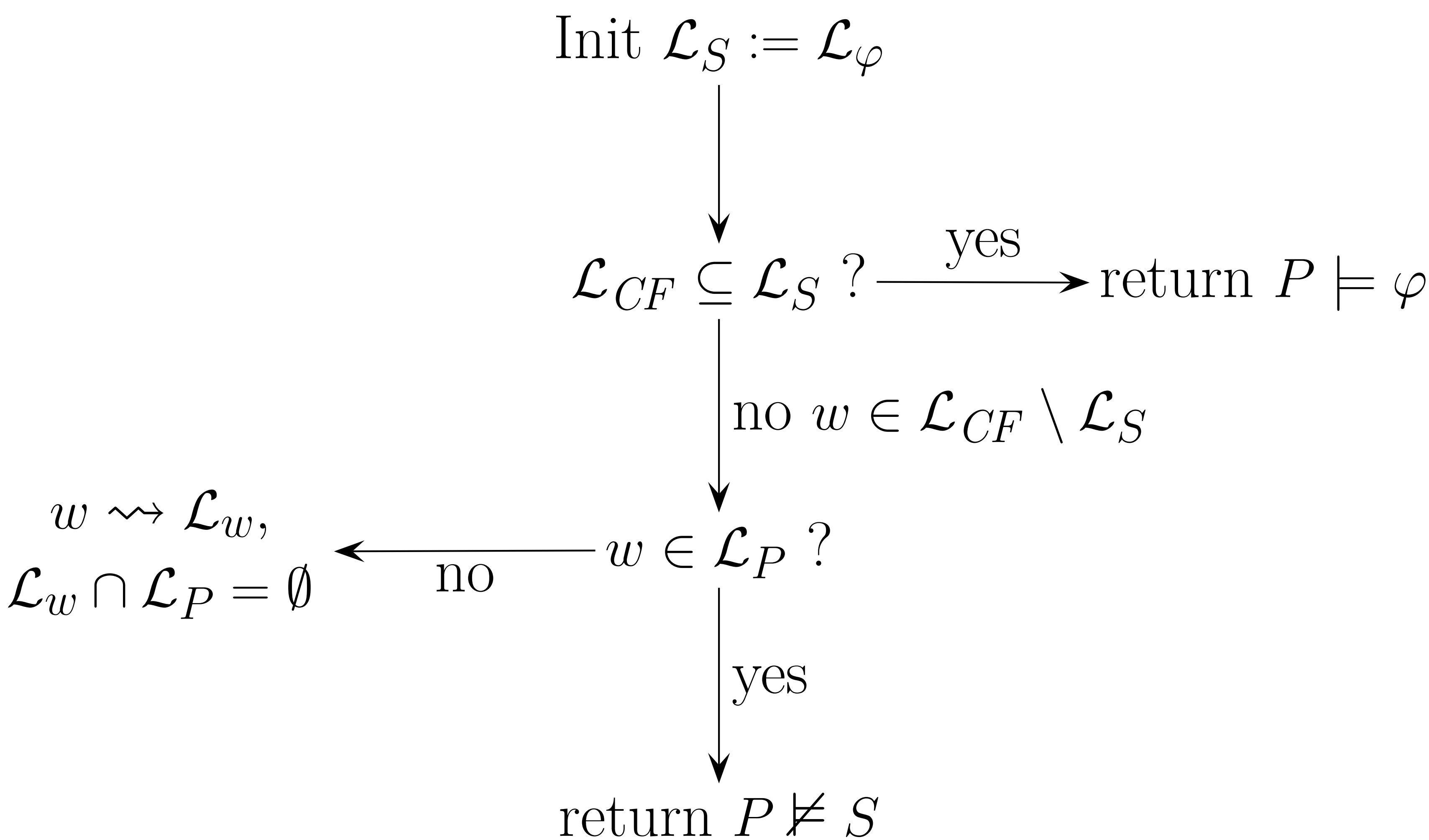
CEGAR Loop

$$
\begin{tikzcd}
\text{Init } \mathcal{L}_S := \mathcal{L}_\varphi \arrow[d] & \\
\mathcal{L}_{CF} \subseteq \mathcal{L}_S \text{ ?} \arrow[r, "\text{yes}"] \arrow[d, "\text{no } w \in \mathcal{L}_{CF} \setminus \mathcal{L}_S"] & \text{return } P \models \varphi \\
w \in \mathcal{L}_P \text{ ?} &
\end{tikzcd}
$$

