Program Analysis Probably Counts

Chris Hankin\textsuperscript{1}

c.hankin@imperial.ac.uk

joint work with

Alessandra Di Pierro\textsuperscript{2} and Herbert Wiklicky\textsuperscript{1}

\textsuperscript{1}\textsuperscript{1}Department of Computing, Imperial College London
\textsuperscript{2}\textsuperscript{2}Dipartimento di Informatica, Università di Verona

Computer Journal Lecture, November 2008
Outline

1. Introduction
2. Information Flow
3. Covert Channels
4. Conclusions
Outline

1. Introduction
2. Information Flow
3. Covert Channels
4. Conclusions
Outline

1. Introduction
2. Information Flow
3. Covert Channels
4. Conclusions
Outline

1. Introduction
2. Information Flow
3. Covert Channels
4. Conclusions
Program Analysis

- Aim: to determine properties of program run-time (dynamic) behavior without running it (statically)
- Based on formal model (semantics) of program
- Most interesting problems are *undecidable* – Turing’s Halting Problem
- Program analysis uses abstract semantics where issues are decidable, e.g. *rule of signs*:
  - Penalty is safe (or close) answers rather than accurate answers – false positives ($\oplus + \ominus = \{\oplus, \ominus\}$)
Program Analysis

- Aim: to determine properties of program run-time (dynamic) behavior without running it (statically)
- Based on formal model (semantics) of program
- Most interesting problems are *undecidable* – Turing’s Halting Problem
- Program analysis uses abstract semantics where issues are decidable, e.g. *rule of signs*:
- Penalty is safe (or close) answers rather than accurate answers – false positives ($\oplus + \ominus = \{\oplus, \ominus\}$)
Program Analysis

- Aim: to determine properties of program run-time (dynamic) behavior without running it (statically)
- Based on formal model (semantics) of program
- Most interesting problems are *undecidable* – Turing’s Halting Problem
- Program analysis uses abstract semantics where issues are decidable, e.g. *rule of signs*:
  
  - Penalty is safe (or close) answers rather than accurate answers – false positives ($\oplus + \ominus = \{\oplus, \ominus\}$)
Program Analysis

- Aim: to determine properties of program run-time (dynamic) behavior without running it (statically)
- Based on formal model (semantics) of program
- Most interesting problems are *undecidable* – Turing’s Halting Problem
- Program analysis uses abstract semantics where issues are decidable, e.g. *rule of signs*:
  - Penalty is safe (or close) answers rather than accurate answers – false positives ($\oplus + \ominus = \{\oplus, \ominus\}$)
Program Analysis

- Aim: to determine properties of program run-time (dynamic) behavior without running it (statically)
- Based on formal model (semantics) of program
- Most interesting problems are *undecidable* – Turing’s Halting Problem
- Program analysis uses abstract semantics where issues are decidable, e.g. *rule of signs:*
  \[ 25698 \times -471258 = -12110388084 \]
- Penalty is safe (or close) answers rather than accurate answers – false positives (\(\oplus + \ominus = \{\oplus, \ominus\}\))
Program Analysis

- Aim: to determine properties of program run-time (dynamic) behavior without running it (statically)
- Based on formal model (semantics) of program
- Most interesting problems are *undecidable* – Turing’s Halting Problem
- Program analysis uses abstract semantics where issues are decidable, e.g. *rule of signs*:
  \[ 25698 \times -471258 = -12110388084 \]

- Penalty is safe (or close) answers rather than accurate answers – false positives \( \oplus + \ominus = \{\oplus, \ominus\} \)
Program Analysis

- Aim: to determine properties of program run-time (dynamic) behavior without running it (statically)
- Based on formal model (semantics) of program
- Most interesting problems are *undecidable* – Turing’s Halting Problem
- Program analysis uses abstract semantics where issues are decidable, e.g. *rule of signs*:
  \[ 25698 \times -471258 = -12110388084 \]
  
  \[ \oplus \times \ominus = \ominus \]

- Penalty is safe (or close) answers rather than accurate answers – false positives (\( \oplus + \ominus = \{ \oplus, \ominus \} \))
Program Semantics

- Complete Partial Orders + Continuous Functions
- Labelled Transition Systems
- Hilbert Spaces + Bounded, Linear Operators
- Decidability:
  - restrict to variables (etc) in program under analysis
  - restrict sets of values to finite sets
  - truncate iterations
Program Semantics

- Complete Partial Orders + Continuous Functions
- Labelled Transition Systems
- Hilbert Spaces + Bounded, Linear Operators
- Decidability:
  - restrict to variables (etc) in program under analysis
  - restrict sets of values to finite sets
  - truncate iterations
Program Semantics

- Complete Partial Orders + Continuous Functions
- Labelled Transition Systems
- Hilbert Spaces + Bounded, Linear Operators

