Verification
Verification

• We use languages to describe the implementation and the specification of a system.

• We reduce the verification problem to language inclusion between implementation and specification.
• Configuration: triple \([l, n_x, n_y]\) where
  • \(l\) is the current value of the program counter, and
  • \(n_x, n_y\) are the current values of \(x, y\)

Examples: \([1,1,1], [5,0,1]\)

• Initial configuration: configuration with \(l = 1\)

• Potential execution: finite or infinite sequence of configurations

Examples: \([1,1,1][4,1,0]\)
            \([2,1,0][5,1,0]\)
            \([1,1,0][2,1,0][4,1,0][1,1,0]\)
• **Execution**: potential execution starting at an initial configuration, and where configurations are followed by their „legal successors“ according to the program semantics.

  Examples: \([1,1,1][2,1,1][3,1,1][4,0,1][1,0,1][5,0,1][1,1,0][2,1,0][4,1,0][1,1,0]\)

• **Full execution**: execution that cannot be extended (either infinite or ending at a configuration without successors)

```plaintext
1   while x = 1 do
2      if y = 1 then
3          x ← 0
4      y ← 1 − x
5     end
```
Verification as a language problem

• Implementation: set $E$ of executions
• Specification:
  – subset $P$ of the potential executions that satisfy a property, or
  – subset $V$ of the potential executions that violate a property
• Implementation satisfies specification if:
  – $E \subseteq P$, or
  – $E \cap V = \emptyset$.
• If $E$ and $P$ regular: inclusion checkable with automata
• If $E$ and $V$ regular: disjointness checkable with automata
Verification as a language problem

- Implementation: set $E$ of executions
- Specification:
  - subset $P$ of the potential executions that satisfy a property, or
  - subset $V$ of the potential executions that violate a property
- Implementation satisfies specification if:
  - $E \subseteq P$, or
  - $E \cap V = \emptyset$.
- If $E$ and $P$ regular: inclusion checkable with automata
- If $E$ and $V$ regular: disjointness checkable with automata

- How often is the case that $E, P, V$ are regular?
System NFA

1 while $x = 1$ do
2     if $y = 1$ then
3         $x \leftarrow 0$
4     $y \leftarrow 1 - x$
5 end
System NFA
System NFA
Property NFA

• Is there a full execution such that
  – initially $y = 1$,
  – finally $y = 0$, and
  – $y$ never increases?

• Set of potential executions for this property:
  $$[l, x, 1][l, x, 1]^* [l, x, 0]^* [5, x, 0]$$

• Automaton for this set:
Intersection of the system and property NFAs

- Automaton is empty, and so no execution satisfies the property
Another property

• Is the assignment \( y \leftarrow x - 1 \) redundant?
• Potential executions where the assignment is executed at least once and it changes the value of \( y \):
  \[
  [l, x, y]^* ([4, x, 0][1, x, 1] + [4, x, 1][1, x, 0]) [l, x, y]^*
  \]
• Therefore: assignment redundant iff none of these potential executions is a real execution of the program.
Networks of automata

1 while $x = 1$ do
2     if $y = 1$ then
3         $x \leftarrow 0$
4     $y \leftarrow 1 - x$
5 end

$x = 0 \Rightarrow y \leftarrow 1,$
$x \leftarrow 0,$
$x = 1 \Rightarrow y \leftarrow 0,$
$x \neq 1$

