Algebraic program analysis

- A methodology for designing program analyses based on algebra.
Algebraic program analysis

- A methodology for designing program analyses based on **algebra**.
- High-level intuition: define analysis by recursion on program syntax

\[ \mathcal{A}[-] : \text{Program} \rightarrow \text{Summary} \]
\[ \mathcal{A}[S_1; S_2] = \mathcal{A}[S_1] \cdot \mathcal{A}[S_2] \]
\[ \mathcal{A}[\text{if}(\ast)\{S_1\}\text{else}\{S_2\}] = \mathcal{A}[S_1] + \mathcal{A}[S_2] \]
\[ \mathcal{A}[\text{while}(\ast)\{S\}] = (\mathcal{A}[S])^* \]
Algebraic program analysis

- A methodology for designing program analyses based on **algebra**.
- High-level intuition: define analysis by recursion on program syntax

\[ A[\_] : \text{Program} \rightarrow \text{Summary} \]

\[ A[S_1; S_2] = A[S_1] \cdot A[S_2] \]

\[ A[\text{if}(\ast)\{S_1\}\text{else}\{S_2\}] = A[S_1] + A[S_2] \]

\[ A[\text{while}(\ast)\{S\}] = (A[S])^* \]

- Framework for understanding **compositionality**
A program analysis is **compositional** if the result for a composite program is a function of the results for its components

\[ \mathcal{A}[-] : \text{Program} \rightarrow \text{Summary} \]

\[ \mathcal{A}[S_1; S_2] = \mathcal{A}[S_1] \cdot \mathcal{A}[S_2] \]

\[ \mathcal{A}[\text{if}(\ast)\{S_1\}\text{else}\{S_2\}] = \mathcal{A}[S_1] + \mathcal{A}[S_2] \]

\[ \mathcal{A}[\text{while}(\ast)\{s\}] = (\mathcal{A}[S])^* \]
Compositional program analysis

• A program analysis is compositional if the result for a composite program is a function of the results for its components

\[ \mathcal{A}[P] : Program \rightarrow Summary \]
\[ \mathcal{A}[S_1; S_2] = \mathcal{A}[S_1] \cdot \mathcal{A}[S_2] \]
\[ \mathcal{A}[\text{if}(\ast)\{S_1\}\text{else}\{S_2\}] = \mathcal{A}[S_1] + \mathcal{A}[S_2] \]
\[ \mathcal{A}[\text{while}(\ast)\{s\}] = (\mathcal{A}[S])^* \]

• Benefits:
  • Potential to scale
  • Easy to parallelize
  • Can be applied to incomplete programs (e.g. libraries)
  • Can respond quickly to program edits
  • Enables new kinds of analysis techniques
  • …
Outline

Regular algebraic program analysis

Semantic foundations of algebraic program analysis

Interprocedural analysis

ω-regular program analysis
Algebraic path problems

• Common structure exhibited by several algorithms:
  [Aho et al. ’74, Backhouse & Carré ’75, Lehmann ’77, Tarjan ’81, …]
  • Kleene’s (NFA → regexp) algorithm
  • Warshall’s transitive closure algorithm
  • Floyd’s shortest path algorithm
  • Gauss-Jordan algorithm for solving system of linear equations
  • …
Algebraic path problems

• Common structure exhibited by several algorithms:
  [Aho et al. ’74, Backhouse & Carré ’75, Lehmann ’77, Tarjan ’81, ...]
  • Kleene’s (NFA → regexp) algorithm
  • Warshall’s transitive closure algorithm
  • Floyd’s shortest path algorithm
  • Gauss-Jordan algorithm for solving system of linear equations
  • ...

• Algebraic approach to solving path problems in graphs [Tarjan ’81]:
  1. Compute a regular expression recognizing a set of paths of interest
  2. Interpret the regular expression in a suitable algebraic structure
Path expressions

A path expression for a directed graph $G = (V, E)$: regular expression $R$ over the alphabet of edges $E$ such that each word recognized by $R$ corresponds to a path in $G$. 
Path expressions

A path expression for a directed graph $G = (V, E)$: regular expression $R$ over the alphabet of edges $E$ such that each word recognized by $R$ corresponds to a path in $G$.

