Outline

Regular algebraic program analysis

Semantic foundations of algebraic program analysis

Interprocedural analysis

ω-regular program analysis
Non-terminating state semantics

- Let $P$ be program, given by a control flow graph $G = (V, E)$ with entry $r$
  - Program configurations: $V \times \text{State}$ (where, say, \text{State} $\triangleq \mathbb{Z}^X$)
  - Program transition relation: $\rightarrow_P \subseteq (V \times \text{State}) \times (V \times \text{State})$
- Non-terminating state semantics: for each vertex $v$,

\[
N_v \triangleq \{ s \in \text{State} : \text{exists } c_1, c_2, \ldots \text{ with } \langle v, s \rangle \rightarrow_P c_1 \rightarrow_P c_2 \rightarrow_P \ldots \} 
\]
Non-terminating state semantics

- Let $P$ be program, given by a control flow graph $G = (V, E)$ with entry $r$
- Program configurations: $V \times \text{State}$ (where, say, $\text{State} \triangleq \mathbb{Z}^X$)
- Program transition relation: $\rightarrow_P \subseteq (V \times \text{State}) \times (V \times \text{State})$

- Non-terminating state semantics: for each vertex $v$,

$$N_v \triangleq \{ s \in \text{State} : \text{exists } c_1, c_2, \ldots \text{ with } \langle v, s \rangle \rightarrow_P c_1 \rightarrow_P c_2 \rightarrow_P \ldots \}$$

- Equational formulation – greatest solution to:

$$i := i - 2 \quad [i \neq 0]$$

$$X_r = (\langle r, a \rangle \Box X_a) \boxplus (\langle r, b \rangle \Box X_b)$$

$$X_a = \langle a, r \rangle \Box X_r$$

$$X_b = 0$$
Non-terminating state semantics

- Let $P$ be program, given by a control flow graph $G = (V, E)$ with entry $r$
- Program configurations: $V \times \text{State}$ (where, say, State $\triangleq \mathbb{Z}^X$)
- Program transition relation: $\rightarrow_P \subseteq (V \times \text{State}) \times (V \times \text{State})$
- Non-terminating state semantics: for each vertex $v$,
  \[ N_v \triangleq \{ s \in \text{State} : \text{exists } c_1, c_2, \ldots \text{ with } \langle v, s \rangle \rightarrow_P c_1 \rightarrow_P c_2 \rightarrow_P \ldots \} \]
- Equational formulation – *greatest* solution to:
  \[
  \begin{align*}
  b &\leftarrow [i = 0] \quad r \\
  i &\leftarrow i - 2 \quad [i \neq 0] \\
  a &\leftarrow [i \neq 0]
  \end{align*}
\]

\[
X_r = (\langle r, a \rangle \, \Box X_a) \, \bigoplus \, (\langle r, b \rangle \, \Box X_b)
\]

\[
X_a = \langle a, r \rangle \, \Box X_r
\]

\[
X_b = 0
\]
Non-terminating state semantics

- Let $P$ be program, given by a control flow graph $G = (V, E)$ with entry $r$
  - Program configurations: $V \times \text{State}$ (where, say, $\text{State} \triangleq \mathbb{Z}^X$)
  - Program transition relation: $\rightarrow_P \subseteq (V \times \text{State}) \times (V \times \text{State})$
- Non-terminating state semantics: for each vertex $v$,
  \[ N_v \triangleq \{ s \in \text{State} : \text{exists } c_1, c_2, \ldots \text{ with } \langle v, s \rangle \rightarrow_P c_1 \rightarrow_P c_2 \rightarrow_P \ldots \} \]
- Equational formulation – *greatest* solution to:

\[
\begin{align*}
    b & \leftarrow [i = 0] & r \\
    i & \leftarrow i - 2 & [i \neq 0] \\
    a & \leftarrow [i \neq 0] \\

    X_r = (\langle r, a \rangle \Box X_a) \uplus (\langle r, b \rangle \Box X_b) \\
    X_a = \langle 1, r \rangle \Box X_r \\
    X_b = 0
\end{align*}
\]
Closed-form solutions: \( \omega \)-regular expressions

\( \omega \)-regular expression syntax:

\[
R \in \text{RegExp}(\Sigma) ::= a \mid 0 \mid 1 \mid R_1 + R_2 \mid R_1 \cdot R_2 \mid R^*
\]
\[
S \in \omega\text{-RegExp}(\Sigma) ::= R^\omega \mid S_1 \sqcup S_2 \mid R \sqcap S
\]

