Parameterized Verification
Keeping Crowds Safe

Javier Esparza
Technical University of Munich
Program Verification? Why don’t you give up?“

Theorem (Alan Turing, 1936)
Program termination is undecidable.
Program Verification? Why don’t you give up?“

**Theorem (Alan Turing, 1936)**
Program termination is undecidable.

**Theorem (Henry G. Rice, 1961)**
Every non-trivial property of programs is undecidable.
Program Verification? Why don’t you give up?“

Theorem (Alan Turing, 1936)
Program termination is undecidable.

Theorem (Henry G. Rice, 1961)
Every non-trivial property of programs is undecidable.

Theorem (Marvin Minsky, 1969)
Every non-trivial property of while-programs with two counter variables is undecidable.
Because ...

- Undecidability requires some source of "infinity":
  - Variables with an infinite range
  - Dynamic data structures (lists, trees)
  - Unbounded recursion

- Concurrent systems
  - are difficult to get right, and
  - often have a finite state space.
Dijkstra’s Mutual Exclusion Algorithm

Solution of a Problem in Concurrent Programming Control

E. W. Dijkstra
Technological University, Eindhoven, The Netherlands

A number of mainly independent sequential-cyclic processes with restricted means of communication with each other can be made in such a way that at any moment one and only one of them is engaged in the “critical section” of its cycle.

CACM 8:9, 1965
The program for the $i$th computer ($1 \leq i \leq N$) is:

```
"integer j;
L0:  b[i] := false;
L1:  if k \neq i then
L2:  begin c[i] := true;
L3:  if b[k] then k := i;
   go to L1
end
else
L4:  begin c[i] := false;
    for j := 1 step 1 until N do
       if j \neq i and not c[j] then go to L1
end;
critical section;
c[i] := true;  b[i] := true;
remainder of the cycle in which stopping is allowed;
go to L0"
```
Concurrent programs are often finite-state

The program for the $i$th computer ($1 \leq i \leq N$) is:

```
"integer j;
L10: b[i] := false;
L11: if $k \neq i$ then
L12: begin c[i] := true;
L13: if $b[k]$ then $k :=$
        go to L11
    end
    else
L14: begin c[i] := false;
        for $j := 1$ step 1 until $N$ do
            if $j \neq i$ and not $c[j]$ then go to L11
        end;
critical section;
c[i] := true;  b[i] := true;
remainder of the cycle in which stopping is allowed;
go to L10"
```
A Leader Election Algorithm (90s)

An $O(n \log n)$ Unidirectional Distributed Algorithm for Extrema Finding in a Circle

Danny Dolev, Maria Klawe, and Michael Rodeh*

behavior of an active process $v$.

A0. Send the message $\langle 1, \max(v) \rangle$.

A1. If a message $\langle 1, i \rangle$ arrives do as follows:
   1. If $i \neq \max(v)$ then send the message $\langle 2, i \rangle$, and assign $i$ to left($v$).
   2. Otherwise, halt—$\max(v)$ is the global maximum.

A2. If a message $\langle 2, j \rangle$ arrives do as follows:
   1. If left($v$) is greater than both $j$ and $\max(v)$
      then assign left($v$) to $\max(v)$, and send the message $\langle 1, \max(v) \rangle$.
   2. Otherwise, become passive.
A Cache-Coherence Protocol (00s)

A Model of a Bluetooth Driver (10s)

```c
struct DEVICE_EXTENSION {
    int pendingIo;
    bool stoppingFlag;
    bool stoppingEvent;
};

bool stopped;

void main() {
    DEVICE_EXTENSION *e = malloc(sizeof(DEVICE_EXTENSION));
    e->pendingIo = 1;
    e->stoppingFlag = false;
    e->stoppingEvent = false;
    stopped = false;
    async BCSP_PnpStop(e);
    BCSP_PnpAdd(e);
}
```

```c
int BCSP_IoIncrement(DEVICE_EXTENSION *e) {
    if (e->stoppingFlag)
        return -1;
    atomic {
        e->pendingIo = e->pendingIo + 1;
    }
    return 0;
}

void BCSP_IoDecrement(DEVICE_EXTENSION *e) {
    int pendingIo;
    atomic {
        e->pendingIo = e->pendingIo - 1;
        pendingIo = e->pendingIo;
    }
    if (pendingIo == 0)
        e->stoppingEvent = true;
}

void BCSP_PnpAdd(DEVICE_EXTENSION *e) {
    int status;
    status = BCSP_IoIncrement (e);
    if (status == 0) {
        // do work here
        assert !stopped;
    }
    BCSP_IoDecrement(e);
}

void BCSP_PnpStop(DEVICE_EXTENSION *e) {
    e->stoppingFlag = true;
    BCSP_IoDecrement(e);
    assume e->stoppingEvent;
    // release allocated resources
    stopped = true;
}
```
A Model of a Biochemical System (10s)

Source: Shanghai Institutes for Biological Sciences
Robot swarms, flocks of birds, vehicular networks ... (into the 20s?)