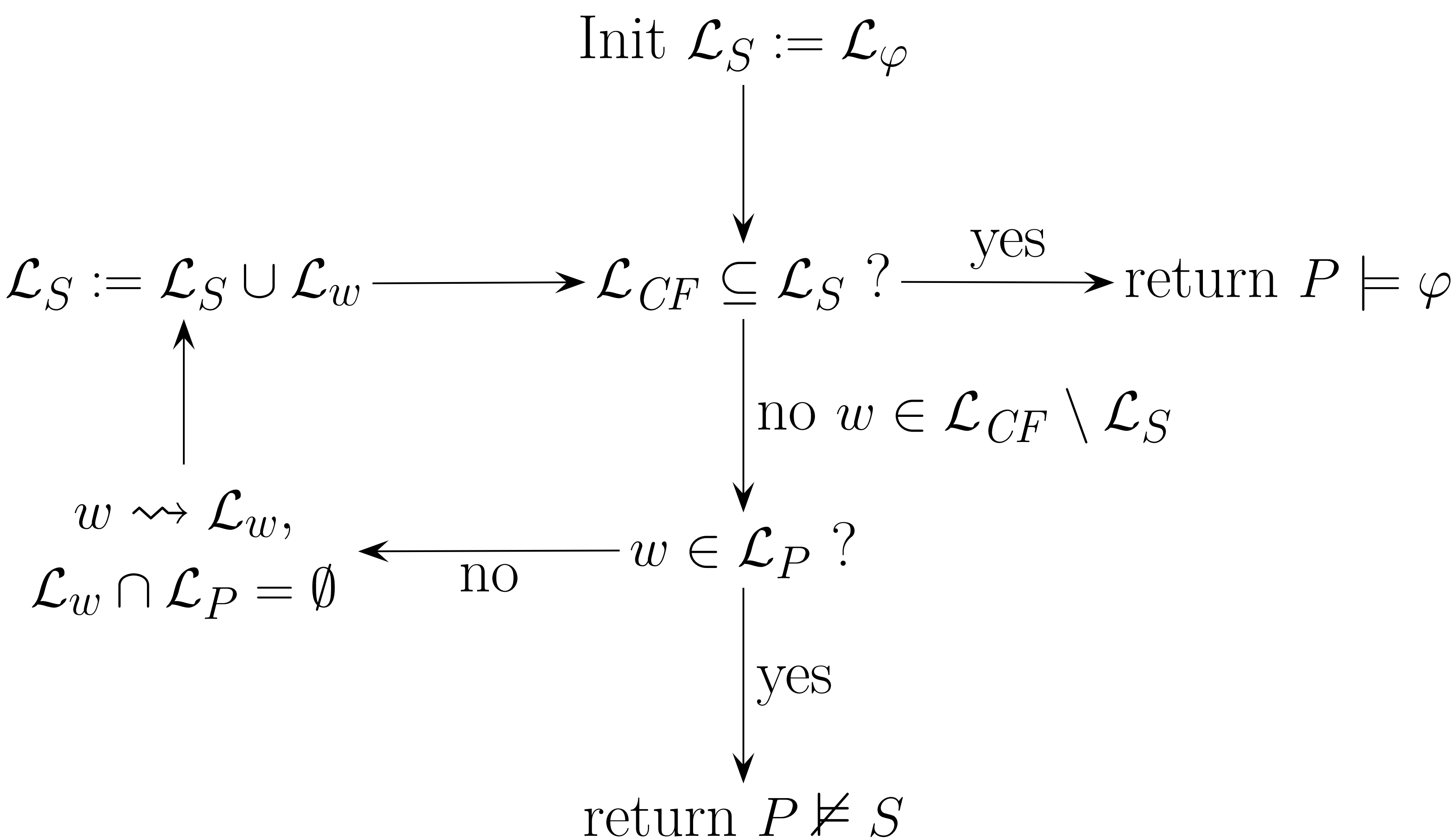
CEGAR Loop

$$\begin{tikzcd} \text{Init } \mathcal{L}_S := \mathcal{L}_\varphi \arrow[d] & \\ \mathcal{L}_{CF} \subseteq \mathcal{L}_S\ ? \arrow[r, "\text{yes}"] \arrow[d, "\text{no } w \in \mathcal{L}_{CF} \setminus \mathcal{L}_S"] & \text{return } P \models \varphi \\ w \in \mathcal{L}_P\ ? \arrow[d, "\text{yes}"] & \\ \text{return } P \not\models S & \end{tikzcd}$$

CEGAR Loop


Init $\mathcal{L}_S := \mathcal{L}_\varphi$
$\mathcal{L}_{CF} \subseteq \mathcal{L}_S$?
yes
return $P \models \varphi$
no $w \in \mathcal{L}_{CF} \setminus \mathcal{L}_S$
$w \in \mathcal{L}_P$?
no
$w \rightsquigarrow \mathcal{L}_w$,
$\mathcal{L}_w \cap \mathcal{L}_P = \emptyset$
yes
return $P \not\models S$

CEGAR Loop



Init $\mathcal{L}_S := \mathcal{L}_\varphi$
$\mathcal{L}_S := \mathcal{L}_S \cup \mathcal{L}_w$
$\mathcal{L}_{CF} \subseteq \mathcal{L}_S$?
yes
return $P \models \varphi$
no $w \in \mathcal{L}_{CF} \setminus \mathcal{L}_S$
$w \rightsquigarrow \mathcal{L}_w$,
$\mathcal{L}_w \cap \mathcal{L}_P = \emptyset$
no
$w \in \mathcal{L}_P$?
yes
return $P \not\models S$