\( \omega \)-regular expression semantics:

\[
\mathcal{L}[R^\omega] = \{w \in \Sigma^\omega : w = v_1v_2v_3 \ldots \text{ for some } v_1, v_2, \ldots \in \mathcal{L}[R]\}\] Infinite repetition

\[
\mathcal{L}[S_1 \sqcup S_2] = \mathcal{L}[R_1] \cup \mathcal{L}[R_2]
\] Union

\[
\mathcal{L}[R \sqcap S] = \{vw : v \in \mathcal{L}[R], w \in \mathcal{L}[S]\}
\] Prepend
\(\omega\)-regular expression semantics

- An interpretation \(\mathcal{I}\) consists of a regular algebra, a semantic function, and an \(\omega\)-algebra.
$\omega$-regular expression semantics

- An interpretation $\mathcal{I}$ consists of a regular algebra, a semantic function, and an $\omega$-algebra.
- An $\omega$-algebra $B = \langle B, \boxplus, \boxtimes, \omega \rangle$ over a regular algebra $A$ consists of:
  - A universe $B$
  - Binary operation $\boxplus : B \times B \rightarrow B$ (choice)
  - Binary operation $\boxtimes : A \times B \rightarrow B$ (prepend)
  - Unary operation $(-)^\omega : A \rightarrow B$ (omega)
Non-terminating state interpretation

- Regular algebra: binary state relations
- $\omega$-algebra Universe: set of (non-terminating) states

\[ R^\omega \triangleq \{ s : \exists s_1, s_2, \ldots \text{ with } \langle s, s_1 \rangle, \langle s_1, s_2 \rangle \cdots R \} \]  
Non-terminating states of $R$

\[ R \Box S \triangleq \{ s : \exists s' \cdot \langle s, s' \rangle \in R \land s' \in S \} \]  
Preimage

\[ S_1 \boxplus S_2 \triangleq S_1 \cup S_2 \]  
Union
Non-terminating state interpretation

- Regular algebra: binary state relations
- $\omega$-algebra Universe: set of (non-terminating) states

\[
R^\omega \triangleq \{ s : \exists s_1, s_2, \ldots \text{ with } \langle s, s_1 \rangle, \langle s_1, s_2 \rangle \cdots R \} \quad \text{Non-terminating states of } R
\]

\[
R \boxdot S \triangleq \{ s : \exists s'. \langle s, s' \rangle \in R \land s' \in S \} \quad \text{Preimage}
\]

\[
S_1 \boxplus S_2 \triangleq S_1 \cup S_2 \quad \text{Union}
\]

- Computing closed forms: Gaussian elimination
  - Key step: \( X = (R \boxdot X) \boxplus S \leadsto X = R^\omega + (R^* \boxdot S) \)
Non-terminating state interpretation

- Regular algebra: binary state relations
- $\omega$-algebra Universe: set of (non-terminating) states

\[
R^\omega \triangleq \{ s : \exists s_1, s_2, \ldots \text{ with } \langle s, s_1 \rangle, \langle s_1, s_2 \rangle \cdots R \} \quad \text{Non-terminating states of } R
\]

\[
R \boxtimes S \triangleq \{ s : \exists s'. \langle s, s' \rangle \in R \land s' \in S \} \quad \text{Preimage}
\]

\[
S_1 \boxplus S_2 \triangleq S_1 \cup S_2 \quad \text{Union}
\]

- Computing closed forms: Gaussian elimination
  - Key step: $X = (R \boxtimes X) \boxplus S \leadsto X = R^\omega + (R^* \boxtimes S)$
  - Efficient algorithm: adapt Tarjan’s path expression algorithm [Zhu & K '21]
\( \omega \)-path expressions

\[
\begin{align*}
\text{step} &= 8 \\
\text{while } (\text{true}) \text{ do} \\
\quad m &:= 0 \\
\quad \text{while } (m < \text{step}) \text{ do} \\
\qquad \text{if } (n < 0) \text{ then} \\
\qquad &\quad \text{halt} \\
\qquad \text{else} \\
\qquad &\quad m := m + 1 \\
\qquad &\quad n := n - 1
\end{align*}
\]
\textbf{\(\omega\)-path expressions}

\begin{align*}
\text{step} &= 8 \\
\text{while} \ (true) \ \text{do} \\
\text{\quad} m &:= 0 \\
\text{\quad while} \ (m < \text{step}) \ \text{do} \\
\text{\quad\quad} \text{if} \ (n < 0) \ \text{then} \\
\text{\quad\quad\quad} \text{halt} \\
\text{\quad\quad} \text{else} \\
\text{\quad\quad\quad} m &:= m + 1 \\
\text{\quad\quad\quad} n &:= n - 1
\end{align*}