Regular expression syntax:

$$R \in \text{RegExp}(\Sigma) ::= a \in \Sigma | 0 | 1 | R_1 + R_2 | R_1 R_2 | R^*$$

Regular expression semantics:

$$L[0] = \emptyset \quad \quad L[R_1 \cdot R_2] = \{w_1 w_2 : w_1 \in L[R_1], w_2 \in L[R_2]\}$$
$$L[1] = \{\epsilon\} \quad \quad L[R_1 + R_2] = L[R_1] \cup L[R_2]$$
$$L[a] = \{a\} \quad \text{For } a \in \Sigma \quad \quad L[R^*] = L[R]^*$$
Regular expression semantics

• An interpretation $\mathcal{I}$ consists of a regular algebra and a semantic function
Regular expression semantics

- An **interpretation** $\mathcal{I}$ consists of a *regular algebra* and a *semantic function*
- A *regular algebra* $A = \langle A, 0^A, 1^A, +^A, \cdot^A, *^A \rangle$ consists of
  - A set $A$ (the *universe* or *carrier* of the algebra)
  - Distinguished elements $0^A, 1^A \in A$
  - Two binary operators $\cdot^A, +^A : A \times A \rightarrow A$ (*sequencing* and *choice*)
  - A unary operator $*^A : A \rightarrow A$ (*iteration*)
Regular expression semantics

- An **interpretation** $\mathcal{I}$ consists of a *regular algebra* and a *semantic function*
- A *regular algebra* $A = \langle A, 0^A, 1^A, +^A, \cdot^A, \ast^A \rangle$ consists of
  - A set $A$ (the *universe* or *carrier* of the algebra)
  - Distinguished elements $0^A, 1^A \in A$
  - Two binary operators $\cdot^A, +^A : A \times A \rightarrow A$ (*sequencing* and *choice*)
  - A unary operator $\ast^A : A \rightarrow A$ (*iteration*)
- A *semantic function* $f : \Sigma \rightarrow A$ maps letters of the alphabet into the algebra
Regular expression semantics

• **An interpretation** $\mathcal{I}$ **consists of a regular algebra** and a semantic function

• A **regular algebra** $A = \langle A, 0^A, 1^A, +^A, \cdot^A, \ast^A \rangle$ consists of
  - A set $A$ (the *universe* or *carrier* of the algebra)
  - Distinguished elements $0^A, 1^A \in A$
  - Two binary operators $\cdot^A, +^A : A \times A \to A$ (sequencing and choice)
  - A unary operator $\ast^A : A \to A$ (iteration)

• A **semantic function** $f : \Sigma \to A$ maps letters of the alphabet into the algebra

• Define interpretation $\mathcal{I}[-] : \text{RegExp}(\Sigma) \to A$:

\[
\begin{align*}
\mathcal{I}[0] &= 0^A \\
\mathcal{I}[1] &= 1^A \\
\mathcal{I}[a] &= f(a) \quad \text{For } a \in \Sigma \\
\mathcal{I}[R_1 \cdot R_2] &= \mathcal{I}[R_1] \cdot^A \mathcal{I}[R_2] \\
\mathcal{I}[R_1 + R_2] &= \mathcal{I}[R_1] +^A \mathcal{I}[R_2] \\
\mathcal{I}[R^*] &= \mathcal{I}[R]^{\ast^A}
\end{align*}
\]
Warm-up: shortest paths

- Consider an edge-weighted graph:

- Suppose we want to compute smallest-weight path from \( a \) to \( c \)
Warm-up: shortest paths

- Consider an edge-weighted graph:

- Suppose we want to compute smallest-weight path from $a$ to $c$
  1. Compute a path expression recognizing paths from $a$ to $c$

$$\langle a, b \rangle \langle b, d \rangle (\langle d, e \rangle \langle e, d \rangle)^* \langle d, a \rangle)^* \langle a, b \rangle (\langle b, c \rangle + \langle b, d \rangle (\langle d, e \rangle \langle e, d \rangle)^* \langle d, c \rangle)$$
Warm-up: shortest paths

• Consider an edge-weighted graph:

• Suppose we want to compute smallest-weight path from $a$ to $c$
  1. Compute a path expression recognizing paths from $a$ to $c$
    $$
    (\langle a, b \rangle \langle b, d \rangle (\langle d, e \rangle \langle e, d \rangle)^* \langle d, a \rangle)^* \langle a, b \rangle (\langle b, c \rangle + \langle b, d \rangle (\langle d, e \rangle \langle e, d \rangle)^* \langle d, c \rangle)
    $$
  2. Interpret the path expression within a distance algebra
Algebra of distances