$x = 1$

$x = 1 \Rightarrow y \leftarrow 0,$
$y \neq 1$

$x = 0 \Rightarrow y \leftarrow 1,$
$y = 1$

$x = 0 \Rightarrow y \leftarrow 1$

$x = 1 \Rightarrow y \leftarrow 0$
Networks of automata

\[ A_2 \]
4, 5, 6, 7

\[ A_1 \]
2, 3, 6, 7

\[ A_0 \]
1, 3, 5, 7

\[ \text{inc}_2 \]
0, 1, 2, 3

\[ \text{inc}_2 \]
0, 1, 4, 5

\[ \text{inc}_2 \]
0, 2, 4, 6
• Tuple $\mathcal{A} = \langle A_1, \ldots, A_n \rangle$ of NFAs.
• Each NFA has its own alphabet $\Sigma_i$ of actions.
• Alphabets usually not disjoint!
• $A_i$ participates in action $a$ if $a \in \Sigma_i$.
• A configuration is a tuple $\langle q_1, \ldots, q_n \rangle$ of states, one for each automaton of the network.
• $\langle q_1, \ldots, q_n \rangle$ enables $a$ if every participant in $a$ is in a state from which an $a$-transition is possible.
• Enabled actions can occur, and their occurrence simultaneously changes the states of their participants. Non-participants stay idle and don‘t change their states.
Configuration graph of the network
Asynchronous product

AsyncProduct($A_1, \ldots, A_n$)

**Input:** a network of automata $\mathcal{A} = \langle A_1, \ldots, A_n \rangle$, where $A_i = (Q_i, \Sigma_i, \delta_i, Q_{0i}, F_i)$ for every $i = 1, \ldots, n$.

**Output:** NFA $A_1 \otimes \cdots \otimes A_n = (Q, \Sigma, \delta, Q_0, F)$ recognizing $L(\mathcal{A})$.

1. $Q, \delta, F \leftarrow \emptyset$
2. $Q_0 \leftarrow Q_{01} \times \cdots \times Q_{0n}$
3. $W \leftarrow Q_0$
4. while $W \neq \emptyset$ do
5.  pick $[q_1, \ldots, q_n]$ from $W$
6.  add $[q_1, \ldots, q_n]$ to $Q$
7.  if $\bigwedge_{i=1}^n q_i \in F_i$ then add $[q_1, \ldots, q_n]$ to $F$
8.  for all $a \in \Sigma_1 \cup \ldots \cup \Sigma_n$ do
9.      for all $i \in [1..n]$ do
10.     if $a \in \Sigma_i$ then $Q'_i \leftarrow \delta_i(q_i, a)$ else $Q'_i = \{q_i\}$
11.    for all $[q'_1, \ldots, q'_n] \in Q'_1 \times \cdots \times Q'_n$ do
12.       if $[q'_1, \ldots, q'_n] \notin Q$ then add $[q'_1, \ldots, q'_n]$ to $W$
13.       add $([q_1, \ldots, q_n], a, [q'_1, \ldots, q'_n])$ to $\delta$
14.  return $(Q, \Sigma, \delta, Q_0, F)$
Concurrent programs as networks of automata: Lamport‘s 1-bit algorithm (JACM86)

Shared variables: \( b[0], ..., b[n-1] \in \{0,1\} \), initially 0
Process \( i \in \{0, ..., n-1\} \)

```
repeat forever
  noncritical section
  T:  b[i]:=1
  for \( j \in \{0, ..., i-1\} \)
    if \( b[j]=1 \) then    b[i]:=0
      await \( \neg b[j] \)
      goto T
  for \( j \in \{i+1, ..., n-1\} \)  await \( \neg b[j] \)
  critical section
  b[i]:=0
```
Network for the two-process case
Asynchronous product
Checking properties of the algorithm

- **Deadlock freedom**: every configuration has at least one successor.
- **Mutual exclusion**: no configuration of the form \([b_0, b_1, c_0, c_1]\) is reachable.
- **Bounded overtaking (for process 0)**: after process 0 signals interest in accessing the critical section, process 1 can enter the critical section at most once before process 0 enters.
  - Let \(NC_i, T_i, C_i\) be the configurations in which process i is non-critical, trying, or critical.
  - Set of potential executions violating the property:
    \[\Sigma^* T_0 (\Sigma \setminus C_0)^* C_1 (\Sigma \setminus C_0)^* NC_1 (\Sigma \setminus C_0)^* C_1 \Sigma^*\]
The state-explosion problem

• In sequential programs, the number of reachable configurations grows exponentially in the number of variables.

• **Proposition**: The following problem is PSPACE-complete.
  
