Pattern-based Verification of Concurrent Programs

Tomáš Poch

Pierre Ganty, IMDEA SW

CHARLES UNIVERSITY IN PRAGUE
Faculty of Mathematics and Physics
Based on the POPL 2011 talk

Pattern-based Verification of Concurrent Programs

Pierre Ganty    Javier Esparza

POPL practice talk
IMDEA theory lunch
Reachability Analysis of Concurrent Programs

Instance:

<table>
<thead>
<tr>
<th>Thread $t_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Locals</td>
</tr>
<tr>
<td>Procedures</td>
</tr>
</tbody>
</table>

| Th. $t_2$ |

Shared Memory

Question: Is label error reachable?

Fundamental problem in software verification
Reachability Analysis of Concurrent Programs

Instance:

<table>
<thead>
<tr>
<th>Thread $t_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x, y, z, \ldots$</td>
</tr>
<tr>
<td>Procedures</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Th. $t_2$</th>
</tr>
</thead>
</table>

| Shared Memory |

Question: Is label **error** reachable?

Fundamental problem in software verification
Reachability Analysis of Concurrent Programs

Instance:

<table>
<thead>
<tr>
<th>Thread $t_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x, y, z, \ldots$</td>
</tr>
<tr>
<td>.</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Th. $t_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Shared Memory

Question: Is label error reachable?

Fundamental problem in software verification
Reachability Analysis of Concurrent Programs

Instance:

Thread $t_1$

$x, y, z, \ldots$

\[ \vdots \]

\[ g \]

Th. $t_2$

Question: Is label error reachable?

Fundamental problem in software verification
Reachability Analysis of Concurrent Programs

Instance:

<table>
<thead>
<tr>
<th>Thread $t_1$</th>
<th>Th. $t_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x, y, z, \ldots$</td>
<td></td>
</tr>
<tr>
<td>\text{unbounded control} $\in \Gamma^*$</td>
<td>\text{unbounded data} $\in \mathbb{Z}$</td>
</tr>
</tbody>
</table>

$g$

Question: Is label \textcolor{red}{\text{error}} reachable?

\textcolor{red}{\text{Fundamental problem in software verification}}
Reachability Analysis of Concurrent Programs

Instance:

<table>
<thead>
<tr>
<th>Thread $t_1$</th>
<th>Th. $t_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x, y, z, \ldots$</td>
<td></td>
</tr>
</tbody>
</table>

Undecidable

unbounded control $\in \Gamma^*$

unbounded data $\in \mathbb{Z}$

$g$

Question: Is label error reachable?

Fundamental problem in software verification
Reachability Analysis of Concurrent Programs

Instance:

Thread $t_1$

$x, y, z, \ldots$

Th. $t_2$

bounded data $\in \mathbb{B}$

bounded data $\in \Sigma$

unbounded control $\in \Gamma^*$

Question: Is label error reachable?

Fundamental problem in software verification
Reachability Analysis of Concurrent Programs

Instance:

- Thread $t_1$
  - $x, y, z, \ldots$
- Th. $t_2$
  - Bounded data $\in \mathbb{B}$
  - Bounded data $\in \Sigma$

Question: Is label error reachable?

Fundamental problem in software verification
Context-Switches

Th. $t_1$  Th. $t_2$

Shared Memory
Context-Switches

Th. $t_1$

Th. $t_2$

$\sigma$
Context-Switches

Th. $t_1$

\[ \sigma \]

Th. $t_2$

\[ \sigma \]
Context-Switches

\[ \sigma' \]

\[ \sigma \]
Context-Switches

\[ \text{Th. } t_1 \quad \sigma' \quad \text{Th. } t_2 \]

\[ \sigma \]
Context-Switches

\[ \sigma \to \sigma' \]
Context-Switches

\[ \text{Th. } t_1 \]

\[ \text{Th. } t_2 \]

\[ \sigma' \]

\[ \sigma' \]
Context-Switches

\[ \text{Th. } t_1 \quad \text{Th. } t_2 \]

\[ \sigma' \quad \sigma'' \]

\[ \sigma' \sigma'' \]
Context-Switches
Context-Switches

\[ \begin{array}{ccc}
\text{Th. } t_1 & \sigma & \sigma' & \sigma'' & \ldots \\
\text{Th. } t_2 & \sigma''
\end{array} \]
Context-Switches

\[
\begin{array}{c|c}
\text{Th. } t_1 & \text{Th. } t_2 \\
\hline
\text{hourglass} & \sigma'' \\
\multicolumn{2}{c}{\sigma \sigma' \sigma'' \ldots}
\end{array}
\]