- Distance algebra universe: $\mathbb{Z} \cup \{-\infty, \infty\}$
- Operations:

\[
\begin{align*}
0^D &= \infty \\
1^D &= 0 \\
d_1 +^D d_2 &\triangleq \min(d_1, d_2) \\
d_1 \cdot^D d_2 &\triangleq d_1 + d_2 \\
^{D^k}d &\triangleq \begin{cases} 
-\infty & \text{if } d < 0 \\
0 & \text{otherwise}
\end{cases} \quad \text{Infimum of } \{nd : n \in \mathbb{N}\}
\end{align*}
\]
Interpreting a path expression DAG

\[(\langle a, b \rangle\langle b, d \rangle (\langle d, e \rangle\langle e, d \rangle)^* \langle d, a \rangle)^*\]
\[(\langle a, b \rangle (\langle b, c \rangle + \langle b, d \rangle (\langle d, e \rangle\langle e, d \rangle)^* \langle d, c \rangle))\]

- Explicit path expressions can be exponential in graph size
- DAG representation to share repeated subexpressions ⇒ polynomial size
Interpreting a path expression DAG

\[ 0^D = \infty \]
\[ 1^D = 0 \]
\[ d_1 +^D d_2 \triangleq \min(d_1, d_2) \]
\[ d_1 \cdot^D d_2 \triangleq d_1 + d_2 \]
\[ d^\ast = \begin{cases} -\infty & \text{if } d < 0 \\ 0 & \text{otherwise} \end{cases} \]

Minimum
Addition

\[ \langle a, b \rangle \]
\[ \langle d, a \rangle \]
\[ \langle b, c \rangle \]
\[ \langle b, \rangle \]
Interpreting a path expression DAG

\[
0^D = \infty \\
1^D = 0 \\
d_1 +^D d_2 \triangleq \min(d_1, d_2) \\
d_1 \cdot^D d_2 \triangleq d_1 + d_2 \\
d^{*D} \triangleq \begin{cases} 
-\infty & \text{if } d < 0 \\
0 & \text{otherwise} 
\end{cases} \\
\text{Minimum} \\
\text{Addition} \\
\inf\{nd : n \in \mathbb{N}\} \\
\langle d, a \rangle \\
\langle b, c \rangle \\
\langle b, d \rangle \\
\langle d, 0 \rangle \\
\langle d, e \rangle \\
\langle e, 2 \rangle \\
\langle a, b \rangle \\
\langle d, c \rangle
Interpreting a path expression DAG

\[
0^D = \infty \\
1^D = 0 \\
d_1 + _D d_2 \triangleq \min(d_1, d_2) \\
d_1 \cdot _D d_2 \triangleq d_1 + d_2 \\
d^* = \begin{cases} 
-\infty & \text{if } d < 0 \\
0 & \text{otherwise}
\end{cases} \\
\text{Minimum} \\
\text{Addition} \\
\inf \{nd : n \in \mathbb{N}\}
\]
Interpreting a path expression DAG

\[ 0^D = \infty \]
\[ 1^D = 0 \]
\[ d_1 +^D d_2 \triangleq \min(d_1, d_2) \]
\[ d_1 \cdot^D d_2 \triangleq d_1 + d_2 \]
\[ d^*_D \triangleq \begin{cases} -\infty & \text{if } d < 0 \\ 0 & \text{otherwise} \end{cases} \]
Parallel developments

- **Algebraic path problems**: line of work in algorithms / operations research
- **Elimination-style dataflow analysis**: dataflow analysis using algorithms resembling Gaussian elimination
  
  [Allen & Cocke ’76, Hecht & Ullman ’73, Graham & Wegman ’76]
**Convergence** [Tarjan ’81]