Decidability:
- restrict to variables (etc) in program under analysis
- restrict sets of values to finite sets
- truncate iterations
Program Semantics

- Complete Partial Orders + Continuous Functions
- Labelled Transition Systems
- Hilbert Spaces + Bounded, Linear Operators

Decidability:
- restrict to variables (etc) in program under analysis
- restrict sets of values to finite sets
- truncate iterations
Abstraction

Consider a **Concrete Domain** $C$ and an **Abstract Domain** $D$:

\[
\begin{array}{ccc}
C & \xrightarrow{A} & D \\
\downarrow T & & \downarrow T^\# \\
C & \xrightarrow{A} & D
\end{array}
\]

With an **abstraction** $A : C \rightarrow D$ and a **concretisation** $G : D \rightarrow C$:

\[T^\# = GTA\]
Consider a **Concrete Domain** $C$ and an **Abstract Domain** $D$:

\[
\begin{array}{ccc}
C & \xrightarrow{A} & D \\
\downarrow T & & \downarrow T^\# \\
C & \xrightarrow{A} & D \\
\end{array}
\]

With an abstraction $A : C \rightarrow D$ and a concretisation $G : D \rightarrow C$:

$T^\# = GTA$
Consider a Concrete Domain $\mathcal{C}$ and an Abstract Domain $\mathcal{D}$:

\[
\begin{array}{ccc}
\mathcal{C} & \xrightarrow{A} & \mathcal{D} \\
\uparrow T & & \downarrow T^\# \\
\mathcal{C} & \xrightarrow{A} & \mathcal{D}
\end{array}
\]

With an abstraction $A : \mathcal{C} \rightarrow \mathcal{D}$ and a concretisation $G : \mathcal{D} \rightarrow \mathcal{C}$:

\[T^\# = GTA\]
Abstraction

Consider a **Concrete Domain** \( C \) and an **Abstract Domain** \( D \):

\[
\begin{array}{ccc}
C & \xrightarrow{A} & D \\
\downarrow T & & \downarrow T^# \\
C & \xrightarrow{A} & D
\end{array}
\]

With an **abstraction** \( A : C \rightarrow D \) and a **concretisation** \( G : D \rightarrow C \):

\[T^# = GTA\]

Abstract Interpretation: \( (A, G) \) form a **Galois Connection**.
Abstraction

Consider a **Concrete Domain** $C$ and an **Abstract Domain** $D$:

$$
\begin{array}{c}
\text{C} \\
\text{T} \\
\text{C}
\end{array} \\
\bigg\downarrow \\
\bigg\downarrow \\
\bigg\downarrow

\begin{array}{c}
\text{A} \\
\text{D} \\
\text{A}
\end{array} \\
\bigg\downarrow \\
\bigg\downarrow

\begin{array}{c}
\text{D}
\end{array}

With an **abstraction** $A : C \rightarrow D$ and a **concretisation** $G : D \rightarrow C$:

$T^\# = GTA$

Probabilistic Abst.Int.: $(A, G)$ **Moore-Penrose Pseudo-Inverse.**
Galois Connections

Definition
Let $\mathcal{C} = (\mathcal{C}, \leq)$ and $\mathcal{D} = (\mathcal{D}, \sqsubseteq)$ be two partially ordered sets. If there are two functions $\alpha : \mathcal{C} \to \mathcal{D}$ and $\gamma : \mathcal{D} \to \mathcal{C}$ such that for all $c \in \mathcal{C}$ and all $d \in \mathcal{D}$:

$$c \leq_{\mathcal{C}} \gamma(d) \text{ iff } \alpha(c) \sqsubseteq d,$$

then $(\mathcal{C}, \alpha, \gamma, \mathcal{D})$ form a Galois connection.
Moore Penrose Pseudo-Inverse

Definition
Let $\mathcal{C}$ and $\mathcal{D}$ be two Hilbert spaces and $A : \mathcal{C} \rightarrow \mathcal{D}$ a bounded linear map. A bounded linear map $A^\dagger = G : \mathcal{D} \rightarrow \mathcal{C}$ is the Moore-Penrose pseudo-inverse of $A$ iff

(i) $A \circ G = P_A$,
(ii) $G \circ A = P_G$,

where $P_A$ and $P_G$ denote orthogonal projections onto the ranges of $A$ and $G$.

An (orthogonal) projection is an operator $P$ with $P^* = P = PP$. 
Moore Penrose Pseudo-Inverse

Definition
Let $\mathcal{C}$ and $\mathcal{D}$ be two Hilbert spaces and $A: \mathcal{C} \to \mathcal{D}$ a bounded linear map. A bounded linear map $A^\dagger = G: \mathcal{D} \to \mathcal{C}$ is the Moore-Penrose pseudo-inverse of $A$ iff

(i) $A \circ G = P_A$,
(ii) $G \circ A = P_G$,

where $P_A$ and $P_G$ denote orthogonal projections onto the ranges of $A$ and $G$.