\# of writes to the SM = size of the tape

2005*: reachability for tape word no longer than \( k \)

2010†: reachability for tape word in regular expression

* Shaz Qadeer, Jakob Rehof in TACAS ’05
† Pierre Ganty, Rupak Majumdar, Benjamin Monmege in CAV ’10
Context-Switches

\[
\begin{array}{c|c}
\text{Th. } t_1 & \text{Th. } t_2 \\
\begin{array}{c}
\text{hourglass}
\end{array} & \begin{array}{c}
\sigma''
\end{array}
\end{array}
\]

\[
\begin{array}{c|c|c|c}
\sigma & \sigma' & \sigma'' & \ldots
\end{array}
\]

\# of writes to the SM = size of the tape

2005*: reachability for tape word no longer than $k$

2010‡: reachability for tape word in $w_1^*...w_n^*$, $w_i \in \Sigma^*$

* Shaz Qadeer, Jakob Rehof in TACAS ’05
‡ Pierre Ganty, Rupak Majumdar, Benjamin Monmege in CAV ’10
Pattern-based Analysis

\[
\begin{align*}
\{ & 1 \text{ write to the SM} \\
& 1 \text{ context-switch} \\
& 1 \text{ communication} \} = 1 \text{ tape letter} \\
\end{align*}
\]

- Context Bounding Analysis: tape word in \( \sum \cdots \sum \) up to \( k \) copies

- Pattern-based Analysis: tape word in \( w_1^* \cdots w_n^*, w_i \in \sum^* \)

Instance: A multithreaded prg + pattern

Question: Is label error reachable following the pattern?
Pattern-based Analysis: Example

Thread $t_1$

L1: bit = F;
if bit == T
  goto error;
else
  goto L1;

error : print "busted";

Thread $t_2$

L: bit = T;
goto L;
Pattern-based Analysis: Example

Thread \( t_1 \)

- \( L_1: \) bit = F;
- if bit == T
  - goto error;
- else
  - goto \( L_1 \);
- error:
  - print "busted";

Thread \( t_2 \)

- \( L: \) bit = T;
- goto \( L \);

SM: bit
Pattern-based Analysis: Example

Thread $t_1$

L1: bit = F;
    if bit == T
    goto error;
    else
    goto L1;

error: print "busted";

Thread $t_2$

L: bit = T;
  goto L;

Is error reachable for pattern $(\text{bit} = \text{F} \cdot \text{bit} = \text{T})^*$?
Pattern-based Analysis: Example

Thread $t_1$

L1:
bit = F;
if bit == T
goto error;
else
goto L1;

error:
print "busted";

Thread $t_2$

L:
bit = T
goto T

Is error reachable for pattern $(bit = F \cdot bit = T)^*$?
Pattern-based Analysis: Example

Thread $t_1$

L1: bit = F;
    if bit == T
        goto error;
    else
        goto L1;
    print "busted";

error :

Thread $t_2$

L: bit
  goto

SM: bit

Is error reachable for pattern $(\text{bit} = \text{F} \cdot \text{bit} = \text{T})^*$?
Pattern-based Analysis: Example

Thread $t_1$

L1: bit = F
    if bit = T
        goto error;
    else
        goto L1;

error: print "busted";

Thread $t_2$

L: bit = T;
goto L;

SM: bit

bit = F

Is error reachable for pattern $(bit = F \cdot bit = T)^*$?
Pattern-based Analysis: Example

Thread $t_1$

L1: bit = F
    if bit = T
    goto error;
    else
    goto L1;

error: print "busted";

SM: bit

Thread $t_2$

L: bit = T;
    goto L;

Is error reachable for pattern $(\text{bit} = \text{F} \cdot \text{bit} = \text{T})^*$?
Pattern-based Analysis: Example