- Dataflow analysis as an algebraic path problem
  - **Graph:** control flow graph
  - **Algebra:** transfer functions \( L \rightarrow L \) (for some lattice \( L \)) [Graham & Wegman ’76]
• Dataflow analysis as an algebraic path problem
  • Graph: control flow graph
  • Algebra: transfer functions $L \rightarrow L$ (for some lattice $L$) [Graham & Wegman ’76]
• Efficient (almost linear time) algorithm for single-source path expression problem
  • Given: Graph $G = (V, E)$ and root vertex $r$
  • Compute: For each $v \in V$, a path expression $P(r, v)$ recognizing all paths from $r$ to $v$ in $G$
Program summarization as an algebraic path problem

\[ step = 8 \]

while (true) do
    \[ m := 0 \]
    while (\( m < step \)) do
        if (\( n < 0 \)) then
            halt
        else
            \[ m := m + 1 \]
            \[ n := n - 1 \]
Program summarization as an algebraic path problem

\[ \text{step} = 8 \]
\[ \text{while (true) do} \]
\[ \text{m} := 0 \]
\[ \text{while (m < step) do} \]
\[ \text{if (n < 0) then} \]
\[ \text{halt} \]
\[ \text{else} \]
\[ \text{m} := m + 1 \]
\[ \text{n} := n - 1 \]
Program summarization as an algebraic path problem

\[step = 8\]
\[\text{while } (true) \text{ do}\]
\[m := 0\]
\[\text{while } (m < step) \text{ do}\]
\[\text{if } (n < 0) \text{ then}\]
\[\text{halt}\]
\[\text{else}\]
\[m := m + 1\]
\[n := n - 1\]

Inner loop
[\langle r, a \rangle \langle a, b \rangle \langle b, c \rangle \langle c, d \rangle \langle d, e \rangle \langle e, b \rangle \langle b, a \rangle \rangle * \langle a, b \rangle \langle b, c \rangle \langle c, f \rangle]

Outer loop

Program summarization as an algebraic path problem

\[ \text{step} = 8 \]
\[ \text{while (true) do} \]
\[ m := 0 \]
\[ \text{while (} m < \text{step) do} \]
\[ \text{if (} n < 0 \text{) then} \]
\[ \text{halt} \]
\[ \text{else} \]
\[ m := m + 1 \]
\[ n := n - 1 \]

Recursion on regular expression \( \sim \) recursion on program syntax

\[ \mathcal{A}[\text{Program}] \rightarrow \text{Summary} \]
\[ \mathcal{A}[S_1; S_2] = \mathcal{A}[S_1] \cdot \mathcal{A}[S_2] \]
\[ \mathcal{A}[\text{if}(*\{S_1\}\text{else}\{S_2\})] = \mathcal{A}[S_1] + \mathcal{A}[S_2] \]
\[ \mathcal{A}[\text{while}(\ast\{S\})] = (\mathcal{A}[S])^* \]
Transition Formulas

- Transition formula $F(X, X')$: logical formula $\sim$ binary relation on states
  - $X$: pre-state variables
  - $X' \triangleq \{x' : x \in X\}$: post-state variables
  - $\text{tf}(x := x + 1) \triangleq x' = x + 1 \land y' = y \land z' = z$
Transition Formulas

- Transition formula $F(X, X')$: logical formula $\sim$ binary relation on states
  - $X$: pre-state variables
  - $X' \triangleq \{ x' : x \in X \}$: post-state variables
    
    \[
    tf(x := x + 1) \triangleq x' = x + 1 \land y' = y \land z' = z
    \]

- To verify an assertion:
  1. Compute path expression $R$ from entry to $\text{assert}(P)$
  2. Check $\mathcal{TF}[R] \land \neg P(X')$
    - UNSAT: assertion verified ✓
    - SAT: No conclusion
Transition Formulas

- Transition formula $F(X, X')$: logical formula $\sim$ binary relation on states
  - $X$: pre-state variables
  - $X' \triangleq \{x' : x \in X\}$: post-state variables

$$tf(x := x + 1) \triangleq x' = x + 1 \land y' = y \land z' = z$$

- To verify an assertion:
  1. Compute path expression $R$ from entry to $\text{assert}(P)$
  2. Check $\text{TF}[R] \land \neg P(X')$
     - UNSAT: assertion verified $\checkmark$
     - SAT: No conclusion

- To bound time usage:
  - Compute path expression $R$ from entry to exit
  - Redefine semantic function $tf'(e) \triangleq tf(e) \land t' = t + 1$
  - Maximize $t'$ w.r.t. $\text{TF}[R]$
**Transition Formula Algebras**

Universe: set of transition formulas $F(X, X')$ over a fixed set of variables $X$

\[
0^{\text{TF}} \triangleq false
\]  
Empty relation

\[
1^{\text{TF}} \triangleq \bigwedge_{x \in X} x' = x
\]  
Identity relation
Transition Formula Algebras