CEGAR Loop

$$
\begin{tikzcd}
 & \text{Init } \mathcal{L}_S := \mathcal{L}_\varphi \arrow[d] & \\
\mathcal{L}_S := \mathcal{L}_S \cup \mathcal{L}_w \arrow[r] & \mathcal{L}_{CF} \subseteq \mathcal{L}_S \ ? \arrow[r, "\text{yes}"] \arrow[d, "\text{no } w \in \mathcal{L}_{CF} \setminus \mathcal{L}_S"] & \text{return } P \models \varphi \\
\begin{array}{c} w \rightsquigarrow \mathcal{L}_w, \\ \mathcal{L}_w \cap \mathcal{L}_P = \emptyset \end{array} \arrow[u] & w \in \mathcal{L}_P \ ? \arrow[l, "\text{no}"'] \arrow[d, "\text{yes}"] & \\
 & \text{return } P \not\models S &
\end{tikzcd}
$$

Algorithmic challenges:

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

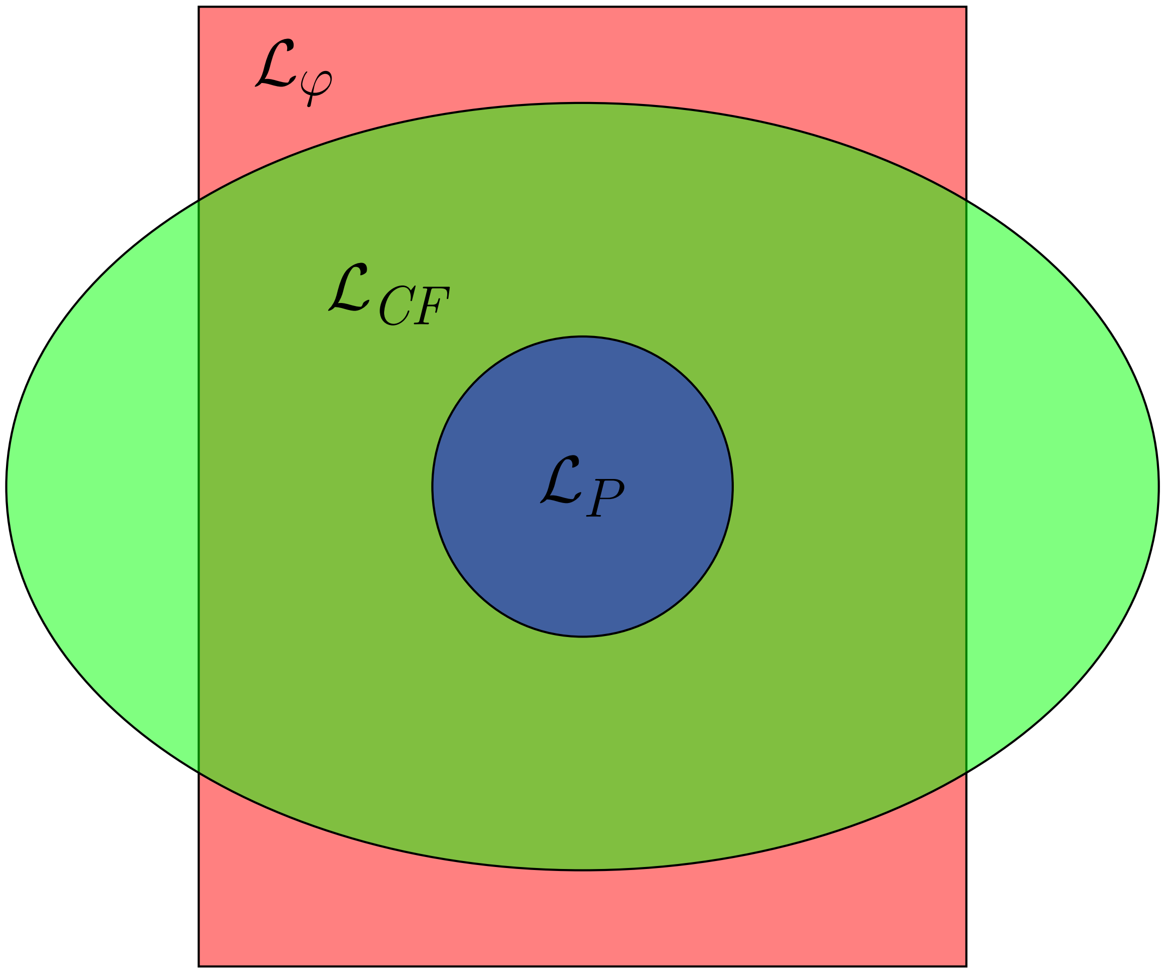
- Recursion schemes!