An (orthogonal) projection is an operator $P$ with $P^* = P = PP$. 
Moore-Penrose Pseudo-Inverse II

Definition
An operator \( A \in \mathcal{B}(\mathcal{H}) \) is Moore-Penrose invertible if there exists an element \( G \in \mathcal{B}(\mathcal{H}) \) such that:

(i) \( AGA = A \),
(ii) \( GAG = G \),
(iii) \( (AG)^* = AG \),
(iv) \( (GA)^* = GA \).

If it exists \( G = A^\dagger \) is called Moore-Penrose pseudo-inverse.
Example: Function Approximation

Concrete and abstract domain are step-functions on \([a, b]\).
The set of (real-valued) step-function \(T_n\) is based on the
sub-division of the interval into \(n\) sub-intervals.
Each step function in \(T_n\) corresponds to a vector in \(\mathbb{R}^n\).
Example: Function Approximation

Concrete and abstract domain are step-functions on $[a, b]$. The set of (real-valued) step-function $\mathcal{I}_n$ is based on the sub-division of the interval into $n$ sub-intervals.

Each step function in $\mathcal{I}_n$ corresponds to a vector in $\mathbb{R}^n$. 

(5 5 6 7 8 4 3 2 8 6 6 7 9 8 8 7)
Example: Function Approximation

Concrete and abstract domain are step-functions on \([a, b]\). The set of (real-valued) step-function \(\mathcal{T}_n\) is based on the sub-division of the interval into \(n\) sub-intervals. Each step function in \(\mathcal{T}_n\) corresponds to a vector in \(\mathbb{R}^n\).
Example: Function Approximation

Concrete and abstract domain are step-functions on \([a, b]\). The set of (real-valued) step-function \(T_n\) is based on the sub-division of the interval into \(n\) sub-intervals. Each step function in \(T_n\) corresponds to a vector in \(\mathbb{R}^n\).
Example: Function Approximation

Concrete and abstract domain are step-functions on \([a, b]\). The set of (real-valued) step-function \(\mathcal{I}_n\) is based on the sub-division of the interval into \(n\) sub-intervals. Each step function in \(\mathcal{I}_n\) corresponds to a vector in \(\mathbb{R}^n\).
Example: Geometric Interpretation
Example: Geometric Interpretation
Example: Geometric Interpretation
Example: Geometric Interpretation
Example: Geometric Interpretation
Example: AI vs PAI
Example: AI vs PAI
Example: AI vs PAI
Example: AI vs PAI
Example: AI vs PAI
Denning’s notion of information flow

Following Denning we divide information flows into two classes: 
*direct* and *indirect*. Indirect flows are just the transitive flows (a flow from $x$ to $y$ followed by a flow from $y$ to $z$ implies a flow from $x$ to $z$).

The direct flows are further divided:

* Explicit flows arise from assignments; for example, $x := y + z$ causes explicit information flows from both $y$ and $z$ to $x$. 
Denning’s notion of information flow

Following Denning we divide information flows into two classes: direct and indirect. Indirect flows are just the transitive flows (a flow from $x$ to $y$ followed by a flow from $y$ to $z$ implies a flow from $x$ to $z$).

The direct flows are further divided:

- **Explicit** flows arise from assignments; for example, $x := y + z$ causes explicit information flows from both $y$ and $z$ to $x$. 
Implicit flows arise when one variable’s value influences the value assigned to another by determining the flow of control. There are two types of implicit flow (though Denning only considered the first type in detail):

**Local flows** arise from guards in conditionals:

\[
\text{if } x \text{ then } y := z \text{ else } y := w.
\]

Here there is local implicit information flow from \(x\) to \(y\), in addition to the explicit flows from \(z\) and \(w\).
Denning contd.

- *Implicit* flows arise when one variable’s value influences the value assigned to another by determining the flow of control. There are two types of implicit flow (though Denning only considered the first type in detail):

  **Global flows** arise from guards in while loops

  \[
  x := y; (\text{while } w \text{ do } x := z); \ldots.
  \]

  Here there is a global implicit flow from \( w \) to all subsequent program points, since reaching those points carries the information that the loop terminated and hence, in this example, that \( w \) was \textbf{false}.  