Universe: set of transition formulas \( F(X, X') \) over a fixed set of variables \( X \)

\[
0^{\text{TF}} \triangleq \text{false} \quad \text{Empty relation}
\]

\[
1^{\text{TF}} \triangleq \bigwedge_{x \in X} x' = x \quad \text{Identity relation}
\]

\[
F +^{\text{TF}} G \triangleq F \lor G \quad \text{Union}
\]
Transition Formula Algebras

Universe: set of transition formulas $F(X, X')$ over a fixed set of variables $X$

$$0^{\text{TF}} \triangleq \text{false}$$  \hspace{1cm} \text{Empty relation}

$$1^{\text{TF}} \triangleq \bigwedge_{x \in X} x' = x$$  \hspace{1cm} \text{Identity relation}

$$F +^{\text{TF}} G \triangleq F \lor G$$  \hspace{1cm} \text{Union}

$$F .^{\text{TF}} G \triangleq \exists X''. F(X, X'') \land G(X'', X')$$  \hspace{1cm} \text{Relational composition}
Transition Formula Algebras

Universe: set of transition formulas $F(X, X')$ over a fixed set of variables $X$

\[0^{\text{TF}} \triangleq \text{false}\]  
Empty relation  

\[1^{\text{TF}} \triangleq \bigwedge_{x \in X} x' = x\]  
Identity relation  

\[F +^{\text{TF}} G \triangleq F \lor G\]  
Union  

\[F \cdot^{\text{TF}} G \triangleq \exists X'' . F(X, X'') \land G(X'', X')\]  
Relational composition  

\[F^*^{\text{TF}} \triangleq \ldots\]  
Approximate transitive closure
Transition Formula Algebras

Universe: set of transition formulas $F(X, X')$ over a fixed set of variables $X$

\[
0^{\text{TF}} \triangleq \text{false} \quad \text{Empty relation}
\]

\[
1^{\text{TF}} \triangleq \bigwedge_{x \in X} x' = x \quad \text{Identity relation}
\]

\[
F +^{\text{TF}} G \triangleq F \lor G \quad \text{Union}
\]

\[
F \cdot^{\text{TF}} G \triangleq \exists X''. F(X, X'') \land G(X'', X') \quad \text{Relational composition}
\]

\[
F^{*^{\text{TF}}} \triangleq \ldots \quad \text{Approximate transitive closure}
\]

Many different implementations
Iteration

\[ (-) : TF \rightarrow TF \]

- Input: transition formula summarizing loop body
  - Regardless of structure of inner loop (nested loops, procedure calls, ...)
- Output: transition formula summarizing loop
  - Output language is the same as input language!
- Related work in CAV community: loop summarization / acceleration
Ex. 1: Predicate abstraction

- Houdini [Flanagan & Leino ’01]
  - Fix a finite set of predicates $P$.
  - Infer loop invariants of the form
    $$\left( \bigwedge_{p \in Q \subseteq P} p \right)$$
    by fixpoint computation.
Ex. 1: Predicate abstraction

- **Houdini** [Flanagan & Leino ’01]
  - Fix a finite set of predicates $P$.
  - Infer loop invariants of the form \( \bigwedge_{p \in \mathcal{Q} \subseteq P} p \) by fixpoint computation

- **Transition** predicate abstraction [Kroening et al. ’08]
  - Fix finite set of *transition* predicates $P$ such that each $p \in P$ is
    - reflexive: $X = X' \models p(X, X')$.
    - transitive: $p(X, X') \land p(X', X'') \models p(X, X''').$
  - Examples: $(x \leq x')$, $(x \geq 0 \Rightarrow x' \geq 0)$, ...
  - Non-examples: $x < x'$ $(x \geq 0 \Rightarrow (x' \leq x))$
Ex. 1: Predicate abstraction

- **Houdini** [Flanagan & Leino ’01]
  - Fix a finite set of predicates \( P \).
  - Infer loop invariants of the form \( \bigwedge_{p \in Q \subseteq P} p \) by fixpoint computation.