Thread $t_1$

L1: bit = F;
    if bit == T
        goto error;
    else
        goto L1;

error: print "busted";

Thread $t_2$

L: bit = T
    goto error

SM: bit

bit = F  bit = T

Is error reachable for pattern (bit = F \cdot bit = T)^*?
Pattern-based Analysis: Example

Thread $t_1$

L1: bit = F;
    if bit == T
       goto error;
    else
       goto L1;
    print "busted";

error :

Thread $t_2$

L: bit = T

Is error reachable for pattern $(\text{bit} = F \cdot \text{bit} = T)^*$?
Pattern-based Analysis: Example

Thread $t_1$

L1: \( \text{bit} = F; \)
    \( \text{if bit} == T \)
    \( \text{goto error;} \)
    else
    \( \text{goto L1;} \)
    \( \text{print "busted";} \)

Thread $t_2$

L: \( \text{bit} = T \)
    \( \text{got} \)

\[ \text{error} \] : 

\[ \text{SM: bit} \]

\[ \text{bit} = F, \text{bit} = T \]

Is error reachable for pattern \((\text{bit} = F \cdot \text{bit} = T)^*\)?
Pattern-based Analysis: Example

And what about \((\text{bit}=\text{F})^{*}(\text{bit}=\text{T})^{*}\) - yes

\((\text{bit}=\text{F})^{*}\) - no
Pattern-based Analysis: Example

Thread $t_1$

L1: bit = F;
if bit == T
  goto error;
else
  goto L1;

print "busted";

Thread $t_2$

L: bit = T
  goto error;

Error in two context switches

→ context bounding will discover it
Consider thread T1 counting the number of times it reads T

- Producer/consumer style
- Context bounding won’t help
Language perspective

- $T_i$ represented by context-free grammar $G_i$
  - Non-terminals encode program positions and local variables
  - Terminals encode communication
    - Rendez-vous style
    - Reads/writes to the shared memory

- Derivation of $G_i \leftrightarrow$ Computation of $T_i$
- $w \in L(G_i)$ - communication history performed by $T_i$ going from start to its final position
- $w \in L(G_1) \cap L(G_2)$ – history allowed by both threads

Emptiness of $L(G_1) \cap L(G_2)$ is non-decidable
Rendez-vous from lang. perspective

```c
void main()
0 int x = 5;
1 while (x>0) {
2    b();
3    x--;
4 }
5
void b(){
6    sync(msg);
7}
```

Init -> [x'=5,pc=1]
[x>0,pc=1] -> [x'=x,pc=2]
[x=0,pc=1] -> [x'=0,pc=5]
[x,pc=2] -> [pc=6][x'=x,pc=3]
[x,pc=3] -> [x'=x-1,pc=1]
[x,pc=5] -> ε

[pc=6] -> <msg>

Process synchronizes 5x (msg signal). Then terminates.

Init \(\rightarrow^*\) <msg><msg><msg><msg><msg><msg>

The process will cooperate a process that can produce 5 messages as well.
Shared memory from lang. prespect.

- Grammar follows computation as in the rendez-vous example – no terminals
- Additional rules model context switch at NT
  - For each NT introduce “nonactive” counterpart
  - Terminals for each context switch
    - yield(memstate) – go to inactive state
    - go(TID) – thread TID gets active
Context-Switches

\[ \text{Th. } t_1 \]

\[ \text{Th. } t_2 \]

\[ \sigma'' \]

\[ \# \text{ of writes to the SM} = \text{size of the tape} \]

2005*: reachability for tape word no longer than \( k \)

2010†: reachability for tape word in regular expression

* Shaz Qadeer, Jakob Rehof in TACAS ’05

† Pierre Ganty, Rupak Majumdar, Benjamin Monmege in CAV ’10
Context switching as language

$G_1$ derivation

$T_1$ is active
Stack changes
No terminals produced
Context switching as language

$G_1$ derivation

$T_1$ is active
Stack changes
No terminals produced
Context switching as language

$G_1$ derivation

$T_1$ is active
Stack changes
No terminals produced

$T_1$ inactive
Other threads can go,
and write whatever value
Context switching as language