\[
\langle r, a \rangle \left( \begin{array}{c}
\left( \langle a, b \rangle \langle b, c \rangle \langle c, d \rangle \langle d, e \rangle \langle e, b \rangle \right)^* \langle b, a \rangle \right)^\omega \\
\left( \langle a, b \rangle \langle b, c \rangle \langle c, d \rangle \langle d, e \rangle \langle e, b \rangle \right)^* \left( \langle b, c \rangle \langle c, d \rangle \langle d, e \rangle \langle e, b \rangle \right)^\omega
\end{array} \right)
\]

outer loop

inner loop
Non-terminating state formula interpretation

- Regular algebra: transition formulas $F(X, X')$ over a fixed set of variables $X$
- $\omega$-algebra Universe: set of state formulas $P(X)$ over $X$
  - Interpretation: any non-terminating state must satisfy $P(X)$

\[
F \square P \triangleq \exists X'.F(X, X') \land P(X')
\]

Preimage
Non-terminating state formula interpretation

- Regular algebra: transition formulas $F(X, X')$ over a fixed set of variables $X$
- $\omega$-algebra Universe: set of state formulas $P(X)$ over $X$
  - Interpretation: any non-terminating state must satisfy $P(X)$

\[
F \sqcup P \triangleq \exists X'. F(X, X') \land P(X')
\]

Preimage

\[
P_1 \sqcup P_2 \triangleq P_1 \lor P_2
\]

Union
Non-terminating state formula interpretation

- Regular algebra: transition formulas $F(X, X')$ over a fixed set of variables $X$
- $\omega$-algebra Universe: set of state formulas $P(X)$ over $X$
  - Interpretation: any non-terminating state must satisfy $P(X)$

\[
F \uplus P \triangleq \exists X'. F(X, X') \land P(X')
\]

\[
P_1 \uplus P_2 \triangleq P_1 \lor P_2
\]

\[
F^\omega \triangleq \ldots
\]

(Over-approximate) non-terminating states
Non-terminating state formula interpretation

- Regular algebra: transition formulas $F(X, X')$ over a fixed set of variables $X$
- $\omega$-algebra Universe: set of state formulas $P(X)$ over $X$
  - Interpretation: any non-terminating state must satisfy $P(X)$

\[
F \square P \triangleq \exists X'. F(X, X') \land P(X')
\]

Preimage

\[
P_1 \square P_2 \triangleq P_1 \lor P_2
\]

Union

\[
F^\omega \triangleq \ldots
\]

(Over-approximate) non-terminating states

Many different implementations
Ex. 1: Linear Ranking Functions

- A linear ranking function for a loop is a linear term that is non-negative and decreases at each iteration
  - LRF exists $\Rightarrow$ loop terminates
- For instance,

  ```
  while (lo < hi)
  if (*) then hi := hi - 1
  else lo := lo + 1
  ```
  
  Ranking function: $hi - lo$
Ex. 1: Linear Ranking Functions

- A **linear ranking function** for a loop is a linear term that is non-negative and decreases at each iteration
  - LRF exists $\Rightarrow$ loop terminates
- For instance,
  
  ```
  \textbf{while} \ (lo < hi) \\
  \quad \textbf{if} \ (*) \ \textbf{then} \ hi := hi - 1 \\
  \quad \textbf{else} \ \ lo := lo + 1
  ```

  \{ Ranking function: $hi - lo$ \}

- Existence of LRFs for polyhedral loops is decidable [Podelski & Rybalchenko ’04]
  - Loop body must be expressed as conjunction of linear inequations
Ex. 1: Linear Ranking Functions

- A linear ranking function for a loop is a linear term that is non-negative and decreases at each iteration
  - LRF exists $\Rightarrow$ loop terminates
- For instance,
  
  ```
  while (lo < hi)
  if (*) then hi := hi - 1
  else lo := lo + 1
  ```

  Ranking function: $hi - lo$

- Existence of LRFs for polyhedral loops is decidable [Podelski & Rybalchenko ’04]
  - Loop body must be expressed as conjunction of linear inequations
- Terminator: “lifts” LRF synthesis to whole programs using guess-and-check loop [Cook et al. ’2006]

  ```
  for (i = 0; i < 4096; i++)
  for (j = 0; j < 4096; j++)
  ...
  ```