- **Transition** predicate abstraction [Kroening et al. ’08]
  - Fix finite set of *transition* predicates \( P \) such that each \( p \in P \) is
    - reflexive: \( X = X' \models p(X, X') \).
    - transitive: \( p(X, X') \land p(X', X'') \models p(X, X'') \).
  - Examples: \((x \leq x')\), \((x \geq 0 \Rightarrow x' \geq 0)\), ...
  - Non-examples: \( x < x' \ (x \geq 0 \Rightarrow (x' \leq x)) \)

- Iteration operator: \( F^* \triangleq \bigwedge \{p \in P : F \models p\} \)
  - No fixpoint computation (max \(|P|\) calls to an SMT solver).
Ex 2: Interval analysis

- **Interval invariant** for a loop is an inductive invariant of the form

\[ a_x \leq x \leq b_x \]

\[ x \in X \]
Ex 2: Interval analysis

- **Interval invariant** for a loop is an inductive invariant of the form

  \[ \bigwedge_{x \in X} a_x \leq x \leq b_x \]

  \[ i = 0; \]
  \[ j = 0; \]
  \[ \textbf{while} \ (i < 10 \land j \neq 20 \land j < 100) \ \{ \]
  \[ \quad i = i + 1; \]
  \[ \quad j = j + 1; \]
  \[ \} \]
Ex 2: Interval analysis

- **Interval invariant** for a loop is an inductive invariant of the form

\[
\bigwedge_{x \in X} a_x \leq x \leq b_x
\]

\[
i = 0;
\]
\[
j = 0;
\]
\[
\textbf{while } (i < 10 \land j \neq 20 \land j < 100) \quad \{ \\
\quad i = i + 1;
\quad j = j + 1;
\ \}
\]

\[
0 \leq i \leq 10 \land 0 \leq j \leq 100
\]
Ex 2: Interval analysis

- Interval invariant for a loop is an inductive invariant of the form

\[ \bigwedge_{x \in X} a_x \leq x \leq b_x \]

\[ i = 0; \]
\[ j = 0; \]
\[ \text{while } (i < 10 \land j \neq 20 \land j < 100) \quad \{ \]
\[ \quad i = i + 1; \]
\[ \quad j = j + 1; \]
\[ \} \]

\[ 0 \leq i \leq 10 \land 0 \leq j \leq 20 \]
Ex 2: Interval analysis

- **Interval invariant** for a loop is an inductive invariant of the form

\[ \bigwedge_{x \in X} a_x \leq x \leq b_x \]

\[ i = 0; \]
\[ j = 0; \]
\[ \textbf{while} \ (i < 10 \land j \neq 20 \land j < 100) \ \{ \]
\[ \quad i = i + 1; \]
\[ \quad j = j + 1; \]
\[ \} \]
**Ex 2: Interval analysis**

- **Interval invariant** for a loop is an inductive invariant of the form

\[ \bigwedge_{x \in X} a_x \leq x \leq b_x \]

\[\begin{align*}
i &= 0; \\
j &= 0; \\
\textbf{while} \ (i < 10 \land j \neq 20 \land j < 100) \ \{ \\
&\quad i = i + 1; \\
&\quad j = j + 1; \\
&\}\end{align*}\]

- Classical approach to computing interval invariants: iterative, using widening/narrowing [Cousot & Cousot ’76]
  - Computes *some* interval invariant; not necessarily *best*
Ex 2: Interval analysis

- Interval invariant for TF $F(X, X')$: for each variable $x$, a pair $a_x, b_x$ such that

$$
|= \forall X, X'. \left( \left( \bigwedge_{x \in X} a_x \leq x \leq b_x \right) \land F(X, X') \right) \Rightarrow \bigwedge_{x \in X} a_x \leq x' \leq b_x
$$

- $F^* \equiv \forall A, B. \text{Inv}(A, B) \land \left( \bigwedge_{x \in X} a_x \leq x \leq b_x \right) \Rightarrow \bigwedge_{x \in X} a_x \leq x' \leq b_x$ [Monniaux '09]

- $F^*$ entails all interval invariants of $F$. 
Ex 2: Interval analysis

- Interval invariant for TF $F(X, X')$: for each variable $x$, a pair $a_x, b_x$ such that

$$
\models \forall X, X'. \left( \left( \bigwedge_{x \in X} a_x \leq x \leq b_x \right) \land F(X, X') \right) \Rightarrow \bigwedge_{x \in X} a_x \leq x' \leq b_x
$$

- $F^* \triangleq \forall A, B. \left( Inv(A, B) \land \bigwedge_{x \in X} a_x \leq x \leq b_x \right) \Rightarrow \bigwedge_{x \in X} a_x \leq x' \leq b_x$