  – **Given**: a boolean program $\pi$ (program with only boolean variables), and a NFA $A_V$ recognizing a set of potential executions
  
  – **Decide**: Is $E_\pi \cap L(A_V)$ empty?
The state-explosion problem

• In concurrent programs, the number of reachable configurations also grows exponentially in the number of components.

• **Proposition**: The following problem is **PSPACE-complete**.
  
  – **Given**: a network of automata \( \mathcal{A} = \langle A_1, \ldots, A_n \rangle \) and a NFA \( A_V \) recognizing a set of potential executions of \( \mathcal{A} \)
  
  – **Decide**: Is \( L(A_1 \otimes \cdots \otimes A_n \otimes A_V) = \emptyset \) ?
On-the-fly Verification

\[ \text{CheckViol}(A_1, \ldots, A_n, V) \]

**Input:** a network \( \mathcal{A} = \langle A_1, \ldots, A_n \rangle \), where \( A_i = (Q_i, \Sigma_i, \delta_i, Q_{0i}, F_i) \) for \( 1 \leq i \leq n \);
an NFA \( V = (Q_V, \Sigma_V, \delta_V, Q_{0v}, F_v) \).

**Output:** \text{true} if \( L(A_1 \otimes \cdots \otimes A_n \otimes V) \) is nonempty, \text{false} otherwise.

1. \( Q \leftarrow \emptyset; \ Q_0 \leftarrow Q_{01} \times \cdots \times Q_{0n} \times Q_{0v} \)
2. \( W \leftarrow Q_0 \)
3. while \( W \neq \emptyset \) do
   4. pick \( [q_1, \ldots, q_n, q] \) from \( W \)
   5. add \( [q_1, \ldots, q_n, q] \) to \( Q \)
   6. for all \( a \in \Sigma_1 \cup \ldots \cup \Sigma_n \) do
      7. for all \( i \in [1..n] \) do
         8. if \( a \in \Sigma_i \) then \( Q'_i \leftarrow \delta_i(q_i, a) \) else \( Q'_i = \{q_i\} \)
         9. \( Q' \leftarrow \delta_V(q, a) \)
   10. for all \( [q'_1, \ldots, q'_n, q'] \in Q'_1 \times \cdots \times Q'_n \times Q' \) do
      11. if \( \land_{i=1}^n q'_i \in F_i \) and \( q' \in F_v \) then return \text{true}
      12. if \( [q'_1, \ldots, q'_n, q'] \notin Q \) then add \( [q'_1, \ldots, q'_n, q'] \) to \( W \)
13. return \text{false}
Compositional verification

To check emptiness of an asynchronous product $A_1 \otimes \cdots \otimes A_n$ we can

- Replace $A_1$ by an automaton $A'_1$ recognizing $\text{proj}_{\Sigma \setminus \Sigma_1}(L(A_1))$ and compute $A_{12} = A'_1 \otimes A_2$;

- Replace $A_{12}$ by an automaton $A'_{12}$ recognizing $\text{proj}_{\Sigma \setminus (\Sigma_1 \cup \Sigma_2)}(L(A_{12}))$ and compute $A_{13} = A'_{12} \otimes A_3$;

- ... 

- Replace $A_{1(n-1)}$ by an automaton $A'_{1(n-1)}$ recognizing $\text{proj}_{\Sigma \setminus (\Sigma_1 \cup \cdots \cup \Sigma_{n-1})}(L(A_{1(n-1)}))$ and compute $A_{1n} = A'_{1(n-1)} \otimes A_n$

This can save space w.r.t. the direct computation.
Compositional verification

$A_2$
4, 5, 6, 7

$A_1$
2, 3, 6, 7

$A_0$
1, 3, 5, 7

0, 1, 2, 3

0, 1, 4, 5

0, 2, 4, 6
Compositional verification

\[ A_{21} \]

\[ A'_{21} \]
Compositional verification

$A_{20}$ \hspace{1cm} $A'_{20}$ (proj. on visible actions)
Symbolic exploration

- A technique to palliate the state-explosion problem
- Configurations can be encoded as words.
- The set of reachable configurations of a program can be encoded as a language.
- We use automata to compactly store the set of reachable configurations.
Flowgraphs