The Language

\[ S \in \text{Statement} \]
\[ C \in \text{Command} \]
\[ \ell \in \text{Lab} \]
\[ x \in \text{Ide} \]
\[ a \in \text{Arith-exp} \]
\[ b \in \text{Bool-exp} \]

\[
S ::= C^\ell \\
C ::= \text{skip} \mid x := a \mid S_1; S_2 \mid \\
\text{if } b \text{ then } S_1 \text{ else } S_2 \mid \text{while } b \text{ do } S \mid \\
\text{new } x. S
\]
The Analysis

We write

\[(\hat{G}, \hat{D}) \models S\]

when \((\hat{G}, \hat{D})\) is an acceptable Information Flow Analysis of the statement \(S\).

\[(\hat{G}, \hat{D}) \models \text{skip}^\ell \iff \hat{D}(\ell) \supseteq \text{Id}\]

\[(\hat{G}, \hat{D}) \models (x := a)^\ell \iff \hat{D}(\ell) \supseteq \text{Id}[x \mapsto \text{FV}(a)]\]

\[(\hat{G}, \hat{D}) \models (C_1^{\ell_1}; C_2^{\ell_2})^\ell \iff (\hat{G}, \hat{D}) \models C_1^{\ell_1} \land (\hat{G}, \hat{D}) \models C_2^{\ell_2} \land \hat{G}(\ell) \supseteq \hat{G}(\ell_1) \cup \hat{G}(\ell_2); \hat{D}(\ell_1) \land \hat{D}(\ell) \supseteq \hat{D}(\ell_2); \hat{D}(\ell_1)\]
The Analysis

We write

\[(\hat{G}, \hat{D}) \models S\]

when \((\hat{G}, \hat{D})\) is an acceptable Information Flow Analysis of the statement \(S\).

\[(\hat{G}, \hat{D}) \models \text{skip}^\ell \iff \hat{D}(\ell) \supseteq \text{Id}\]

\[(\hat{G}, \hat{D}) \models (x := a)^\ell \iff \hat{D}(\ell) \supseteq \text{Id}[x \mapsto \text{FV}(a)]\]

\[(\hat{G}, \hat{D}) \models (C_1^{\ell_1}; C_2^{\ell_2})^\ell \iff (\hat{G}, \hat{D}) \models C_1^{\ell_1} \land (\hat{G}, \hat{D}) \models C_2^{\ell_2} \land \hat{G}(\ell) \supseteq \hat{G}(\ell_1) \cup \hat{G}(\ell_2); \hat{D}(\ell_1) \land \hat{D}(\ell) \supseteq \hat{D}(\ell_2); \hat{D}(\ell_1)\]
The Security Test

Having analysed a program, \( C^\ell \), we determine that there is no breach of security if both

- \( \{ x \mid x \in H \} \cap \hat{G}(\ell) = \emptyset \), and
- \( \forall x \in L. \ \forall y \in H. x \hat{D}(\ell) y \)

i.e. there are no global information flows from high variables and no low variable depends on any high variable.

\[
\begin{align*}
  h &:= l; l := h \\
  [l \mapsto h]; [h \mapsto l]
\end{align*}
\]

\[
\begin{align*}
  l &:= h; h := l \\
  [h \mapsto l]; [l \mapsto h]
\end{align*}
\]
The Security Test

Having analysed a program, \( C^\ell \), we determine that there is no breach of security if both

- \( \{ x \mid x \in H \} \cap \hat{G}(\ell) = \emptyset \), and
- \( \forall x \in L. \nexists y \in H. x \hat{D}(\ell) y \)

i.e. there are no global information flows from high variables and no low variable depends on any high variable.

\[
\begin{align*}
    h &:= l; l := h \\
    [l \mapsto h]; [h \mapsto l]
\end{align*}
\]

\[
\begin{align*}
    l &:= h; h := l \\
    [h \mapsto l]; [l \mapsto h]
\end{align*}
\]
The Security Test

Having analysed a program, \( C^\ell \), we determine that there is no breach of security if both

- \( \{ x \mid x \in H \} \cap \hat{G}(\ell) = \emptyset \), and
- \( \forall x \in L. \ \not\exists y \in H. x \hat{D}(\ell) y \)

i.e. there are no global information flows from high variables and no low variable depends on any high variable.

\[
\begin{align*}
    h & := l; l := h \\
    [l \mapsto h]; [h \mapsto l] \\
    \end{align*}
\]

\[
\begin{align*}
    l & := h; h := l \\
    [h \mapsto l]; [l \mapsto h] \\
\end{align*}
\]
Adding quantities

- Early work on language-based security precluded the use of high security variables to affect control flow.

- Specifically, conditions were restricted to using only low security information.

- If this restriction is weakened, high security data may be leaked through the different timing behaviour of alternative control paths.

