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,

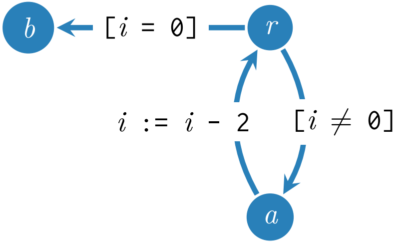
$$N_v \triangleq \{s \in \text{State} : \text{exists } c_1, c_2, \ldots \text{ with } \langle v, s \rangle \rightarrow_P c_1 \rightarrow_P c_2 \rightarrow_P \ldots\}$$

Non-terminating state semantics

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

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

- Equational formulation – *greatest* solution to:



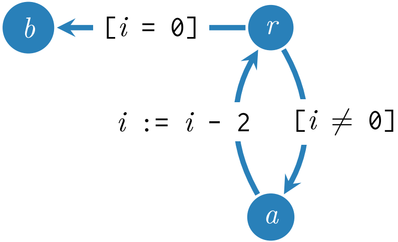
$$X_r = (\langle r, a\rangle \boxdot X_a) \boxplus (\langle r, b\rangle \boxdot X_b)$$
$$X_a = \langle a, r\rangle \boxdot X_r$$
$$X_b = 0$$

Non-terminating state semantics

- Let P be program, given by a control flow graph $G = (V, E)$ with entry r
  - Program configurations: $V \times$ 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:



$$X_r = (\langle r, a\rangle \boxdot X_a) \boxplus (\langle r, b\rangle \boxdot X_b)$$
$$X_a = \langle a, r\rangle \boxdot X_r$$
$$X_b = 0$$

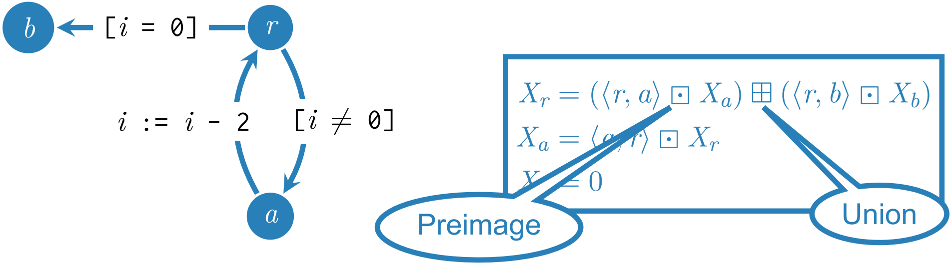
Union

Non-terminating state semantics

- Let P be program, given by a control flow graph $G = (V, E)$ with entry r
  - Program configurations: $V \times$ 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:



Closed-form solutions: ω-regular expressions

ω-regular expression syntax:

$$R \in \text{RegExp}(\Sigma) ::= a \mid 0 \mid 1 \mid R_1 + R_2 \mid R_1 \cdot R_2 \mid R^*$$
$$S \in \omega\text{-RegExp}(\Sigma) ::= R^\omega \mid S_1 \boxplus S_2 \mid R \boxdot S$$

ω-regular expression semantics:

$$\mathscr{L}[\![R^\omega]\!] = \{w \in \Sigma^\omega : w = v_1 v_2 v_3 \ldots \text{ for some } v_1, v_2, \ldots \in \mathscr{L}[\![R]\!]\} \quad \text{Infinite repetition}$$
$$\mathscr{L}[\![S_1 \boxplus S_2]\!] = \mathscr{L}[\![R_1]\!] \cup \mathscr{L}[\![R_2]\!] \quad \text{Union}$$
$$\mathscr{L}[\![R \boxdot S]\!] = \{vw : v \in \mathscr{L}[\![R]\!], w \in \mathscr{L}[\![S]\!]\} \quad \text{Prepend}$$

ω-regular expression semantics

- An **interpretation** $\mathscr{I}$ consists of a regular algebra, a semantic function, and an ω**-algebra**

ω-regular expression semantics

- An **interpretation** $\mathscr{I}$ consists of a regular algebra, a semantic function, and an ω**-algebra**
- An ω**-algebra** $\mathbf{B} = \langle B, \boxplus, \boxdot, \omega \rangle$ over a regular algebra $\mathbf{A}$ consists of
  - A universe B
  - Binary operation $\boxplus : B \times B \to B$ (*choice*)
  - Binary operation $\boxdot : A \times B \to B$ (*prepend*)
  - Unary operation $(-)^\omega : A \to B$ (*omega*)

Non-terminating state interpretation

- Regular algebra: binary state relations
- ω-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

$$R \boxdot S \triangleq \{s : \exists s'.\langle s, s'\rangle \in R \wedge s' \in S\}$$ Preimage

$$S_1 \boxplus S_2 \triangleq S_1 \cup S_2$$ Union

Non-terminating state interpretation

- Regular algebra: binary state relations
- ω-algebra Universe: set of (non-terminating) states

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

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

$$S_1 \boxplus S_2 \triangleq S_1 \cup S_2 \quad \text{Union}$$

- Computing closed forms: Gaussian elmination
  - Key step: $X = (R \boxdot X) \boxplus S \rightsquigarrow X = R^\omega + (R^* \boxdot S)$

Non-terminating state interpretation

- Regular algebra: binary state relations
- ω-algebra Universe: set of (non-terminating) states