Source: Upenn

Source: Iridia-CoDE

Parameterized Verification

- Model-checking tools can only check instances of these systems for particular values of the number $N$ of processes.

Can we prove correctness for every $N$?
Parameterized Verification

• Model-checking tools can only check instances of these systems for particular values of the number N of processes.

Can we prove correctness *for every N*?

• Amounts to checking an *infinite family of finite-state systems*. 
• The safety /coverability problem:

  – **Given:** a program template \( T[i] \) with finite-range variables, a „dangerous“ control point \( \ell \) of \( T[i] \).

  – **Decide:** Is there a number \( N \) such that the crowd

    \[
    T[1] \parallel T[2] \parallel \cdots \parallel T[N]
    \]

    can reach a global state in which at least one of \( T[1], T[2], \ldots, T[N] \) is at \( \ell \) („covers“ \( \ell \) ?
Parameterized Verification: Give up?

Theorem (folklore): The Halting Problem can be reduced to the parameterized coverability problem.

Reduction:
- The template models the behaviour of one tape cell.
  
  TM terminates
  → it uses a finite number N of cells
  → N copies of the template reach the dangerous control point
Parameterized Verification: Give up?

Reduction:
– The template models the behaviour of one tape cell.

    global var: state: {q₀,...,qₙ}

    var: cellᵢ: {a₁,...aₘ}
    var: activeᵢ: boolean

    if activeᵢ then
        ...
        if state = q and cellᵢ = a then
            state := q'; cellᵢ := a';
            activeᵢ := false; activeᵢ₊₁ := true;
        endif
        ...
    endif
Parameterized Verification: Give up?

Reduction:
- The template models the behaviour of one tape cell.

```plaintext
global var: state: \{q_0, \ldots, q_n\}
```

Hey, what happens at the right border?

```plaintext
... if state = q and cell_i = a then
  state := q'; cell_i := a;
  active_i := false; active_{i+1} := true;
endif
... endif
```

\[(q, a) \rightarrow (q', a', R)\]
Parameterized Verification: Give up?

Reduction:
- The template models the behaviour of one tape cell.

```
global var: state: \{q_0, ..., q_n\}

Hey, what happens at the right border?

... if state = q and cell_i = a then
    state := q'; cell_i := a;
    active_i := false; active_{i+1} := true;
else
    ...
endif
```

OK, just substitute \(i + 1 \mod N\) for \(i + 1\)!

\((q, a) \rightarrow (q', a', R)\)
Parameterized Verification: Give up?

**Theorem (folklore):** The Halting Problem can be reduced to the parameterized coverability problem.
Parameterized Verification: Give up?

Theorem (folklore): The Halting Problem can be reduced to the parameterized coverability problem.
Identities

• In this reduction, processes do not execute exactly the same code
• The code makes use of the process identity (the index $i$) to organize processes in a ring.
Identities

• In this reduction, processes do not execute exactly the same code
• The code makes use of the process identity (the index $i$) to organize processes in a ring.
• But many systems do not use identities:
  – DKR Leader Election uses identities.
  – Dijkstra, MESI-protocol, Bluetooth driver, biochemical systems do not.
Identities

• In this reduction, processes do not execute exactly the same code
• The code makes use of the process identity (the index $i$) to organize processes in a ring.
• But many systems do not use identities:
  – DKR Leader Election uses identities.
  – Dijkstra, MESI-protocol, Bluetooth driver, biochemical systems do not.
• In other systems, processes must remain anonymous!
• Goal: investigate the decidability and complexity of the coverability problem for crowds in which
  (1) every process executes exactly the same code, (anonymous crowds), and
  (2) the number of processes is unknown to the processes.
Coverability Problem for Anonymous Crowds