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

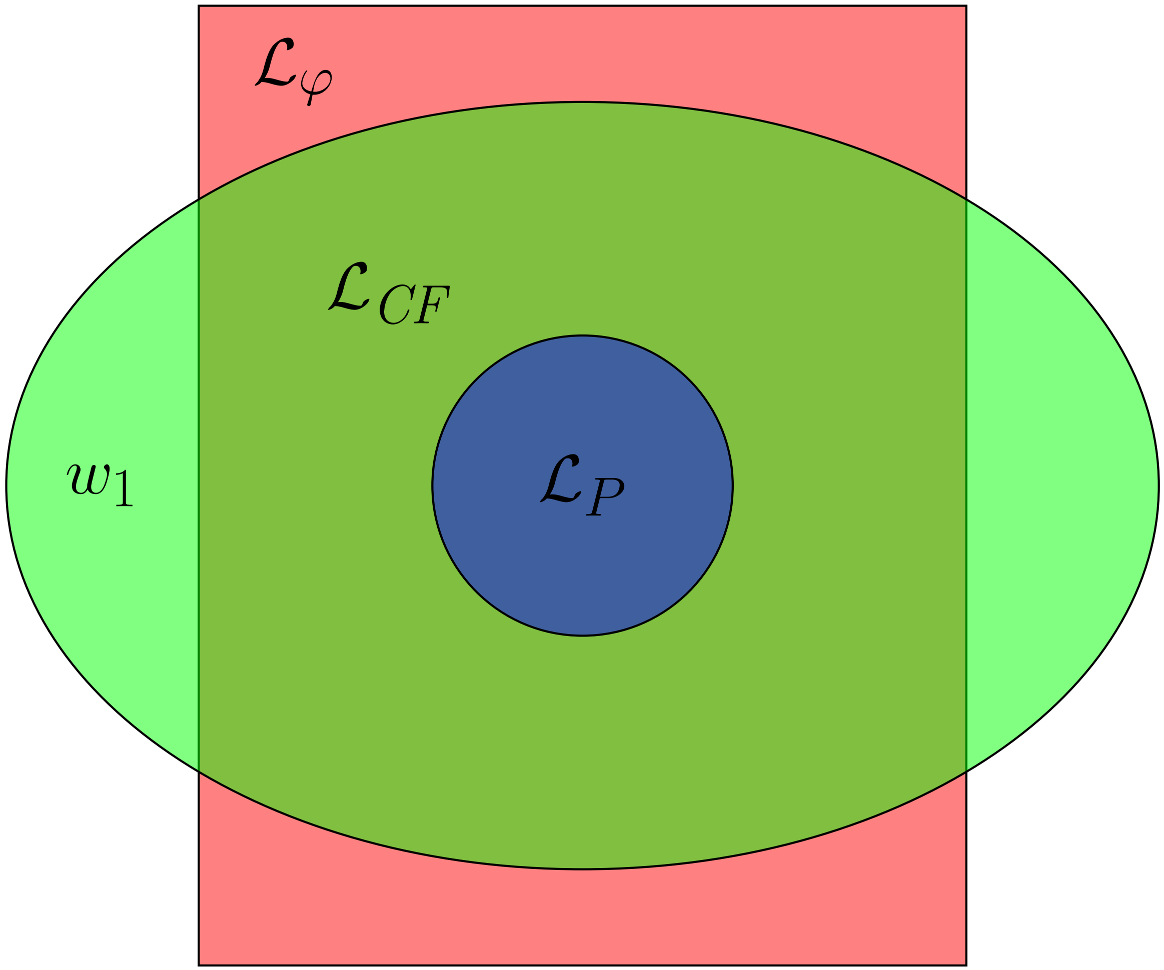
- Hoare Logic

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

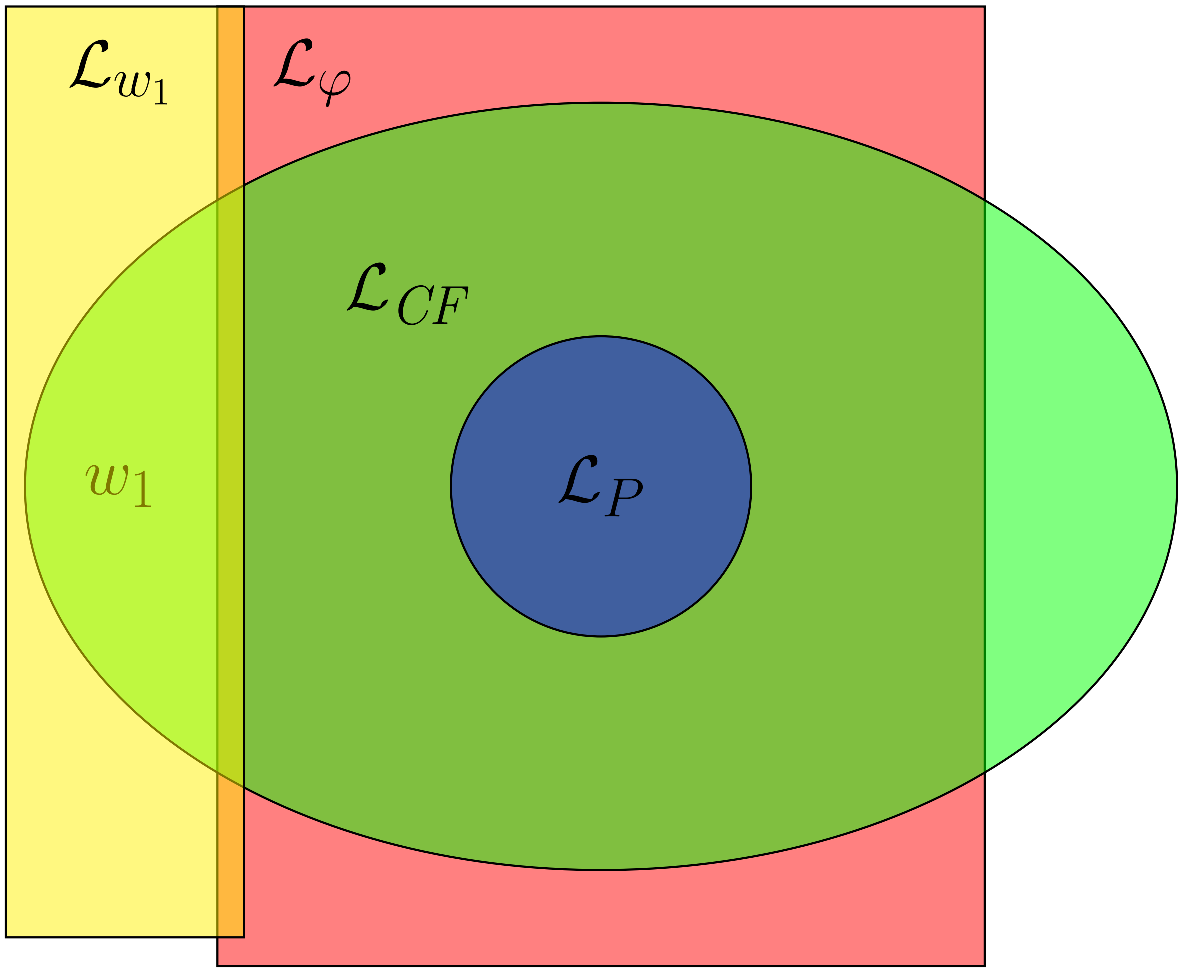
CEGAR Illustration



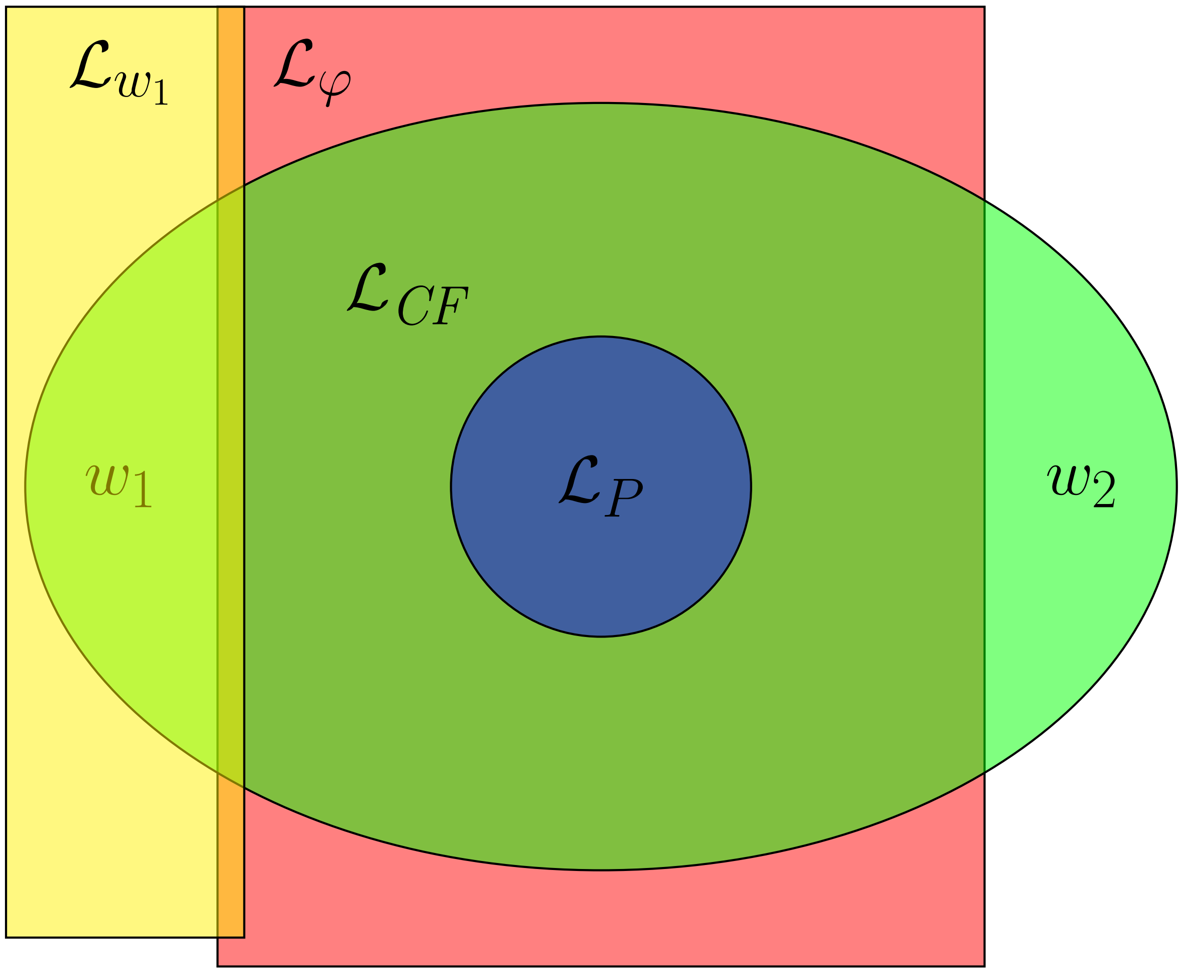
CEGAR Illustration



