Chaotic Iterations

Mooly Sagiv
Content

- Mathematical Background
- Chaotic Iterations
- Examples
- Soundness, Precision and more examples next week
Mathematical Background

- Declaratively define
  - The result of the analysis
  - The exact solution
  - Allow comparison
A partial ordering is a binary relation \( \sqsubseteq : L \times L \rightarrow \{ \text{false, true} \} \)
- For all \( l \in L : l \sqsubseteq l \) (Reflexive)
- For all \( l_1, l_2, l_3 \in L : l_1 \sqsubseteq l_2, l_2 \sqsubseteq l_3 \Rightarrow l_1 \sqsubseteq l_3 \) (Transitive)
- For all \( l_1, l_2 \in L : l_1 \sqsubseteq l_2, l_2 \sqsubseteq l_1 \Rightarrow l_1 = l_2 \) (Anti-Symmetric)

Denoted by \((L, \sqsubseteq)\)
Posets

- In program analysis
  - \( l_1 \sqsubseteq l_2 \iff l_1 \) is more precise than \( l_2 \iff l_1 \) represents fewer concrete states than \( l_2 \)

- Examples
  - Total orders \((N, \leq)\)
  - Powersets \((P(S), \subseteq)\)
  - Powersets \((P(S), \supseteq)\)
  - Even/Odd
  - Constant propagation
    » Single variable
    » Multiple variables
  - Intervals

- Bad examples
  - \((N, <)\)
  - Non-transitive \( x \sqsubseteq y \iff x = y \lor (x = 0 \land y = 1) \lor (x = 1 \land y = 2) \)
  - Non anti-symmetric
Posets

More notations

- \( l_1 \sqsupseteq l_2 \iff l_2 \subseteq l_1 \)
- \( l_1 \sqsubset l_2 \iff l_1 \subseteq l_2 \land l_1 \neq l_2 \)
- \( l_1 \sqsupset l_2 \iff l_2 \sqsubset l_1 \)
Upper and Lower Bounds

- Consider a poset \((L, \sqsubseteq)\)
- A subset \(L' \subseteq L\) has a lower bound \(l \in L\) if for all \(l' \in L'\) : \(l \sqsubseteq l'\)
- A subset \(L' \subseteq L\) has an upper bound \(u \in L\) if for all \(l' \in L'\) : \(l' \sqsubseteq u\)
- A greatest lower bound of a subset \(L' \subseteq L\) is a lower bound \(l_0 \in L\) such that \(l \sqsubseteq l_0\) for any lower bound \(l\) of \(L'\)
- A lowest upper bound of a subset \(L' \subseteq L\) is an upper bound \(u_0 \in L\) such that \(u_0 \sqsubseteq u\) for any upper bound \(u\) of \(L'\)
- For every subset \(L' \subseteq L\):
  - The greatest lower bound of \(L'\) is unique if at all exists
    » \(\sqcap L'\) (meet) \(a \sqcap b = \sqcap\{a, b\}\)
  - The lowest upper bound of \(L'\) is unique if at all exists
    » \(\sqcup L'\) (join) \(a \sqcup b = \sqcup\{a, b\}\)
Complete Lattices

◆ A poset \((L, \subseteq)\) is a complete lattice if every subset has least and upper bounds

◆ \(L = (L, \subseteq) = (L, \subseteq, \sqcup, \sqcap, \bot, \top)\)
  
  – \(\bot = \bigcup \emptyset = \bigcap L\)
  
  – \(\top = \bigcap L = \bigcup \emptyset\)

◆ Examples
  
  – Total orders \((N, \leq)\)
  
  – Powersets \((P(S), \subseteq)\)
  
  – Powersets \((P(S), \supseteq)\)
  
  – Constant propagation
  
  – Intervals
Complete Lattices

Lemma For every poset \((L, \sqsubseteq)\) the following conditions are equivalent

– \(L\) is a complete lattice
– Every subset of \(L\) has a least upper bound
– Every subset of \(L\) has a greatest lower bound
Cartesian Products

- A complete lattice
  \((L_1, \sqsubseteq_1) = (L_1, \sqsubseteq, \sqcup_1, \sqcap_1, \bot_1, T_1)\)

- A complete lattice
  \((L_2, \sqsubseteq_2) = (L_2, \sqsubseteq, \sqcup_2, \sqcap_2, \bot_2, T_2)\)