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

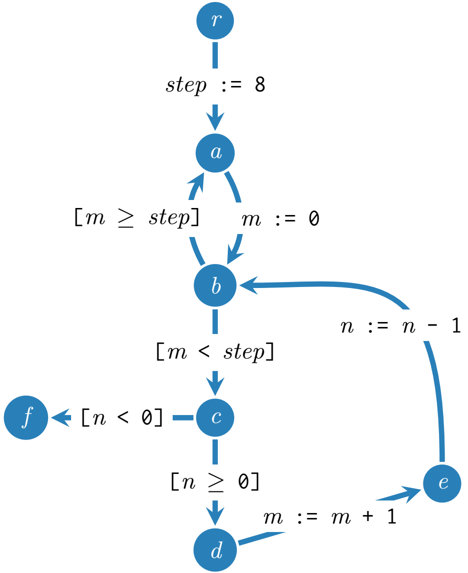
$$R \boxdot S \triangleq \{s : \exists s'. \langle s, s' \rangle \in R \land s' \in S\} \quad \text{Preimage}$$

$$S_1 \boxplus S_2 \triangleq S_1 \cup S_2 \quad \text{Union}$$

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

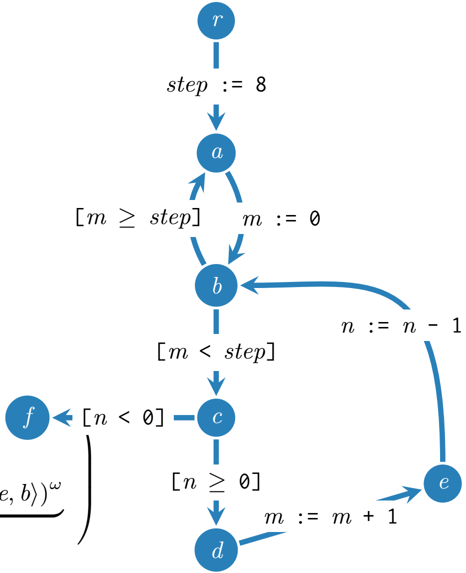
ω-path expressions

```
step = 8
while (true) do
  m := 0
  while (m < step) do
    if (n < 0) then
      halt
    else
      m := m + 1
      n := n - 1
```



ω-path expressions

```
step = 8
while (true) do
  m := 0
  while (m < step) do
    if (n < 0) then
      halt
    else
      m := m + 1
      n := n - 1
```



$$\langle r, a\rangle \left(\overbrace{\left(\langle a, b\rangle \left(\langle b, c\rangle\langle c, d\rangle\langle d, e\rangle\langle e, b\rangle\right)^* \langle b, a\rangle\right)^\omega}^{\text{outer loop}} + \left(\langle a, b\rangle \left(\langle b, c\rangle\langle c, d\rangle\langle d, e\rangle\langle e, b\rangle\right)^* \langle b, a\rangle\right)^* \underbrace{\left(\langle b, c\rangle\langle c, d\rangle\langle d, e\rangle\langle e, b\rangle\right)^\omega}_{\text{inner loop}} \right)$$

Non-terminating state formula interpretation

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

$$F \boxdot P \triangleq \exists X'.F(X, X') \land P(X')$$

Preimage

Non-terminating state formula interpretation

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

$$F \boxdot P \triangleq \exists X'.F(X, X') \land P(X')$$ Preimage

$$P_1 \boxdot P_2 \triangleq P_1 \lor P_2$$ Union

Non-terminating state formula interpretation

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

$$F \boxdot P \triangleq \exists X'. F(X, X') \wedge P(X') \qquad \text{Preimage}$$

$$P_1 \boxdot P_2 \triangleq P_1 \vee P_2 \qquad \text{Union}$$

$$F^\omega \triangleq \ldots \qquad \text{(Over-approximate) non-terminating states}$$

Non-terminating state formula interpretation

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

$$F \boxdot P \triangleq \exists X'. F(X, X') \wedge P(X')$$ Preimage

$$P_1 \boxdot P_2 \triangleq P_1 \vee P_2$$ 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 ⇒ 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,

```
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

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 ⇒ 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) $F(X, X') \models t(X) - 1 \geq t(X')$

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

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

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

  where $\textit{dom}(F) \triangleq \exists X'. F(X, X')$ – set of states with F-successors
- Also works for linear *lexicographic* ranking functions [Gonnord et al. '2015], and more
  - Completeness $\Rightarrow$ ω 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

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

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

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

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

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

```
while (i ≠ n)
  i := i + 2
```

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

$F^{[k]} : i' = i{+}2k{\wedge}n' = n$ (Recurrence analysis)

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

$F^\omega : i > n \vee (n - i \equiv 1 \mathit{\ mod\ } 2)$

Advertisement

- *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

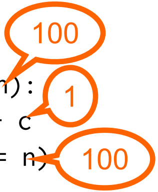
- Compositionality implies *loss of context*. When analyzing a piece of code:
  - We don't know what initial states it might start in (forwards context)
  - We don't know what final states might lead to a subsequent failure (backwards context)

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

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)

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



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)

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

$c = 1 \land n = 100 \land 0 \leq x \leq 100$

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)

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

$$\exists k.((k \geq 1 \land x < n) \lor k = 0) \land x' = x + kc...$$

The context problem

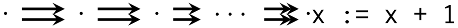
- Compositionality implies *loss of context*. When analyzing a piece of code:
  - We don’t know what initial states it might start in (forwards context)
  - We don’t know what final states might lead to a subsequent failure (backwards context)

```
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!



  - 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!

$$\cdot \rightrightarrows \cdot \rightrightarrows \cdot \rightrightarrows \cdots \rightrightarrows \cdot\, \texttt{x := x + 1}$$

  - Expression DAG with n nodes, corresponding to formula of size 2^n
- **Challenge**: How can we scale SMT-based algebraic analyses?
  - Efficient reasoning about λ 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

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!