- This kind of leakage of information is said to form a covert timing channel.
Adding quantities

- Early work on language-based security precluded the use of high security variables to affect control flow.
- Specifically, conditions were restricted to using only low security information.
- If this restriction is weakened, high security data may be leaked through the different timing behaviour of alternative control paths.
- This kind of leakage of information is said to form a covert timing channel.
Adding quantities

- Early work on language-based security precluded the use of high security variables to affect control flow.
- Specifically, conditions were restricted to using only low security information.
- If this restriction is weakened, high security data may be leaked through the different timing behaviour of alternative control paths.
- This kind of leakage of information is said to form a covert timing channel.
Adding quantities

- Early work on language-based security precluded the use of high security variables to affect control flow.

- Specifically, conditions were restricted to using only low security information.

- If this restriction is weakened, high security data may be leaked through the different timing behaviour of alternative control paths.

- This kind of leakage of information is said to form a covert timing channel.
The Language: Syntax

\[
\begin{align*}
\text{op} & : = + \mid \ast \mid - \mid = \mid ! = \mid < \mid \leq \\
\text{e} & : = \text{v} \mid \text{x} \mid \text{e op e} \\
C, D & : = \text{x := e} \mid \text{skipAsn x e} \\
 & \quad \mid \text{if (e) then } C \text{ else } D \mid \text{skipIf e C} \\
 & \quad \mid \text{while (e) do } C \mid C; D \\
 & \quad \mid \text{[choose]}^p C \text{ or } D \\
\text{v} & : = n \mid \text{TRUE} \mid \text{FALSE}
\end{align*}
\]

The Language: Syntax

\[
\text{op} ::= + | * | - | = | != | < | <= \\
\text{e} ::= v | x | e \text{ op } e \\
C, D ::= x := e | \text{skipAsn } x e \\
\text{if } (e) \text{ then } C \text{ else } D | \text{skipIf } e C \\
\text{while } (e) \text{ do } C | C ; D \\
[\text{choose}]^p C \text{ or } D \\
\text{v} ::= n | \text{TRUE } | \text{FALSE}
\]

The Language: Syntax

\[ \text{op} ::= + | \ast | - | = | != | < | <= \]
\[ \text{e} ::= v \mid x \mid e \text{ op } e \]
\[ \text{C, D} ::= x := e \mid \text{skipAsn } x e \]
\[ \mid \text{if } (e) \text{ then } C \text{ else } D \mid \text{skipIf } e \ C \]
\[ \mid \text{while } (e) \text{ do } C \mid C ; D \]
\[ \mid [\text{choose}]^p C \text{ or } D \]
\[ \text{v} ::= n \mid \text{TRUE} \mid \text{FALSE} \]

Operational Semantics

(Choose)

\[ \langle E \mid \text{[choose]}^p C \text{ or } D \rangle \xrightarrow{p:t_{ch}} \langle E \mid C \rangle \]

\[ \langle E \mid \text{[choose]}^p C \text{ or } D \rangle \xrightarrow{(1-p):t_{ch}} \langle E \mid D \rangle \]

(If)

\[ E \vdash e \Downarrow \text{TRUE} \]

\[ \langle E \mid \text{if } (e) \text{ then } C \text{ else } D \rangle \xrightarrow{1:t_{e \cdot t_{br}}} \langle E \mid C \rangle \]

\[ E \vdash e \Downarrow \text{FALSE} \]

\[ \langle E \mid \text{if } (e) \text{ then } C \text{ else } D \rangle \xrightarrow{1:t_{e \cdot t_{br}}} \langle E \mid D \rangle \]
Security Types

Security levels
\[ s ::= \text{L} \mid \text{H} \]

Base types
\[ \overline{\tau} ::= \text{Int} \mid \text{Bool} \]

Security types
\[ \tau ::= \overline{\tau}_s \]

Sub-typing
\[ \frac{S_1 \leq S_2}{\overline{\tau}_{S_1} \leq \overline{\tau}_{S_2}} \]
Low Equivalence

We associate to every variable \( x \) a security level

\[
\Gamma(x) \in \bar{\tau}_{\{L,H\}}
\]

Identify environments which agree on the low variables:

\[
E_1 =_L E_2 \text{ iff } \forall l \text{ with } \Gamma(l) = \bar{\tau}_L : E_1(l) = E_2(l)
\]
Low Equivalence

We associate to every variable $x$ a security level

$$\Gamma(x) \in \bar{\tau}_{\{L,H\}}$$

Identify environments which agree on the low variables:

$$E_1 =_L E_2 \iff \forall l \text{ with } \Gamma(l) = \bar{\tau}_L : E_1(l) = E_2(l)$$
Γ-Bisimilarity (Agat)

Γ-Bisimilarity \(\sim_\Gamma\) is the largest symmetric relation on commands that satisfies:

\[ C_1 \sim_\Gamma C_2 \] if for all \(E_1, E_2\) such that \(E_1 =_L E_2\) we have

- \(\langle E_1 \mid C_1 \rangle \xrightarrow{as} \langle E'_1 \mid D_1 \rangle\) implies \(\langle E_2 \mid C_2 \rangle \xrightarrow{as} \langle E'_2 \mid D_2 \rangle\) and \(E'_1 =_L E'_2\) and \(D_1 \sim_\Gamma D_2\)

- \(\langle E_1 \mid C_1 \rangle \xrightarrow{ts \cdot \sqrt{\cdot}} E'_1\) implies \(\langle E_2 \mid C_2 \rangle \xrightarrow{ts \cdot \sqrt{\cdot}} E'_2\) and

\[ E'_1 =_L E'_2. \]
Lifted Bisimilarity

We have to deal for probabilistic programs we need to consider distributions $\text{Dist}(\text{Conf})$.