$G_1$ derivation

- $T_1$ is active
  - Stack changes
  - No terminals produced

- $T_1$ inactive
  - Other threads can go, and write whatever value

- $T_1$ is active again
Context switching as language

$G_1$ derivation

$T_1$ is active
Stack changes
No terminals produced

$T_1$ is active again

$(<\text{go}(2)>+<\text{go}(3)>+...+<\text{go}(\text{TIDmax})>;
<y\text{ield}(g_1)>+<\text{yield}(g_2)>+...+<\text{yield}(g_{\text{Dom}})>)*$

$<\text{go}(1)>$

$T_1$ inactive
Other threads can go,
and write whatever value
Context switching as language

G₁ derivation

T₁ is active
Stack changes
No terminals produced

T₁ inactive
Other threads can go, and write whatever value

T₁ is active again

G₂ derivation

T₂ inactive

T₂ is active

T₂ inactive
Context switching as language

\[ w \in L(G_1) \cap L(G_2) \] – history of global memory admissible by both threads

Emptiness of \( L(G_1) \cap L(G_2) \) still non-decidable
Computation as language

<table>
<thead>
<tr>
<th>bit=F</th>
<th>bit=T</th>
</tr>
</thead>
</table>

0  bit=F;

1  if bit==T

2  goto 4;

3  return;

4  print “busted”
Inactive states, communication

TID=0
Inactive states, communication

\[ \text{TID}=0 \]

<table>
<thead>
<tr>
<th>bit=F</th>
<th>bit=T</th>
<th>Inactive bit=F</th>
<th>Inactive bit=T</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Circle" /></td>
<td><img src="image2" alt="Circle" /></td>
<td><img src="image3" alt="Circle" /></td>
<td><img src="image4" alt="Circle" /></td>
</tr>
</tbody>
</table>
Inactive states, communication

TID=0

<table>
<thead>
<tr>
<th>bit=F</th>
<th>bit=T</th>
<th>Inactive bit=F</th>
<th>Inactive bit=T</th>
</tr>
</thead>
</table>

<yield(bit=F)>
Inactive states, communication

TID=0

bit=F  

<yield(bit=F)>

bit=T  

<go(TID=0)>

Inactive bit=F

Inactive bit=T
Inactive states, communication

TID=0

bit=F  bit=T  Inactive bit=F  Inactive bit=T

<yield(bit=F)>  <go(TID=1)>  <go(TID=0)>
Inactive states, communication

TID=0

bit=F

<yield(bit=F)>

bit=T

<yield(bit=T)>

Inactive
bit=F

<go(TID=1)>

Inactive
bit=T

<go(TID=0)>
Inactive states, communication

**TID=0**

- **bit=F**
  - `<yield(bit=F)>`
- **bit=T**
  - `<yield(bit=T)>`
- Inactive **bit=F**
  - `<go(TID=1)>`
  - `<yield(bit=F)>`
  - `<go(TID=0)>`
- Inactive **bit=T**
  - `<go(TID=1)>`
  - `<yield(bit=F)>`
  - `<go(TID=0)>`
Inactive states, communication

TID=0

bit=F
bit=T

Inactive bit=F
Inactive bit=T

<yield(bit=...)>  
<go(TID=...)>
TID = 0

0  bit=F;

1  if bit==T

2  goto 4;

3  return;

4  print “busted”
TID = 0

0  bit=F;

1  if bit==T

2  goto 4;

3  return;

4  print “busted”
Decision procedure

• Reachability in concurrent program transformed to a language problem
  – Intersection of context-free languages
    • Emptiness of $L(G_1) \cap L(G_2)$ non-decidable

• Context bounded verification
  – At most $k$ context switches
    • Emptiness of $L(G_1) \cap L(G_2) \cap \{\text{go(TID), yield(gmem)}\}^{2k}$

• Pattern based verification
  – Context switches follow the pattern
    • Emptiness of $L(G_1) \cap w_1^* ... w_n^* \cap L(G_2)$
    • $w_i \in \{\text{go(TID), yield(gmem)}\}^*$
    • Example – at most $k$ ctx sw : $(\text{go}(0)^* \ \text{go}(1)^* \ \text{yield(true)}^* \ \text{yield(false)}^*)^K$
Decision Procedure