• **Given:** A finite automaton \( A \) (a template) and a „dangerous“ state \( q_d \) of \( A \)

• **Decide:** Is there a number \( N \) such that the anonymous crowd consisting of \( N \) copies of \( A \) can reach a global state that covers (puts at least one process in) state \( q_d \)?
Coverability Problem for Anonymous Crowds

• **Given:** A finite automaton $A$ (a template) and a „dangerous“ state $q_d$ of $A$

• **Decide:** Is there a number $N$ such that the anonymous crowd consisting of $N$ copies of $A$ can reach a global state that covers (puts at least one process in) state $q_d$?
Coverability Problem for Anonymous Crowds

• **Given:** A finite automaton \( A \) (a template) and a "dangerous" state \( q_d \) of \( A \)

• **Decide:** Is there a number \( N \) such that the anonymous crowd consisting of \( N \) copies of \( A \) can reach a global state that covers (puts at least one process in) state \( q_d \) ?

• **Initial configurations not yet specified.**
Coverability Problem for Anonymous Crowds

• Given: A finite automaton \( A \) (a template) and a „dangerous“ state \( q_d \) of \( A \)
• Decide: Is there a number \( N \) such that the anonymous crowd consisting of \( N \) copies of \( A \) can reach a global state that covers (puts at least one process in) state \( q_d \)?
• Initial configurations not yet specified.
• Communication mechanism not yet specified
Initial Configurations

- A configuration of the system is completely determined by the number of processes in each state of the template (no identities)
- **Configuration**: function \( Q \rightarrow \mathbb{N} \), or vector of natural numbers with \(|Q|\) components, or parallel expression

\[
Q = \{ q_1, q_2, q_3, q_4 \}
\]

\[
(3, 0, 2, 1) \quad q_1^3 \parallel q_2^2 \parallel q_4
\]
Initial Sets of Configurations

• Parametrized configurations with leaders:
  – Exactly one process in some states
  – Zero processes in other states
  – Arbitrary number of processes in the rest
Initial Sets of Configurations

- Parametrized configurations with leaders:
  - Exactly one process in some states \((1,0,N)\)
  - Zero processes in other states \((1,0,1,N_1,N_2)\)
  - Arbitrary number of processes in the rest

- Leaderless parametrized configurations
  - Zero processes in some states
  - Arbitrary number of processes in the rest
Initial Sets of Configurations

• Parametrized configurations with leaders:
  – Exactly one process in some states
  – Zero processes in other states
  – Arbitrary number of processes in the rest

• Leaderless parametrized configurations
  – Zero processes in some states
  – Arbitrary number of processes in the rest

(1,0,N) 
(1,0,1,N₁,N₂)

Crowds with leaders as DEFAULT option
Initial Sets of Configurations

• Parametrized configurations with leaders:
  – Exactly one process in some states
  – Zero processes in other states
  – Arbitrary number of processes in the rest

  \[
  (1, 0, N) \\
  (1, 0, 1, N_1, N_2)
  \]

• Leaderless parametrized configurations
  – Zero processes in some states
  – Arbitrary number of processes in the rest

  \[
  (0, 0, N) \\
  (0, 0, N_1, N_2)
  \]
Initial Sets of Configurations

• Parametrized configurations with leaders:
  – Exactly one process in some states
  – Zero processes in other states
  – Arbitrary number of processes in the rest
  \[ (1,0,N) \quad (1,0,1,N_1,N_2) \]

• Leaderless parametrized configurations
  – Zero processes in some states
  – Arbitrary number of processes in the rest
  \[ (0,0,N) \quad (0,0,N_1,N_2) \]

• Two variants of the problem, depending on the initial parametrized configurations allowed
  – Crowds with leaders as DEFAULT option
Coverability Problem with Leaders

- **Given:** A template $A$, an initial parametrized configuration $c$ of $A$ with leaders, a „dangerous“ state $q_d$ of $A$

- **Decide:** Is there a configuration $c \in C$ such that the crowd starting at $c$ can reach a configuration covering $q_d$?
Equivalent Formulation

• **Given:** A finite set $A_1, \ldots, A_n$ of "leader templates" with initial states $q_1, \ldots, q_n$
a finite set $B_1, \ldots, B_n$ of "crowd templates" with initial states $r_1, \ldots, r_m$,
a "dangerous" state $q_d$