- Define a Poset \(L = (L_1 \times L_2, \sqsubseteq)\) where
  - \((x_1, x_2) \sqsubseteq (y_1, y_2)\) if
    - \(x_1 \sqsubseteq y_1\) and
    - \(x_2 \sqsubseteq y_2\)

- \(L\) is a complete lattice
Finite Maps

- A complete lattice
  \((L_1, \sqsubseteq_1) = (L_1, \sqsubseteq, \sqcup_1, \sqcap_1, \bot_1, \top_1)\)
- A finite set \(V\)
- Define a Poset \(L = (V \rightarrow L_1, \sqsubseteq)\) where
  - \(e_1 \sqsubseteq e_2\) if for all \(v \in V\)
    - \(e_1v \sqsubseteq e_2v\)
- \(L\) is a complete lattice
Chains

- A subset $Y \subseteq L$ in a poset $(L, \sqsubseteq)$ is a chain if every two elements in $Y$ are ordered
  - For all $l_1, l_2 \in Y$: $l_1 \sqsubseteq l_2$ or $l_2 \sqsubseteq l_1$

- An ascending chain is a sequence of values
  - $l_1 \sqsubseteq l_2 \sqsubseteq l_3 \sqsubseteq \ldots$

- A strictly ascending chain is a sequence of values
  - $l_1 \sqsubset l_2 \sqsubset l_3 \sqsubset \ldots$

- A descending chain is a sequence of values
  - $l_1 \triangleright eq l_2 \triangleright eq l_3 \triangleright eq \ldots$

- A strictly descending chain is a sequence of values
  - $l_1 \triangleright l_2 \triangleright l_3 \triangleright \ldots$

- $L$ has a finite height if every chain in $L$ is finite

- **Lemma** A poset $(L, \sqsubseteq)$ has finite height if and only if every strictly decreasing and strictly increasing chains are finite
Monotone Functions

- A poset \((L, \sqsubseteq)\)

- A function \(f: L \rightarrow L\) is monotone if for every \(l_1, l_2 \in L:\)
  \[ l_1 \sqsubseteq l_2 \Rightarrow f(l_1) \sqsubseteq f(l_2) \]
Fixed Points

- A monotone function $f: L \to L$ where $(L, \sqsubseteq, \sqcup, \sqcap, \bot, \top)$ is a complete lattice
- $\text{Fix}(f) = \{ l: l \in L, f(l) = l \}$
- $\text{Red}(f) = \{ l: l \in L, f(l) \sqsubseteq l \}$
- $\text{Ext}(f) = \{ l: l \in L, l \sqsubseteq f(l) \}$
  - $l_1 \sqsubseteq l_2 \Rightarrow f(l_1) \sqsubseteq f(l_2)$
- Tarski’s Theorem 1955: if $f$ is monotone then:
  - $\text{lfp}(f) = \bigwedge \text{Fix}(f) = \bigwedge \text{Red}(f) \in \text{Fix}(f)$
  - $\text{gfp}(f) = \bigvee \text{Fix}(f) = \bigvee \text{Ext}(f) \in \text{Fix}(f)$
Chaotic Iterations

- A lattice $L = (L, \sqsubseteq, \cup, \cap, \bot, \tau)$ with finite strictly increasing chains
- $L^n = L \times L \times \cdots \times L$
- A monotone function $f: L^n \rightarrow L^n$
- Compute $\text{lfp}(f)$
- The simultaneous least fixed of the system $\{x[i] = f_i(x): 1 \leq i \leq n\}$

\[
\begin{align*}
\text{x} &:= (\bot, \bot, \ldots, \bot) \\
\text{while } (f(x) \neq x) \text{ do} & \\
\text{x} &:= f(x) \\
\text{for } i := 1 \text{ to } n \text{ do} & \\
\text{x}[i] &:= \bot \\
\text{WL} &:= \{1, 2, \ldots, n\} \\
\text{while } (\text{WL} \neq \emptyset) \text{ do} & \\
\text{select and remove an element } i \in \text{WL} & \\
\text{new} &:= f_i(x) \\
\text{if } (\text{new} \neq \text{x}[i]) \text{ then} & \\
\text{x}[i] &:= \text{new}; \\
\text{Add all the indexes that directly depends on } i \text{ to WL}
\end{align*}
\]
Specialized Chaotic Iterations
System of Equations

\[ S = \begin{cases} 
  \text{df}_{\text{entry}}[s] = \tau \\
  \text{df}_{\text{entry}}[v] = \bigsqcup \{ f(u, v)(\text{df}_{\text{entry}}[u]) \mid (u, v) \in E \} 
\end{cases} \]