$L(G_1) \cap w_1^* \ldots w_n^* \cap L(G_2) = \emptyset$
Decision procedure

- Counting of w matters - \( w_1^i \ w_2^j \ldots \ w_n^k \in L(G) \)

- Modify G to G’
  - Accept only words from pattern
    - Intersection of CFG with the regular grammar
  - Produce single terminal \( a_p \) instead of the word \( w_p \)
    - Homomorphism

- \( w_1^i \ldots w_n^k \in L(G) \iff a_1^i \ldots a_n^k \in L(G’) \)
  - \( a_p \) are distinct
  - \( a_p \) fit on their position by construction of G’
Parikh image

- Fixed linear order over alphabet
  - \( \Sigma = \{ \alpha_1, \alpha_2 \ldots \alpha_p \} \)

- Parikh image of \( w \in \Sigma^* \) is a \( p \)-dimensional vector
  - i-th part is the number of occurrences of i-th symbol in \( w \)
  - \( \Pi(w) = \langle i_1, i_2, \ldots, i_p \rangle \), \( \Pi(\alpha_1\alpha_1\alpha_1\alpha_2) = \langle 3, 1, 0, \ldots, 0 \rangle \)

- Parikh image of language \( L \subseteq \Sigma^* \)
  - set of Parikh images of words from \( L \)
  - \( \Pi(L) = \{ \pi, \exists w \in L \Pi(w) = \pi \} \)
Decision procedure

- $w_1^i \ldots w_n^k \in L(G) \iff a_1^i \ldots a_n^k \in L(G')$
  - $a_p$ are distinct, fit on their position by construction of $G'$
  - $\pi \in \Pi(G') \iff a_1^{\pi(1)} \ldots a_n^{\pi(k)} \in L(G')$

- Parikh image of a context free language can be characterized by a formula in the Pressburger arithmetics (no multiplication)
  - $\Psi_{G'}(\pi) = \text{True} \iff \pi \in \Pi(G')$

- Pressburger arithmetic is decidable
From Language to Formula

\[
L(G_{T1}) \cap w_1^i w_2^j \ldots w_n^k \cap L(G_{T2}) = \emptyset
\]
\[\iff\]
\[
L(G_{T1}') \cap L(G_{T2}') = \emptyset
\]
\[\iff\]
\[
\Pi(G_{T1}') \cap \Pi(G_{T2}') = \emptyset
\]
\[\iff\]
\[
\Psi_{T1}' \& \Psi_{T2}' \text{ is unsatisfiable}
\]
• Petri-net intuition
  – net is counting the symbols
  – does not capture the order of symbols
• Structure
  – Place for each terminal and non-terminal
  – Transition for each rule
  – One token to the initial non-terminal

\[ X \rightarrow aXb \]
PN example

\[ X \rightarrow aXb \]
\[ X \rightarrow \varepsilon \]
PN example

\[ X \rightarrow aXb \]

\[ X \rightarrow \varepsilon \]
PN example

X -> aXb
X -> ε
PN example

\[ X \rightarrow aXb \]

\[ X \rightarrow \varepsilon \]
PN example

\[ X \rightarrow aXb \]
\[ X \rightarrow \varepsilon \]
PN example

\[ X \rightarrow aXb \]

\[ X \rightarrow \varepsilon \]
PN example

$\mathbf{X} \rightarrow \mathbf{aXb}$

$\mathbf{X} \rightarrow \mathbf{\varepsilon}$

$L(X) = a^i b^i$

$\Pi(X) = [i, i]$

Configurations corresponding $w \in L(X)$

→ all tokens in terminal places
PN examples

\[ X \rightarrow aXb \]
\[ X \rightarrow \varepsilon \]
\[ X \rightarrow abX \]
\[ X \rightarrow \varepsilon \]
\[ X \rightarrow Xba \]
\[ X \rightarrow \varepsilon \]

Configurations corresponding \( w \in L(X) \)

\( \Pi(X) = [i,i] \)