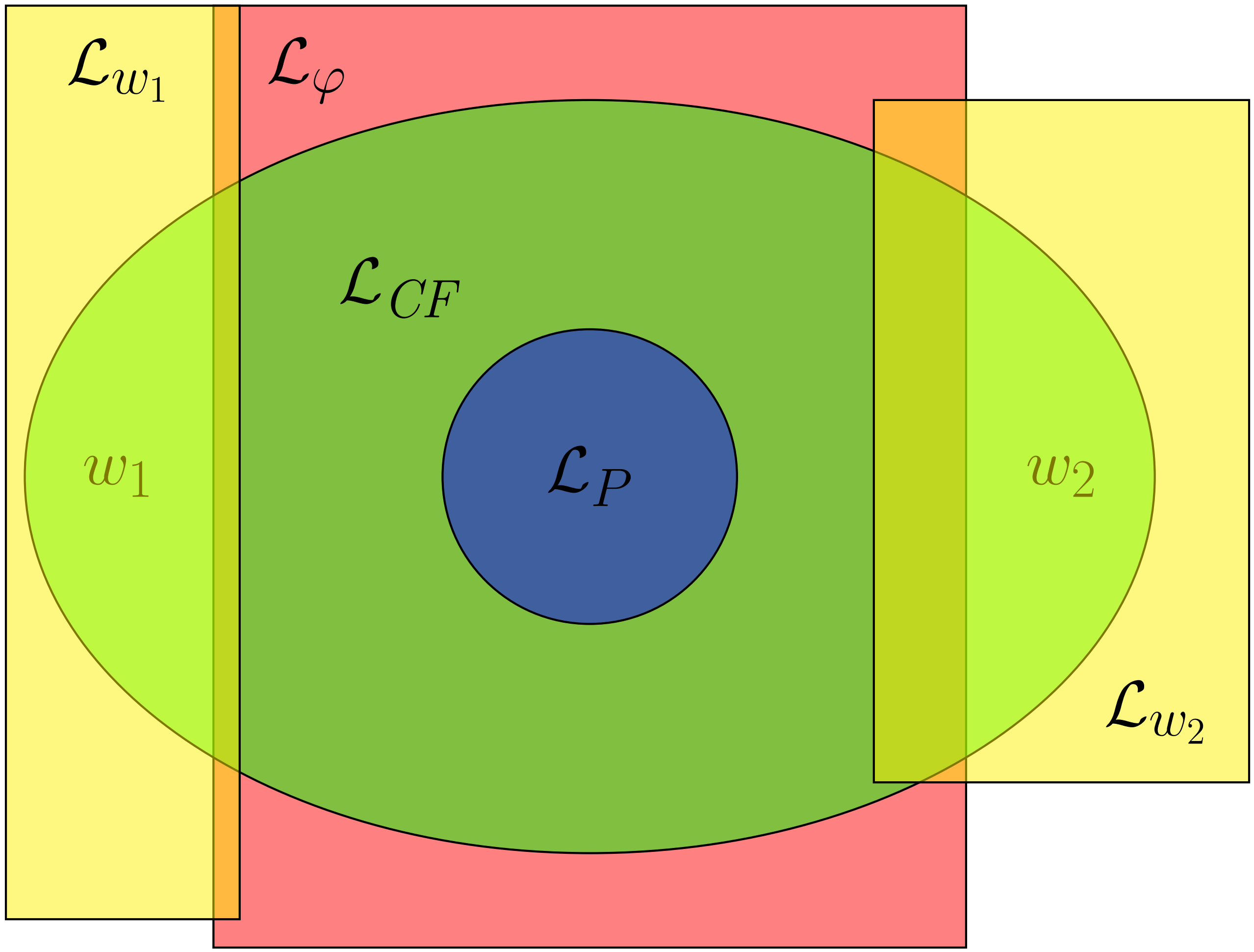
CEGAR Illustration



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 φ

Question: Is there an instantiation P' that satisfies φ?

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 φ

Question: Is there an instantiation P' that satisfies φ?

Approach:

- Language-theoretic synthesis
- CEGAR loop

Types of Non-determinism

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

Demonic

Program input:

- handle all possiblities.

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

becomes

$$F = rd(x, 0)\ G\ \wedge rd(x, 1)\ H$$

Angelic

Program branch:

- choose best.