  May not discover LRFs that exists
Ex. 2: Linear Ranking Functions

• A linear ranking function for a TF $F(X, X')$ is a linear term $t(X)$ such that
  1. (Non-negative) $F(X, X') \models t(X) \geq 0$
  2. (Decreasing) $F(X, X') \models t(X) - 1 \geq t(X')$
A linear ranking function for a TF $F(X, X')$ is a linear term $t(X)$ such that:

1. (Non-negative) $F(X, X') \models t(X) \geq 0$
2. (Decreasing) $F(X, X') \models t(X) - 1 \geq t(X')$

Existence of a LRF is decidable:

- $F$ has a LRF iff convex hull of $F$ has a LRF
- Existence of a LRF for a polyhedron can be checked by LP
A linear ranking function for a TF $F(X, X')$ is a linear term $t(X)$ such that

1. (Non-negative) $F(X, X') \models t(X) \geq 0$
2. (Decreasing) $F(X, X') \models t(X) - 1 \geq t(X')$

Existence of a LRF is decidable:

- $F$ has a LRF iff convex hull of $F$ has a LRF
- Existence of a LRF for a polyhedron can be checked by LP

$$F^\omega \triangleq \begin{cases} false & \text{if } F \text{ has an LRF} \\ dom(F) & \text{otherwise} \end{cases}$$

where $dom(F) \triangleq \exists X'. F(X, X')$ – set of states with $F$-successors
Ex. 2: Linear Ranking Functions

- A **linear ranking function** for a TF \( F(X, X') \) is a linear term \( t(X) \) such that
  1. (Non-negative) \( F(X, X') \models t(X) \geq 0 \)
  2. (Decreasing) \( F(X, X') \models t(X) - 1 \geq t(X') \)

- Existence of a LRF is decidable:
  - \( F \) has a LRF iff convex hull of \( F \) has a LRF
  - Existence of a LRF for a polyhedron can be checked by LP

- \( F^\omega \triangleq \begin{cases} \text{false} & \text{if } F \text{ has an LRF} \\ \text{dom}(F) & \text{otherwise} \end{cases} \)
  - where \( \text{dom}(F) \triangleq \exists X'. F(X, X') \) – set of states with \( F \)-successors

- Also works for linear **lexicographic** ranking functions [Gonnord et al. ’2015], and more
  - Completeness \( \Rightarrow \omega \) is monotone
Ex. 2: Termination analysis for free

- Any overapproximate transitive closure operator \((\_\_\_\_\_\_)^*\) induces a conditional termination analysis \((\_\_\_\_\_\_)^\omega\) [Zhu & K ’21]
Ex. 2: Termination analysis for free

- Any overapproximate transitive closure operator \((-)^*\) induces a conditional termination analysis \((-)^\omega\) [Zhu & K ’21]

- Over-approximate \(k\)-fold composition of \(F\) with

\[
F^{[k]} \triangleq (F \land k' = k - 1)^*[k' \mapsto 0]
\]
Ex. 2: Termination analysis for free

- Any overapproximate transitive closure operator \((-)^*\) induces a conditional termination analysis \((-)^\omega\) [Zhu & K ’21]
- Over-approximate \(k\)-fold composition of \(F\) with
  \[
  F^{[k]} \triangleq (F \land k' = k - 1)^*[k' \mapsto 0]
  \]
- \(F^\omega \triangleq \forall k. k \geq 0 \Rightarrow (\exists X'. F^{[k]}(X, X') \land \text{dom}(F)(X'))\)
Any overapproximate transitive closure operator $(-)^*$ induces a conditional termination analysis $(-)^\omega$ [Zhu & K ’21]

Over-approximate $k$-fold composition of $F$ with

$$F^{[k]} \triangleq (F \land k' = k - 1)^*[k' \mapsto 0]$$

$$F^\omega \triangleq \forall k. k \geq 0 \Rightarrow (\exists X'. F^{[k]}(X, X') \land \text{dom}(F)(X'))$$

while $(i \neq n)$

\[
i := i + 2
\]

$F^{[k]} : i' = i + 2k \land n' = n$

$F : i \neq n \land i' = i + 2 \land n' = n$

$\text{dom}(F) : i \neq n$

$F^\omega : i > n \lor (n - i \equiv 1 \mod 2)$

(Recurrence analysis)
Reflections on Termination of Linear Loops with Shaowei Zhu, on Wednesday

- Applies decision procedures for linear loops to general transition formulas
- Algebraic termination analysis “lifts” loop termination analysis to whole-program termination analysis
Challenges & Future Directions
The context problem