[Monniaux ’09]
Ex 2: Interval analysis

- Interval invariant for TF $F(X, X')$: for each variable $x$, a pair $a_x, b_x$ such that

\[
\models \forall X, X'. \left( \left( \bigwedge_{x \in X} a_x \leq x \leq b_x \right) \land F(X, X') \right) \Rightarrow \bigwedge_{x \in X} a_x \leq x' \leq b_x
\]

- $F^* \triangleq \forall A, B. \left( Inv(A, B) \land \bigwedge_{x \in X} a_x \leq x \leq b_x \right) \Rightarrow \bigwedge_{x \in X} a_x \leq x' \leq b_x$

[Monniaux ’09]

- $F^*$ entails all interval invariants of $F$. 
Ex 3: Recurrence analysis

while \( (x > 0) \) do
    if \( (y < 0) \) then
        \[ x := x + y \]
        \[ y := y - 1 \]
    else
        \[ x := x - 2 \]
        \[ y := y - 3 \]

---

\begin{align*}
\text{loop body} \quad & \quad x > 0 \\
& \land (y < 0 \land x' = x + y \land y' = y - 1) \\
& \land (y \geq 0 \land x' = x - 2 \land y' = y - 2)
\end{align*}
Ex 3: Recurrence analysis

while \( x > 0 \) do
    if \( y < 0 \) then
        \( x := x + y \)
        \( y := y - 1 \)
    else
        \( x := x - 2 \)
        \( y := y - 3 \)
end

loop body

\[ x > 0 \]
\[ \land \left( (y < 0 \land x' = x + y \land y' = y - 1) \lor (y \geq 0 \land x' = x - 2 \land y' = y - 2) \right) \]

recurrences

\[ y^{(k)} \leq y^{(k-1)} - 1 \]
\[ y^{(k)} \geq y^{(k-1)} - 3 \]
\[ (2x^{(k)} - y^{(k)}) \leq (2x^{(k-1)} - y^{(k-1)}) - 1 \]


**Ex 3: Recurrence analysis**

while \( x > 0 \) do

if \( y < 0 \) then

\[
\begin{align*}
  x &:= x + y \\
  y &:= y - 1
\end{align*}
\]

else

\[
\begin{align*}
  x &:= x - 2 \\
  y &:= y - 3
\end{align*}
\]

\[
\begin{align*}
  y^{(k)} &\leq y^{(k-1)} - 1 \\
  y^{(k)} &\geq y^{(k-1)} - 3 \\
  (2x^{(k)} - y^{(k)}) &\leq (2x^{(k-1)} - y^{(k-1)}) - 1
\end{align*}
\]

\[
\begin{align*}
  x > 0 \\
  \land \left( y < 0 \land x' = x + y \land y' = y - 1 \right) \\
  \lor \left( y \geq 0 \land x' = x - 2 \land y' = y - 2 \right)
\end{align*}
\]

loop body

\[
\begin{align*}
  y^{(k)} &\leq y^{(0)} - k \\
  y^{(k)} &\geq y^{(0)} - 3k \\
  (2x^{(k)} - y^{(k)}) &\leq (2x^{(0)} - y^{(0)}) - k
\end{align*}
\]

closed forms

recurrences
Ex 3: Recurrence analysis

while \((x > 0)\) do

if \((y < 0)\) then

\[x := x + y, \quad y := y - 1\]

else

\[x := x - 2, \quad y := y - 3\]

\[y^{(k)} \leq y^{(k-1)} - 1\]

\[y^{(k)} \geq y^{(k-1)} - 3\]

\[(2x^{(k)} - y^{(k)}) \leq (2x^{(k-1)} - y^{(k-1)}) - 1\]

\[x > 0 \land (y < 0) \land x' = x + y \land y' = y - 1\]

\[\lor (y \geq 0 \land x' = x - 2 \land y' = y - 2)\]

\[y^{(k)} \leq y^{(0)} - k\]

\[y^{(k)} \geq y^{(0)} - 3k\]

\[(2x^{(k)} - y^{(k)}) \leq (2x^{(0)} - y^{(0)}) - k\]

\[\exists k. k \geq 0 \land y' \leq y - k \land y' \geq y - 3k \land (2x' - y') \leq (2x - y) - k\]
Ex 3: Recurrence analysis