An equivalence relation $\sim \subseteq S \times S$ on $S$ can be lifted to an (equivalence) relation $\sim^* \subseteq \text{Dist}(S) \times \text{Dist}(S)$ on distributions on $S$ via

$$\mu \sim^* \nu \iff \forall [s] \in S/\sim : \mu([s]_{\sim}) = \nu([s]_{\sim}).$$
Lifted Bisimilarity

We have to deal for probabilistic programs we need to consider distributions $\text{Dist}(\text{Conf})$.

An equivalence relation $\sim \subseteq S \times S$ on $S$ can be lifted to an (equivalence) relation $\sim^* \subseteq \text{Dist}(S) \times \text{Dist}(S)$ on distributions on $S$ via

$$\mu \sim^* \nu \quad \text{iff} \quad \forall [s] \in S/\sim : \mu([s]_\sim) = \nu([s]_\sim).$$
Probabilistic Time Bisimilarity

A probabilistic time bisimilarity $\sim_{PT}=\sim$ is the largest symmetric relation on configurations such that whenever $c_1 \sim c_2$, then

$$c_1 \Rightarrow \chi_1 \text{ implies } \exists \chi_2 \text{ such that } c_2 \Rightarrow \chi_2 \text{ and } \chi_1 \sim^* L \chi_2$$

with $\langle E_1 \mid C \rangle \sim_L \langle E_2 \mid C \rangle$ iff $\forall l$ with $\Gamma(l) = \bar{\tau}_L : E_1(l) = E_2(l)$
Note: $\sim_L^* = (\sim_L)^* \neq (\sim^*)_L$
Note: $\sim^*_L = (\sim_L)^* \neq (\sim^*)_L$
Security Typing I

(Assign\(_H\)) \[
\frac{\Gamma \vdash \leq e : \tau_s \quad \Gamma \vdash x : \tau_H \quad s \leq H}{\Gamma \vdash x := e : \text{skipAsn} x e}
\]

(Assign\(_L\)) \[
\frac{\Gamma \vdash \leq e : \tau_L \quad \Gamma \vdash x : \tau_L}{\Gamma \vdash x := e : x := e}
\]

(Seq) \[
\frac{\Gamma \vdash C : C_L \quad \Gamma \vdash D : D_L}{\Gamma \vdash C ; D : C_L ; D_L}
\]
Security Typing II

(If_H) \[
\frac{\Gamma \vdash \leq e : \text{Bool}_H \quad \Gamma \vdash C : C_L \quad \Gamma \vdash D : D_L}{\Gamma \vdash \text{if} (e) \text{ then } C \text{ else } D : \text{skipIf } e \ C_L}
\]

(If_L) \[
\frac{\Gamma \vdash \leq e : \text{Bool}_L \quad \Gamma \vdash C : C_L \quad \Gamma \vdash D : D_L}{\Gamma \vdash \text{if} (e) \text{ then } C \text{ else } D : \text{if} (e) \text{ then } C_L \text{ else } D_L}
\]

(While) \[
\frac{\Gamma \vdash \leq e : \text{Bool}_L \quad \Gamma \vdash C : C_L}{\Gamma \vdash \text{while} (e) \text{ do } C : \text{while} (e) \text{ do } C_L}
\]
Security Typing III

(Choose)
\[ \Gamma \vdash C : C_L \quad \Gamma \vdash D : D_L \implies \Gamma \vdash [\text{choose}]^p C \text{ or } D : [\text{choose}]^p C_L \text{ or } D_L \]

(SkipAsn)
\[ \Gamma \vdash \text{skipAsn} \ x \ e : \text{skipAsn} \ x \ e \]

(SkipIf)
\[ \Gamma \vdash C : C_L \implies \Gamma \vdash \text{skipIf} \ e \ C : \text{skipIf} \ e \ C_L \]
Transformation I

\[ \Gamma \vdash C \leftrightarrow D | D_L \]

(Assign$_H$)

\[
\begin{align*}
\Gamma \vdash \leq e : \tau_S & \quad \Gamma \vdash = x : \tau_H \\
\Gamma \vdash x := e \leftrightarrow x := e & \quad \text{skipAsn } x \ e
\end{align*}
\]