\[ F_S : L^n \rightarrow L^n \]

\[ F_S(X)[s] = \tau \]
\[ F_S(X)[v] = \bigsqcup \{ f(u, v)(X[u]) \mid (u, v) \in E \} \]

\[ \text{lfp}(S) = \text{lfp}(F_S) \]
Specialized Chaotic Iterations

Chaotic(G(V, E): Graph, s: Node, L: Lattice, τ: L, f: E \rightarrow (L \rightarrow L) ){
    for each v in V to n do df\_entry[v] := \bot
    df[s] = τ
    WL = \{s\}
    while (WL \neq \emptyset) do
        select and remove an element u \in WL
        for each v, such that. (u, v) \in E do
            temp = f(e)(df\_entry[u])
            new := df\_entry(v) \sqcup temp
            if (new \neq df\_entry[v]) then
                df\_entry[v] := new;
                WL := WL \cup \{v\}
            WL := WL \cup \{v\}
Constant Propagation

1. \( z := 3 \)
2. \( z <= 0 \)
3. \( z > 0 \)
4. \( x = 1 \)
5. \( x! = 1 \)
6. \( y = 7 \)
7. \( y := z + 4 \)

N | Value | WL
---|---|---
1 | \([x\rightarrow?, y\rightarrow?, z\rightarrow?]\) | \{2, 3, 4, 5, 6, 7\}
2 | \([x\rightarrow?, y\rightarrow?, z\rightarrow3]\) | \{3, 4, 5, 6, 7\}
3 | \([x\rightarrow?, y\rightarrow?, z\rightarrow3]\) | \{4, 5, 6, 7\}
4 | \([x\rightarrow1, y\rightarrow7, z\rightarrow3]\) | \{5, 6, 7\}
5 | \([x\rightarrow?, y\rightarrow7, z\rightarrow3]\) | \{6, 7\}
6 | \([x\rightarrow?, y\rightarrow7, z\rightarrow3]\) | \{7\}
Example Constant Propagation

\[
\begin{align*}
DF(1) &= [x \mapsto 0] \\
DF(2) &= DF(1)[x \mapsto 3] \sqcup DF(2) \\
DF(3) &= DF(2)
\end{align*}
\]

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>[x \mapsto 0]</td>
<td>[x \mapsto 3]</td>
<td>[x \mapsto 3]</td>
</tr>
<tr>
<td>[x \mapsto 0]</td>
<td>[x \mapsto ?]</td>
<td>[x \mapsto?]</td>
</tr>
<tr>
<td>[x \mapsto 7]</td>
<td>[x \mapsto 9]</td>
<td>[x \mapsto 7]</td>
</tr>
<tr>
<td>[x \mapsto ?]</td>
<td>[x \mapsto 3]</td>
<td>[x \mapsto 3]</td>
</tr>
</tbody>
</table>
Complexity of Chaotic Iterations

- Parameters:
  - \( n \) the number of CFG nodes
  - \( k \) is the maximum outdegree of edges
  - A lattice of height \( h \)
  - \( c \) is the maximum cost of
    - applying \( f(e) \)
    - \( \sqcup \)
    - \( L \) comparisons

- Complexity
  \[ O(n \times h \times c \times k) \]
Soundness

- Every detected constant is indeed such
- Every error will be detected
- The least fixed points represents all occurring runtime states
- Next week
\[ \forall a: f(\gamma(a)) \sqsubseteq \gamma(f^\#(a)) \]
Finite Height Case
Soundness Theorem (1)

1. Let \((\alpha, \gamma)\) form Galois connection from \(C\) to \(A\)
2. \(f: C \rightarrow C\) be a monotone function
3. \(f^\#: A \rightarrow A\) be a monotone function
4. \(\forall a \in A: f(\gamma(a)) \subseteq \gamma(f^#(a))\)

- \(\text{lfp}(f) \subseteq \gamma(\text{lfp}(f^#))\)
- \(\alpha(\text{lfp}(f)) \subseteq \text{lfp}(f^#)\)
Completeness

- Every constant is indeed detected as such
- Every detected error is real
- Cannot be guaranteed in general
The Join-Over-All-Paths (JOP)