while \((x > 0)\) do

if \((y < 0)\) then

\[x := x + y\]
\[y := y - 1\]

else

\[x := x - 2\]
\[y := y - 3\]

\[x > 0\]
\[\land \left( (y < 0 \land x' = x + y \land y' = y - 1) \lor (y \geq 0 \land x' = x - 2 \land y' = y - 2) \right)\]

\[y^{(k)} \leq y^{(k-1)} - 1\]
\[y^{(k)} \geq y^{(k-1)} - 3\]
\[(2x^{(k)} - y^{(k)}) \leq (2x^{(k-1)} - y^{(k-1)}) - 1\]

\[\exists k. k \geq 0 \land y' \leq y - k \land y' \geq y - 3k \land (2x' - y') \leq (2x - y) - k\]

loop body
recurrences
closed forms
loop abstraction
Ex 3: Recurrence analysis

- A **linear recurrence relation** for a TF $F(X, X')$ is a formula of the form $a^T x' \leq a^T x + b$ entailed by $F$
  - $x/x'$ denote vectors containing $X/X'$
Ex 3: Recurrence analysis

- A **linear recurrence relation** for a TF $F(X, X')$ is a formula of the form $a^T x' \leq a^T x + b$ entailed by $F$
  - $x/x'$ denote vectors containing $X/X'$
- $\text{Rec}(F) \triangleq$ convex cone of all linear recurrence relations of $F$
  - $\text{Rec}(F) \cong$ valid inequalities of $\Delta(F) \triangleq \exists X, X'. F(X, X') \land \bigwedge_{x \in X} \delta_x = (x' - x)$

That is,

$$F \models a^T x' \leq a^T x + b \iff \Delta(F) \models a^T \delta \leq b$$

- Generators of $\text{Rec}(F)$ can be computed from convex hull of $\Delta(F)$ [Ancourt et al. ’10, Farzan & K ’2015]
  - i.e., we can compute all implied linear recurrence relations
... and many more

- Polynomial recurrence relations with polynomial / complex exponential closed forms [K et al. '2018]
- Polynomial recurrence relations with polynomial / rational exponential closed forms [K et al. '2019]
- Vector addition systems [Silverman & K '19]
- Octagonal relations [Bozga et al. '09]
- Combinations thereof
- ...
Ex 4: Affine relation analysis [Karr ’76]

• An affine relation is a TF of the form $Ax' = Bx + c$
  • Subuniverse of transition formulas
Ex 4: Affine relation analysis [Karr '76]

- An affine relation is a TF of the form $Ax' = Bx + c$
  - Subuniverse of transition formulas
- Closed under relational composition, but not disjunction
  - $F + G \triangleq$ affine hull of $F \vee G$
Ex 4: Affine relation analysis [Karr ’76]

- An **affine relation** is a TF of the form \( Ax' = Bx + c \)
  - **Subuniverse of transition formulas**
- Closed under relational composition, but not disjunction
  - \( F + G \triangleq \text{affine hull of } F \lor G \)
- Lattice of affine relations has no infinite increasing chains
  - \( 1 \subseteq 1 + F \subseteq 1 + F + (F \circ F) \subseteq \ldots \) reaches limit at some \( n \leq 2|X| \)
  - \( F^* \triangleq \sum_{i=0}^{n} F \circ \ldots \circ F \) \( i \text{ times} \) (Least solution to \( F^* \circ F + 1 = F^* \))
Designing an algebraic analysis

1 Define:
   - Semantic algebra \( \mathcal{A} = \langle A, \cdot, +, *, 0, 1 \rangle \)
   - Semantic function \( f : E \to A \)

2 Apply: Tarjan’s path expression algorithm
## Iterative vs. algebraic program analysis

<table>
<thead>
<tr>
<th>Iterative Framework</th>
<th>Algebraic Framework</th>
</tr>
</thead>
<tbody>
<tr>
<td>Join semi-lattice</td>
<td>Semantic Algebra</td>
</tr>
<tr>
<td>Abstract transformers</td>
<td>Semantic function</td>
</tr>
<tr>
<td>Chaotic iteration algorithm</td>
<td>Path-expression algorithm</td>
</tr>
</tbody>
</table>

### Key differences
- Algebraic analyses are *compositional*
- Loop analysis is *internal* to an algebraic program analysis