(Assign$_L$)

\[
\begin{align*}
\Gamma \vdash \leq e : \tau_L & \quad \Gamma \vdash = x : \tau_L \\
\Gamma \vdash x := e \leftrightarrow x := e & \quad x := e
\end{align*}
\]

(Seq)

\[
\begin{align*}
\Gamma \vdash C_1 \leftrightarrow D_1 | D_{1L} & \quad \Gamma \vdash C_2 \leftrightarrow D_2 | D_{2L} \\
\Gamma \vdash C_1; C_2 \leftrightarrow D_1; D_2 | D_{1L}; D_{2L}
\end{align*}
\]
Transformation I

\[ \Gamma \vdash C \leftrightarrow D \mid D_L \]

\((\text{Assign}_H)\)

\[ \frac{\Gamma \vdash \leq e : \tau_s \quad \Gamma \vdash = x : \tau_H \quad s \leq H}{\Gamma \vdash x := e \leftrightarrow x := e \mid \text{skipAsn} \ x \ e} \]

\((\text{Assign}_L)\)

\[ \frac{\Gamma \vdash \leq e : \tau_L \quad \Gamma \vdash = x : \tau_L}{\Gamma \vdash x := e \leftrightarrow x := e \mid x := e} \]

\((\text{Seq})\)

\[ \frac{\Gamma \vdash C_1 \leftrightarrow D_1 \mid D_{1L} \quad \Gamma \vdash C_2 \leftrightarrow D_2 \mid D_{2L}}{\Gamma \vdash C_1; C_2 \leftrightarrow D_1; D_2 \mid D_{1L}; D_{2L}} \]
Transformation II (Deterministic)

\[
\begin{align*}
\Gamma \vdash \leq e : \text{Bool}_H & \quad \text{ge}(D_{1L}) = \emptyset \quad \text{ge}(D_{2L}) = \emptyset \\
\Gamma \vdash C_1 \leftrightarrow D_1 \mid D_{1L} & \quad \Gamma \vdash C_2 \leftrightarrow D_2 \mid D_{2L} \\
\Gamma \vdash \text{if } (e) \text{ then } C_1 \text{ else } C_2 & \leftrightarrow \text{if } (e) \\
& \text{ then } D_1 ; D_{2L} \\
& \text{ else } D_{1L} ; D_2 \\
| \text{ skipIff } e \ (D_{1L} ; D_{2L})
\end{align*}
\]

\[
\begin{align*}
\Gamma \vdash \leq e : \text{Bool}_L & \quad \Gamma \vdash C_1 \leftrightarrow D_1 \mid D_{1L} \quad \Gamma \vdash C_2 \leftrightarrow D_2 \mid D_{2L} \\
\Gamma \vdash \text{if } (e) \text{ then } C_1 \text{ else } C_2 & \leftrightarrow \text{if } (e) \text{ then } D_1 \text{ else } D_2 \\
& | \text{ if } (e) \text{ then } D_{1L} \text{ else } D_{2L}
\end{align*}
\]
Transformation II (Probabilistic)

\[ \Gamma \vdash \leq e : \text{Bool}_H \quad \text{ge}(D_{1L}) = \emptyset \quad \text{ge}(D_{2L}) = \emptyset \]

\[ \Gamma \vdash C_1 \hookrightarrow D_1 \mid D_{1L} \quad \Gamma \vdash C_2 \hookrightarrow D_2 \mid D_{2L} \]

\[ \Gamma \vdash \text{if } (e) \text{ then } C_1 \text{ else } C_2 \hookrightarrow \text{if } (e) \text{ then } ([\text{choose}]^p D_1 \text{ or } D_{1L}; D_{2L}) \text{ else } ([\text{choose}]^p D_2 \text{ or } D_{1L}; D_2) \mid \text{skipIf } e (D_{1L}; D_{2L}) \]

\[ \Gamma \vdash \leq e : \text{Bool}_L \]

\[ \Gamma \vdash C_1 \hookrightarrow D_1 \mid D_{1L} \quad \Gamma \vdash C_2 \hookrightarrow D_2 \mid D_{2L} \]

\[ \Gamma \vdash \text{if } (e) \text{ then } C_1 \text{ else } C_2 \hookrightarrow \text{if } (e) \text{ then } D_1 \text{ else } D_2 \mid \text{if } (e) \text{ then } D_{1L} \text{ else } D_{2L} \]
Transformation III

(While)

\[
\Gamma \vdash \leq e : \text{Bool}_L \quad \Gamma \vdash C \leftrightarrow D \mid D_L \\
\Gamma \vdash \text{while} (e) \text{ do } C \leftrightarrow \text{while} (e) \text{ do } D \\
\quad \mid \text{while} (e) \text{ do } D_L
\]