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

becomes

$$F = G \vee H$$

Language-Theoretic Synthesis

Model as a higher-order two player perfect information game

- Player $\Box$ – uncontrollable non-determinism
- Player $\circ$ – controllable non-determinism

Is there a strategy s for $\circ$ such that

$$\mathcal{L}_{G@s} \subseteq \mathcal{L}_{\varphi}$$

I.e. when Player $\circ$ uses s all generated words are in $\mathcal{L}_{\varphi}$

Language-Theoretic Synthesis

Model as a higher-order two player perfect information game

- Player $\Box$ – uncontrollable non-determinism
- Player $\circ$ – controllable non-determinism

Is there a strategy s for $\circ$ such that

$$\mathcal{L}_{G@s} \subseteq \mathcal{L}_{\varphi}$$

I.e. when Player $\circ$ 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 $\mathcal{L}_S := \mathcal{L}_\varphi$

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

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

$w \in \mathcal{L}_P$? $\xrightarrow{\text{no}}$ $w \rightsquigarrow \mathcal{L}_w$, $\mathcal{L}_w \cap \mathcal{L}_P = \emptyset$

yes

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

Algorithmic challenges:

Game $\exists s.\mathcal{L}_{CF@s} \subseteq \mathcal{L}_S$

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

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

Higher-Order Inclusion Games

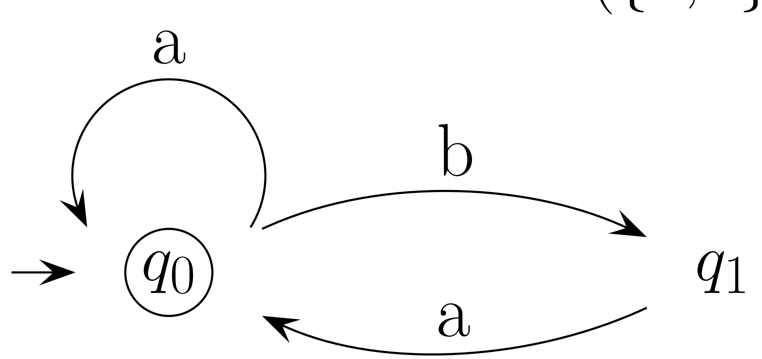
Higher-Order Games: Input

Given a

- Higher-Order Recursion Scheme
- Ownership partition of non-terminals

$$\begin{aligned} S_\circ &= F_\circ \; G_\Box \\ F_\circ \; f &= a \; (F_\circ \; f) \vee a \; (f \;\; b) \\ G_\Box \; x &= x \wedge b \; x \end{aligned}$$

- Finite automaton $\mathcal{A}$ over terminals $(\{a, b\})$



Safety Games

We study safety games:

Can Player $\circ$ avoid generating a word $w \notin \mathcal{L}_{\mathcal{A}}$?

Example Play

$$
\begin{aligned}
S_\circ &= F_\circ \; G_\Box \\
F_\circ \; f &= a \; (F_\circ \; f) \vee a \; (f \; b) \\
G_\Box \; x &= x \wedge b \; x
\end{aligned}
$$

$S_\circ$

Example Play

$$\boxed{S_\circ = F_\circ \; G_\square}$$

$$F_\circ \; f = a \; (F_\circ \; f) \vee a \; (f \; b)$$

$$G_\square \; x = x \wedge b \; x$$

$S_\circ$

Example Play

$$\boxed{S_\circ \;=\; F_\circ\ G_\square}$$

$$F_\circ\ f \;=\; a\ (F_\circ\ f) \vee a\ (f\ b)$$

$$G_\square\ x \;=\; x \wedge b\ x$$

$$\begin{array}{c} F_\circ \\ | \\ G_\square \end{array}$$

Example Play

$$S_\circ = F_\circ \; G_\Box$$

$$F_\circ \; f = a \; (F_\circ \; f) \vee a \; (f \; b)$$

$$G_\Box \; x = x \wedge b \; x$$

$$\begin{array}{c} F_\circ \\ | \\ G_\Box \end{array}$$

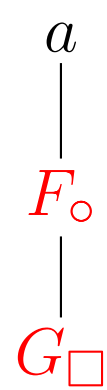
Example Play

$$
\begin{aligned}
S_\circ &= F_\circ\ G_\Box \\
\boxed{F_\circ\ f = a\ (F_\circ\ f)} &\lor a\ (f\ b) \\
G_\Box\ x &= x \land b\ x
\end{aligned}
$$

$$
\begin{tikzcd}
a \arrow[d, dash] \\
F_\circ \arrow[d, dash] \\
G_\Box
\end{tikzcd}
$$

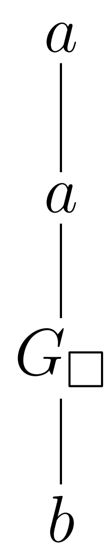
Example Play

$$
\begin{aligned}
S_\circ &= F_\circ \; G_\square \\
\boxed{F_\circ \; f} &= a \; (F_\circ \; f) \vee \boxed{a \; (f \; b)} \\
G_\square \; x &= x \wedge b \; x
\end{aligned}
$$