\( \rightarrow \) all tokens in terminal places
• Petri net is communication-free
  – Each transition has one input place
  – Context-free grammar (one NT on left-hand side)
• Set of admissible configurations of CF-PN can be characterized by Pressburger formula
Formula

- Formula based on
  - Kirchhoff-like rules
    - For each place "# of tokens" = "# of input transition applications" - "# of output transition applications"
  - Reachability rules
    - Each applied transition is reachable from the initial place

- Variables
  - For each place A, $x_A$ is number of tokens in the place, $z_A$ distance from initial place
  - For each transition $y_i$ is the number of applications

\[ x_A = y_1 + y_3 - y_2 - y_4 \]

Verma, Seidl, Schwentick: On the Complexity of Equational Horn Clauses 2005
Tool implementation

• Input
  – Control-flow grammar for each thread
  – Definition of global and local variables

• Goal
  – Transform grammars into formula, run solver
Input

• Ctrlflow grammar
  – Non-terminals – program locations
  – Terminals – manipulation with memory
  – Rules – context free
Input – Ctrlflow grammar

cfGrmRules(R):-
R = [ %add
    rule(add0,inc0,add1),
    LHS RHS
    add0 -> inc0 add1]
Input – Ctrlflow grammar

cfGrmRules(R):-
    R = [
        %add
        rule(add0,inc0,add1),
        rule(add1,term(lvar==true),add2),rule(add1,term(not(lvar==false)),add3),
        rule(add2,term(gvar=false),add3),
        rule(add3,dec0,add4),
        rule(add4,eps),
    ]
**Input**

- **Ctrlflow grammar**
  - Context free rules
  - Non-terminals – program locations
  - Terminals – manipulation with memory

- **Memory definition used to generate memory grammar**
  - Regular rules
  - Non-terminals – content of variables
  - Terminals – memory modifications, queries
bool lvar

Rules expressed symbolically

\[ \text{rule} \left( s(g(C), l(0)), \text{term}(c \neq \text{VAL}), s(g(C), l(0)) \right) : -C \neq \text{VAL} \]
Input

• Ctrlflow grammar
  – Context free rules
  – Non-terminals – program locations
  – Terminals – manipulation with memory

• Memory definition used to generate memory grammar
  – Regular rules
  – Non-terminals – content of variables
  – Terminals – memory modifications, queries

• Pattern expression (yield/go)
  – Transformed into regular grammar
Transformation chain

- **Mem definition** → **Memory grammar** → **Ctx switching rules** → **Pattern grammar**
- **Ctrlflow grammar** → **Intersect.** → **Add rules** → **Restrict** → **Intersect.**
  - Keep only yield/go
  - Word $w_p \to$ symbol $a_p$
- **Petri-net transformation** → **Instantiated grammar** → **Instantiation** → **Symbolic grammar** → **Formula**
Transformation chain

- Mem definition
- Pattern
- $T_1$ Ctrlflow
- Chain
- $T_1$ formula
- $T_2$ Ctrlflow
- Chain
- $T_2$ formula
- $T_n$ Ctrlflow
- Chain
- $T_n$ formula

Formula

Yices

Satisfiable $\rightarrow$ point is reachable

Trace given by solution
Validation

• Windows NT bluetooth driver example
  – Several variants, race conditions reported in [1]
  – We can detect them all, given the proper pattern

• Still toy example
  – On larger examples the instantiation phase takes long

[1] Suwimonteerabuth, Esparza, Schwoon: Symbolic Context-Bounded Analysis of Multithreaded Java Programs, SPIN '08
What the POPL paper is about

Pattern-based Analysis: Time Complexity

**Theorem:** pattern-based reachability is decidable in:

- time **polynomial** in $|\text{proc}|$
- time **exponential** in $\begin{cases} \#\text{proc}/\text{th} \\ |\text{pat}| \\ \#\text{th} \end{cases}$

- Programs often have few threads
- Pattern should be short
- Threads often have no procedure at all
- Programs are arbitrarily long
Conclusion

• Theory works
  – Bluetooth example is small, but real

• Tool runs
  – Lot of technical details solved

• Ongoing work
  – Improve instantiation phase
    • Abstraction
    • Skip it at all
  – Get the trace (Interpret the formula solution)
  – More, larger, examples
Thank you