- Let $\text{paths}(v)$ denote the potentially infinite set of paths from start to $v$ (written as sequences of labels).
- For a sequence of edges $[e_1, e_2, \ldots, e_n]$ define $f[e_1, e_2, \ldots, e_n] : L \rightarrow L$ by composing the effects of basic blocks,
  
  \[ f[e_1, e_2, \ldots, e_n](l) = f(e_n)(\ldots (f(e_2)(f(e_1)(l)) \ldots) \]

- $\text{JOP}[v] = \bigsqcup \{ f[e_1, e_2, \ldots, e_n](i)
  
  \mid [e_1, e_2, \ldots, e_n] \in \text{paths}(v) \}$
JOP vs. Least Solution

- The DF solution obtained by Chaotic iteration satisfies for every $l$:
  - $\text{JOP}[v] \subseteq \text{DF}_{\text{entry}}(v)$

- A function $f$ is additive (distributive) if
  - $f(\bigcup \{x \mid x \in X\}) = \bigcup \{f(x) \mid x \in X\}$

- If every $f_l$ is additive (distributive) for all the nodes $v$
  - $\text{JOP}[v] = \text{DF}_{\text{entry}}(v)$
Conclusions

- Chaotic iterations is a powerful technique
- Easy to implement
- Rather precise
- But expensive
  - More efficient methods exist for structured programs
- Abstract interpretation relates runtime semantics and static information
- The concrete semantics serves as a tool in designing abstractions
  - More intuition will be given in the sequel
Widening

- Accelerate the termination of Chaotic iterations by computing a more conservative solution
- Can handle lattices of infinite heights
Specialized Chaotic Iterations+

Chaotic(G(V, E): Graph, s: Node, L: lattice, τ: L, f: E →(L →L))
{
  for each v in V to n do df_entry[v] := ⊥
  In[v] = τ
  WL = {s}
  while (WL ≠ 0) do
    select and remove an element u ∈ WL
    for each v, such that. (u, v) ∈ E do
      temp = f(e)(df_entry[u])
      new := df_entry(v) ▽ temp
      if (new ≠ df_entry[v]) then
        df_entry[v] := new;
        WL := WL ∪ {v}
}
Example Interval Analysis

- Find a lower and an upper bound of the value of a variable
- Usages?
- Lattice
  \[ L = (\mathbb{Z} \cup \{-\infty, \infty\} \times \mathbb{Z} \cup \{-\infty, \infty\}, \sqsubseteq, \sqcup, \sqcap, \bot, \top) \]
- \([a, b] \sqsubseteq [c, d]\) if \(c \leq a\) and \(d \geq b\)
- \([a, b] \sqcup [c, d] = [\min(a, c), \max(b, d)]\)
- \([a, b] \sqcap [c, d] = [\max(a, c), \min(b, d)]\)
- \(\top = \)
- \(\bot = \)
Example Program

Interval Analysis

\[ x := 1; \]
while \[ x \leq 1000 \] do
\[ x := x + 1; \]

IntEntry(1) = [minint, maxint]
IntExit(1) = [1, 1]
IntEntry(2) = IntExit(1) \sqcup IntExit(3)
IntExit(2) = IntEntry(2)
IntEntry(3) = IntExit(2) \sqcap [minint, 1000]
IntExit(3) = IntEntry(3) + [1, 1]
IntEntry(4) = IntExit(2) \sqcap [1001, maxint]
IntExit(4) = IntEntry(4)
Widening for Interval Analysis

- $\bot \nabla [c, d] = [c, d]$
- $[a, b] \nabla [c, d] = [
  \begin{array}{l}
  \text{if } a \leq c \\
  \quad \text{then } a \\
  \quad \text{else } -\infty, \\
  \text{if } b \geq d \\
  \quad \text{then } b \\
  \quad \text{else } \infty
  \end{array} ]$
Example Program
Interval Analysis

[x := 1]¹ ;
while [x ≤ 1000]² do
[x := x + 1;]³

IntEntry(1) = [-∞, ∞]
IntExit(1) = [1,1]
IntEntry(2) = IntExit(2) ∨ (IntExit(1) ∪ IntExit(3))
IntExit(2) = IntEntry(2)

IntEntry(3) = IntExit(2) ∩ [-∞,1000]
IntExit(3) = IntEntry(3)+[1,1]

IntEntry(4) = IntExit(2) ∩ [1001, ∞]
IntExit(4) = IntEntry(4)
Requirements on Widening