1 While $x = 1$ do
2     if $y = 1$ then
3         $x \leftarrow 0$
4         $y \leftarrow 1 - x$
5     end
Step relations

- Let \( l, l' \) be two control points of a flowgraph.
- The step relation \( S_{l,l'} \) contains all pairs

\[
( [l, x_0, y_0], [l', x'_0, y'_0] )
\]

of configurations such that:

if at point \( l \) the current values of \( x, y \) are \( x_0, y_0 \),
then the program can take a step,
after which the new control point is \( l' \), and the new values of \( x, y \) are \( x'_0, y'_0 \).
The global step relation $S$ is the union of the step relations $S_{l,l'}$ for all pairs $l, l'$ of control points.

$$S_{4,1} = \{ ( [4, x_0, y_0], [1, x_0, 1 - x_0] ) \mid x_0, y_0 \in \{0,1\} \}$$
Computing reachable configurations

- Start with the set of initial configurations.
- Iteratively: add the set of successors of the current set of configurations until a fixed point is reached.
\[ P_0 = I \]
\[ P_1 = P_0 \cup \text{Post}(P_0, S) \]

\[ P_0 = I \]
\[ P_1 = P_0 \cup \text{Post}(P_0, S) \]

\[ P_0 = I \]

\[ P_2 = P_1 \cup \text{Post}(P_1, S) \]
\[ P_1 = P_0 \cup \text{Post}(P_0, S) \]

\[ P_0 = I \]

\[ P_2 = P_1 \cup \text{Post}(P_1, S) \]
\[ P_1 = P_0 \cup \text{Post}(P_0, S) \]

\[ P_0 = I \]

\[ P_2 = P_1 \cup \text{Post}(P_1, S) \]
Reach$(I, R)$

**Input:** set $I$ of initial configurations; relation $R$

**Output:** set of configurations reachable from $I$

1. $OldP \leftarrow \emptyset; P \leftarrow I$
2. while $P \neq OldP$ do
3. \hspace{1em} $OldP \leftarrow P$
4. \hspace{1em} $P \leftarrow \text{Union}(P, \text{Post}(P, S))$
5. return $P$
Example: Transducer for the global step relation

1 while $x = 1$ do
2 if $y = 1$ then
3 \hspace{1em} x \leftarrow 0
4 \hspace{1em} y \leftarrow 1 - x
5 end
Example: DFAs generated by Reach

- Initial configurations

- Configurations reachable in at most 1 step
Example: DFAs generated by Reach

- Configurations reachable in at most 2 steps
Example: DFAs generated by Reach

- Configurations reachable in at most 3 steps
Variable orders

- Consider the set $Y$ of tuples $[x_1, \ldots, x_{2k}]$ of booleans such that $x_1 = x_{k+1}, x_2 = x_{k+2}, \ldots, x_k = x_{2k}$

- A tuple $[x_1, \ldots, x_{2k}]$ can be encoded by the word $x_1x_2 \ldots x_{2k-1}x_{2k}$ but also by the word $x_1x_{k+1} \ldots x_kx_{2k}$.

- For $k = 3$, the encodings of $Y$ are then, respectively

  $\{000000, 001001, 010010, 011011, 100100, 101101, 110110, 111111\}$
  $\{000000, 000011, 001100, 001111, 110000, 110011, 111100, 111111\}$

- The minimal DFAs for these languages have very different sizes!
Another example: Lamport’s algorithm

\[ \langle v_0, v_1, s_0, s_1 \rangle \]
encoded by
\[ s_0 s_1 v_0 v_1 \]

\[ \langle v_0, v_1, s_0, s_1 \rangle \]
encoded by
\[ v_1 s_1 s_0 v_0 \]
Larger sets can yield smaller DFAs!

- DFAs after adding the configuration \( \langle c_0, c_1, 1, 1 \rangle \) to the set
• When encoding configurations, good variable orders can lead to much smaller automata.
• Unfortunately, the problem of finding an optimal encoding for a language represented by a DFA is NP-complete.