(Choose)

\[
\Gamma \vdash C_1 \leftrightarrow D_1 \mid D_{1L} \quad \Gamma \vdash C_2 \leftrightarrow D_2 \mid D_{2L} \\
\Gamma \vdash [\text{choose}]^p C_1 \text{ or } C_2 \leftrightarrow [\text{choose}]^p D_1 \text{ or } D_2 \\
\quad \mid [\text{choose}]^p D_{1L} \text{ or } D_{2L}
\]
Transformation IV

(SpringAsn)

\[ \Gamma \vdash \text{skipAsn} \ x \ e \leftrightarrow \text{skipAsn} \ x \ e \ | \ \text{skipAsn} \ x \ e \]

(SkipIf)

\[ \Gamma \vdash C \leftrightarrow D \ | \ D_L \]

\[ \Gamma \vdash \text{skipIf} \ e \ C \leftrightarrow \text{skipIf} \ e \ D \ | \ \text{skipIf} \ e \ D_L \]
Global Effects

\[
\begin{align*}
ge(x := e) &= \{x\} \\
ge(C_1; C_2) &= ge(C_1) \cup ge(C_2) \\
ge(\text{if } (e) \text{ then } C_1 \text{ else } C_2) &= ge(C_1) \cup ge(C_2) \\
ge(\text{while } (e) \text{ do } C) &= ge(C) \\
ge(\text{choose}^p C_1 \text{ or } C_2) &= ge(C_1) \cup ge(C_2) \\
ge(\text{skipAsn } x \ e) &= \emptyset \\
ge(\text{skipIf } e \ C) &= ge(C)
\end{align*}
\]
Probabilistic Program Transformation

\[\begin{align*}
  &i := 1; \\
  &\text{while } i \leq 3 \text{ do} \\
  &\quad \text{if } k[i] == 1 \text{ then} \\
  &\quad \quad s := s; \\
  &\quad \text{else} \\
  &\quad \quad \text{skip;} \\
  &\quad \text{fi;} \\
  &\quad i := i + 1; \\
  &\text{od;}
\end{align*}\]
Probabilistic Program Transformation

```plaintext
i := 1;
while i<=3 do
  if k[i]==1 then
    s := s;
  else
    skip;
  fi;
  i := i+1;
od;
```

```plaintext
i := 1;
while i<=3 do
  if k[i]==1 then
    s := s; [skip]
  else
    [s := s]; skip
  fi;
  i := i+1;
od;
```
Probabilistic Program Transformation

\[
i := 1;
\]

while \( i \leq 3 \) do

\[
\text{if } k[i] = 1 \text{ then}
\]

\[
\begin{align*}
\text{choose } p: & \quad s := s; \quad [\text{skip}] \\
\text{or } 1-p: & \quad s := s \\
\text{ro}
\end{align*}
\]

\[
\text{else}
\]

\[
\begin{align*}
\text{choose } p: & \quad \text{skip} \\
\text{or } 1-p: & \quad [s := s]; \quad \text{skip} \\
\text{ro}
\end{align*}
\]

fi;

\[
i := i+1;
\]

od;
Application: Security Tradeoffs

c(p)

$\delta'(p)$

t(p)

0 1 2 3 4 5 6 7 8 9 10

Chris Hankin
Imperial College London
Program Analysis
Commercial Applications

- **AbsInt** tools used by major car and electronic goods manufacturers such as BMW, Daimler and Bosch.

- Another example is the work of **Polyspace** (recently taken over by The MathWorks) work with Airbus on the verification of fly-by-wire software.
Embedded Systems

Henzinger and Sifakis make an eloquent case for the reappraisal of the foundations of computing. They call for a new scientific foundation which “...will systematically and even-handedly integrate computation and physicality ...”. They identify the list of issues that must be dealt with which include:

- computation and physical constraints
- nondeterminism and probabilities
- functional and performance requirements
- qualitative and quantitative analysis, and
- Boolean and real values
Concluding Remarks

- Clark, Hunt and Malacaria use information theoretic ideas as a basis for studying information flow.
- General theme of incorporating quantitative and probabilistic techniques into program analysis and reasoning is certain to expand.
- It remains to be seen which approach will eventually prevail but, whichever does, there is no doubt that the arsenal of techniques will be enriched and program analysis probably will count!

Thank you
Concluding Remarks

- Clark, Hunt and Malacaria use information theoretic ideas as a basis for studying information flow.
- General theme of incorporating quantitative and probabilistic techniques into program analysis and reasoning is certain to expand.
- It remains to be seen which approach will eventually prevail but, whichever does, there is no doubt that the arsenal of techniques will be enriched and program analysis probably will count!

Thank you