- For all elements \( l_1 \sqcup l_2 \sqsubseteq l_1 \sqcap l_2 \)
- For all ascending chains
  \( l_0 \sqsubseteq l_1 \sqsubseteq l_2 \sqsubseteq \ldots \)
  the following sequence is finite
    - \( y_0 = l_0 \)
    - \( y_{i+1} = y_i \sqcap l_{i+1} \)
- For a monotonic function
  \( f: L \rightarrow L \)
  define
    - \( x_0 = \bot \)
    - \( x_{i+1} = x_i \sqcap f(x_i) \)
- Theorem:
  - There exits \( k \) such that \( x_{k+1} = x_k \)
  - \( x_k \in \text{Red}(f) = \{ l : l \in L, f(l) \sqsubseteq l \} \)
Narrowing

- Improve the result of widening
- $y \sqsubseteq x \Rightarrow y \sqsubseteq (x \bigtriangleup y) \sqsubseteq x$
- For all decreasing chains $x_0 \sqsubseteq x_1 \sqsubseteq \ldots$
  the following sequence is finite
  - $y_0 = x_0$
  - $y_{i+1} = y_i \bigtriangleup x_{i+1}$
- For a monotonic function $f: L \rightarrow L$ and $x \in \text{Red}(f) = \{l: l \in L, f(l) \sqsubseteq l\}$
  define
  - $y_0 = x$
  - $y_{i+1} = y_i \bigtriangleup f(y_i)$
- Theorem:
  - There exits $k$ such that $y_{k+1} = y_k$
  - $y_k \in \text{Red}(f) = \{l: l \in L, f(l) \sqsubseteq l\}$
Narrowing for Interval Analysis

- $[a, b] \triangle \bot = [a, b]$
- $[a, b] \triangle [c, d] = [
  \text{if } a = -\infty \text{ then } c \\
  \text{else } a, \\
  \text{if } b = \infty \text{ then } d \\
  \text{else } b 
]$
Example Program
Interval Analysis

[x := 1] ;
while [x ≤ 1000] do
   [x := x + 1;]

IntEntry(1) = [−∞, ∞]
IntExit(1) = [1,1]
IntEntry(2) = InExit(2) ∪ (IntExit(1) ∩ IntExit(3))
IntExit(2) = IntEntry(2)
IntEntry(3) = IntExit(2) ∩ [−∞,1000]
IntExit(3) = IntEntry(3)+[1,1]
IntEntry(4) = IntExit(2) ∩ [1001,∞]
IntExit(4) = IntEntry(4)
Non Montonicity of Widening

- \([0,1] \triangledown [0,2] = [0, \infty]\)
- \([0,2] \triangledown [0,2] = [0,2]\)
Widening and Narrowing
Summary

- Very simple but produces impressive precision
- Sometimes non-monotonic
- The McCarthy 91 function

```plaintext
int f(x) [-∞, ∞]
  if x > 100
    then [101, ∞] return x -10 [91, ∞-10];
  else [-∞, 100] return f(f(x+11)) [91, 91];
```

- Also useful in the finite case
- Can be used as a methodological tool
Numeric Abstract Domain Examples

- **signs**
  \[ x \geq 0 \]

- **intervals**
  \[ x \in [a, b] \]

- **octagons**
  \[ \pm x \pm y \leq c \]

- **polyhedra**
  \[ \Sigma a_i x_i \leq c \]
Pointer Language

\[ a ::= x \mid *x \mid &x \mid \ldots \]

\[ b ::= \text{true} \mid a = a \mid \text{not} \ b \]

```
assume b
```

```
x ::= a
```

```
*x ::= y
```
Collecting Semantics for Pointers

State 1 = \([\text{Loc} \rightarrow \text{Loc} \cup Z]\)
Points-To Analysis

- Lattice $L_{pt} =$
- Galois connection
- Meaning of statements
t := &a;
y := &b;
z := &c;

if x > 0
    then p := &y;
    else p := &z;

*p := t;
t := &a; /* {(t, a)} */
y := &b; /* {(t, a), (y, b)} */
z := &c; /* {(t, a), (y, b), (z, c)} */
if x > 0;
    then p := &y; /* {(t, a), (y, b), (z, c), (p, y)} */
else p := &z; /* {(t, a), (y, b), (z, c), (p, z)} */
*p := t;
/* {(t, a), (y, b), (y, c), (p, y), (p, z), (y, a), (z, a)} */
Abstract Transformers