- Compositionality implies *loss of context*. When analyzing a piece of code:
  - We don’t know what initial states it might start in (forwards context)
  - We don’t know what final states might lead to a subsequent failure (backwards context)

```python
x := 0
c := 1
n := 100
while (x < n):
    x := x + c
assert (x == n)
```

**Challenge:** How can we design precise compositional analyses?
The context problem

- Compositionality implies *loss of context*. When analyzing a piece of code:
  - We don’t know what initial states it might start in (forwards context)
  - We don’t know what final states might lead to a subsequent failure (backwards context)

```java
x := 0
c := 1
n := 100
while (x < n):
    x := x + c
assert x == n
```

Challenge: How can we design precise compositional analyses?
The context problem

- Compositionality implies *loss of context*. When analyzing a piece of code:
  - We don’t know what initial states it might start in (forwards context)
  - We don’t know what final states might lead to a subsequent failure (backwards context)

```python
x := 0
c := 1
n := 100

while (x < n):
    x := x + c

assert (x == n)
```

**Challenge**: How can we design precise compositional analyses?
The context problem

- Compositionality implies *loss of context*. When analyzing a piece of code:
  - We don’t know what initial states it might start in (forwards context)
  - We don’t know what final states might lead to a subsequent failure (backwards context)

```python
x := 0
c := 1
n := 100
while (x < n):
    x := x + c
assert (x == n)
```

**Challenge**: How can we design precise compositional analyses?
The context problem

- Compositionality implies *loss of context*. When analyzing a piece of code:
  - We don’t know what initial states it might start in (forwards context)
  - We don’t know what final states might lead to a subsequent failure (backwards context)

```plaintext
x := 0
c := 1
n := 100
while (x < n):
    x := x + c
assert (x == n)
```

- **Challenge**: How can we design *precise* compositional analyses?
Scaling SMT-based algebraic analysis

- Complexity of algebraic program analysis is nearly linear in program size
  - ... assuming unit-cost for each operation of the algebra
- Transition formula algebras are not unit cost!
  
  \[
  \implies \implies \implies \implies \implies \cdot x := x + 1
  \]

- Expression DAG with \( n \) nodes, corresponding to formula of size \( 2^n \)
Scaling SMT-based algebraic analysis

- Complexity of algebraic program analysis is nearly linear in program size
  - ... assuming unit-cost for each operation of the algebra
- Transition formula algebras are not unit cost!

\[
\begin{align*}
&\cdot \implies \cdot \implies \cdot \implies \cdots \implies x := x + 1
\end{align*}
\]

- Expression DAG with \( n \) nodes, corresponding to formula of size \( 2^n \)
- **Challenge**: How can we scale SMT-based algebraic analyses?
  - Efficient reasoning about \( \lambda \) abstractions
  - Formula simplification
Recursive procedures

- Problem: the set of paths through a recursive procedure is not regular
Recursive procedures

- Problem: the set of paths through a recursive procedure is not regular
- Partial solution: the set of paths through a *linearly* recursive procedure can be captured by a tensored regular expression
Recursive procedures

- Problem: the set of paths through a recursive procedure is not regular
- Partial solution: the set of paths through a *linearly* recursive procedure can be captured by a tensored regular expression
- **Challenge:** How can the algebraic approach be applied to summarize *arbitrary* recursive procedures?
  - What is an appropriate language of “closed forms”? (recognizing context-free grammars)
  - How can we design a practical abstract interpretation of such a language?
Expanding the scope of algebraic program analysis

- Current state-of-the-art of algebraic program analysis: numerical invariant generation & termination analysis
Expanding the scope of algebraic program analysis

- Current state-of-the-art of algebraic program analysis: numerical invariant generation & termination analysis
- **Challenge**: How can we design algebraic program analyses for
  - Reasoning about arrays
  - Reasoning about memory
  - Property refutation
  - ...?
Summary

- Algebraic program analysis is a framework for building compositional program analyses
Summary

- Algebraic program analysis is a framework for building compositional program analyses
- Loop analysis *internal* to the analysis
  - Opens the door to new ways of analyzing loops
  - Can achieve theoretical guarantees about analysis behavior
  - Can use the language of algebra to reason about analysis behavior

Lots of work to be done!
Summary

- Algebraic program analysis is a framework for building compositional program analyses
- Loop analysis *internal* to the analysis
  - Opens the door to new ways of analyzing loops
  - Can achieve theoretical guarantees about analysis behavior
  - Can use the language of algebra to reason about analysis behavior
- Lots of work to be done!