Example Play

$$
\begin{aligned}
S_\circ &= F_\circ \; G_\square \\
F_\circ \; f &= a \; (F_\circ \; f) \vee a \; (f \; b) \\
G_\square \; x &= x \wedge b \; x
\end{aligned}
$$



Example Play

$$
\begin{aligned}
S_\circ &= F_\circ\ G_\Box \\
F_\circ\ f &= a\ (F_\circ\ f) \vee a\ (f\ b) \\
\boxed{G_\Box\ x} &= x \wedge \boxed{b\ x}
\end{aligned}
$$

$$
\begin{tikzcd}
a \arrow[d, dash] \\
a \arrow[d, dash] \\
G_\Box \arrow[d, dash] \\
b
\end{tikzcd}
$$

Example Play

$$
\begin{aligned}
S_\circ &= F_\circ \; G_\square \\
F_\circ \; f &= a \; (F_\circ \; f) \vee a \; (f \; b) \\
G_\square \; x &= x \wedge b \; x
\end{aligned}
$$

$$
\begin{tikzcd}
a \arrow[d, dash] \\
a \arrow[d, dash] \\
b \arrow[d, dash] \\
b
\end{tikzcd}
$$

Example Play

$$
\begin{aligned}
S_{\circ} &= F_{\circ}\ G_{\square} \\
F_{\circ}\ f &= a\ (F_{\circ}\ f) \vee a\ (f\ b) \\
G_{\square}\ x &= x \wedge b\ x
\end{aligned}
$$

$$
\begin{tikzcd}
a \arrow[d, dash] \\
a \arrow[d, dash] \\
b \arrow[d, dash] \\
b
\end{tikzcd}
$$

Example Play

$$\begin{aligned} S_\circ &= F_\circ \; G_\Box \\ F_\circ \; f &= a \; (F_\circ \; f) \vee a \; (f \;\; b) \\ G_\Box \; x &= x \wedge b \; x \end{aligned}$$

$$\begin{array}{c} a \\ | \\ a \\ | \\ b \\ | \\ b \end{array}$$

Since $aabb \notin \overline{\Sigma^* bb \Sigma^*}$ 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

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

Solution Sketch: Concrete Semantics

Boolean formula representing game

$$\begin{aligned} S &= a \vee b \\ [\![S]\!] &= a \vee b \end{aligned}$$

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

Solution Sketch: Concrete Semantics

Boolean formula representing game

$$\begin{aligned} S &= a \vee b \\ [\![S]\!] &= a \vee b \end{aligned}$$

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

Formulas may be infinite:

$$[\![S]\!] = (w_1 \vee w_2) \wedge (w_3 \vee w_4 \vee (w_4 \wedge \cdots$$

Solution Sketch: Concrete Semantics

Boolean formula representing game

$$
\begin{aligned}
S &= a \vee b \\
[\![S]\!] &= a \vee b
\end{aligned}
$$

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

Formulas may be infinite:

$$[\![S]\!] = (w_1 \vee w_2) \wedge (w_3 \vee w_4 \vee (w_4 \wedge \cdots$$

The semantics of a function is given as a function

$$F : \tau_1 \to \tau_2$$
$$[\![F]\!] \in D_{\tau_1} \to D_{\tau_2}$$

Solution Sketch: Fixed Points

We compute the semantics via recursive equations

$$F = \lambda x.a\ (F\ x)$$

$$\begin{aligned}[\![F]\!] &= [\![\lambda x.a\ (F\ x)]\!] \\ &= \lambda x.[\![a]\!]\ [\![F]\!]\ x\end{aligned}$$

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 $\circ$ wins from $t : o$
- $[\![t]\!]$ is true under $\mathcal{L}_{\mathcal{A}}$

"True under $\mathcal{L}_{\mathcal{A}}$"

- a proposition w is true iff $w \notin \mathcal{L}_{\mathcal{A}}$.

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

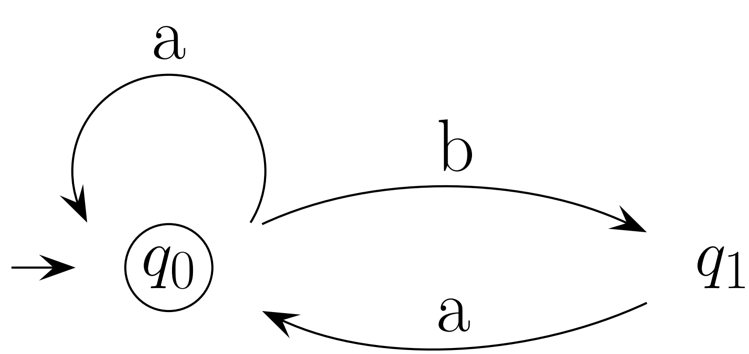
The number of propositions $w \in \Sigma^*$ is infinite

- 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 $\mathcal{A}$ 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 $w \notin \mathcal{L}_{\mathcal{A}}$
- $\alpha(w)$ 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 $[\![t]\!]$ 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 $(k+1)$-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