State\# = \text{P}(\text{Var}^* \times \text{Var}^*)

\[
\begin{align*}
&[x := a ]# \\
&[x := \&y ]# \\
&[x := *y ]# \\
&[x := y ]# \\
&[*x := y ]# \\
&[\text{assume } x ==y ]# \\
&[\text{assume } x !=y ]#
\end{align*}
\]
t := &a; /* {(t, a)} */

y := &b; /* {(t, a), (y, b)} */

z := &c; /* {(t, a), (y, b), (z, c)} */

if x > 0;
    then p := &y; /* {(t, a), (y, b), (z, c), (p, y)} */
    else p := &z; /* {(t, a), (y, b), (z, c), (p, z)} */

*p := t;
/* {(t, a), (y, b), (y, c), (p, y), (p, z), (y, a), (z, a)} */
Flow insensitive points-to-analysis
Steengard 1996

- Ignore control flow
- One set of points-to per program
- Can be represented as a directed graph
- Conservative approximation
  - Accumulate pointers
- Can be computed in almost linear time
  - Union find
t := &a;
y := &b;
z := &c;

if x > 0;
    then p := &y;
    else p := &z;

*p := t;
Precision

- We cannot usually have $\alpha(\text{CS}) = \text{DF}$ on all programs

- But can we say something about precision in all programs?
The Join-Over-All-Paths (JOP)

- Let \( \text{paths}(v) \) denote the potentially infinite set of paths from start to \( v \) (written as sequences of edges).

- For a sequence of edges \([e_1, e_2, ..., e_n]\) define \( f^\#[e_1, e_2, ..., e_n] : L \to L \) by composing the effects of basic blocks:
  \[
f^\#[e_1, e_2, ..., e_n](l) = f^\#(e_n)(... (f^\#(e_2) (f^\#(e_1) (l))) ...)
\]

- \( \text{JOP}[v] = \bigsqcup \{ f^\#[e_1, e_2, ..., e_n](i) \mid [e_1, e_2, ..., e_n] \in \text{paths}(v) \} \)
JOP vs. Least Solution

- The df solution obtained by Chaotic iteration satisfies for every v:
  - JOP[v] ⊆ df[v]

- A function f# is additive (distributive) if
  - f#(⊔{x | x ∈ X}) = ⊔{f#(x) | ∈ X}

- If every f#_(u,v) is additive (distributive) for all the edges (u,v)
  - JOP[v] = df[v]

- Examples
  - Intervals
  - Points-to
Notions of precision

- $CS = \gamma (df)$
- $\alpha(CS) = df$
- Meet(Join) over all paths
- Using best transformers
- Good enough
Complexity of Chaotic Iterations

- Usually depends on the height of the lattice
- In some cases better bound exist
- A function $f$ is fast if $f(f(l)) \subseteq l \sqcup f(l)$
- For fast functions the Chaotic iterations can be implemented in $O(nest \times |V|)$ iterations
  - $nest$ is the number of nested loop
  - $|V|$ is the number of control flow nodes
Challenges

A problem has been detected and Windows has been shut down to prevent damage to your computer.

IRQL_NOT_LESS_OR_EQUAL

If this is the first time you have seen this stop error screen, restart your computer. If this screen appears again, follow these steps:

Check to be sure any new hardware or software is properly installed. If this is a new installation, ask your hardware or software manufacturer for any Windows updates you might need. If problems continue, disable or remove any newly installed hardware or software. Disable BIOS memory options such as caching or shadowing.

If you need to use Safe Mode to remove or disable components, restart your computer, press F8 to select Advanced Startup Options, and then select Safe Mode.

Technical information:

*** STOP: 0x0000000A 0xFFFFFA802880010A,
0x0000000000000002, 0x0000000000000000, 0xFFFFFA8000185E251)
“Things like even software verification, this has been the Holy Grail of computer science for many decades but now in some very key areas, for example, driver verification we’re building tools that can do actual proof about the software and how it works in order to guarantee the reliability” Bill Gates
Success Stories Abstract
Interpretation

- SLAM: Microsoft Device Driver Verification
- The Astrée Static Analyzer
- Panaya Change Impact Analysis
Conclusion

- Static analysis is powerful technique
- But expensive
  - More efficient methods exist for structured programs
- Abstract interpretation relates runtime semantics and static information
- The concrete semantics serves as a tool in designing abstractions
Conclusions

- Chaotic iterations is a powerful technique
- Easy to implement
- Rather precise
- But expensive
  - More efficient methods exist for structured programs