• **Decide:** Are there numbers $k_1, \ldots, k_m$ such that the configuration with
  • one process in each of $q_1, \ldots, q_n$
  • $k_1, \ldots, k_m$ processes in $r_1, \ldots, r_m$
can reach a configuration covering $q_d$?
Coverability Problem without Leaders

• **Given:** A template $A$, an initial parametrized leaderless configuration $c$ of $A$

• **Decide:** Is there a configuration $c \in C$ such that the crowd starting at $c$ can reach a configuration covering $q_a$ ?
Zoo of Communication Mechanisms

**Global guards**
State changes of a process depend on state of all others

**Broadcast**
One process sends, everyone receives immediately

**Rendezvous**
Synchronous exchange between two processes

**Shared memory**
Processes compete for lock
Lock holder reads/writes shared state

**Lock-free shared memory**
No locks, interleaved reads/writes
High or Low Complexity?

Verifiers want low complexity
High or Low Complexity?

Verifiers want low complexity

„Crowd designers“ (swarm intelligence, population protocols, crowdsourcing) want high complexity
Global guards

- Process can make a move if the current state of all other processes satisfies some condition

Examples

- Abstractions of distributed algorithms (bakery, mutex)
- Very hard to implement
The symmetric coverability problem is undecidable for systems communicating with global guards.

Global guards

- Process can make a move if the current state of all other processes satisfies some condition.

Theorem (Emerson and Kahlon, LICS’03)

\[
\forall q_2 \lor q_3 \quad (\forall q_1 \lor q_3) \land \exists q_2
\]

\[
\exists q_1 \land \exists q_3
\]
Counter Programs

• Sequence of labelled commands of the form
  • $\ell: x := x + 1$
  • $\ell: x := x - 1$
  • $\ell: \text{goto } \ell'$
  • $\ell: \text{if } x = 0 \text{ then goto } \ell' \text{ else goto } \ell''$
  • $\ell: \text{halt}$

Theorem (Minsky 69): The problem whether a counter program with all counters initialized to 0 halts is undecidable.
Simulating Counter Programs

• One control template $T$ modelling the control flow of the program, with a state for each program label.
• One counter template $X$ for each counter $x$, with two states: $0_x, 1_x$
  Idea: "$x$ has value $n"$ modeled by $n$ copies of $X$ in state $1_x$
• A configuration $(\ell, n_1, \ldots, n_k)$ of the program is simulated by a configuration of the crowd with
  • one process in $q_\ell$,
  • $n_1$ processes in $1_{x_1}$, ..., $n_k$ processes in $1_{x_k}$
  • All other processes in 0-states
Simulating Counter Programs

- Simulating $\ell$: if $x = 0$ then goto $\ell'$ else goto $\ell''$
Simulating Counter Programs

• Simulating \( \ell: x := x + 1; \ell': \ldots \)
Simulating Counter Programs

• Simulating $\ell: x := x + 1; \ell': \ldots$

```
(\ell) \quad \forall X: 1_x \lor 0_x \quad \exists X: inc_x \quad (\ell')

(0_x) \quad \exists T: inc_x \land \forall X: 0_x \lor 1_x \quad \forall T: \neg inc_x \quad (1_x)

(0_x) \quad \exists T: inc_x \land \forall X: 0_x \lor 1_x \quad \forall T: \neg inc_x \quad (1_x)
```

...
Simulating Counter Programs

- Simulating \( \ell: x := x + 1; \ell': \ldots \)
Simulating Counter Programs

• Simulating $\ell: x := x + 1; \ell': ...$
Simulating Counter Programs

- Simulating $\ell : x := x + 1; \ell' : \ldots$

\[\ell \quad \forall X: 1_x \lor 0_x \quad \text{inc}_x \quad \exists X: \text{inc}_x \quad \ell'\]

\[0_x \quad \exists T: \text{inc}_x \land \forall X: 0_x \lor 1_x \quad \text{inc}_x \quad \forall T: \neg \text{inc}_x \quad 1_x\]

\[0_x \quad \exists T: \text{inc}_x \land \forall X: 0_x \lor 1_x \quad \text{inc}_x \quad \forall T: \neg \text{inc}_x \quad 1_x\]

\[\ldots\]

\[0_x \quad \exists T: \text{inc}_x \land \forall X: 0_x \lor 1_x \quad \text{inc}_x \quad \forall T: \neg \text{inc}_x \quad 1_x\]
Reliable broadcast

- A process sends a message
- All other processes receive the message (instantaneously)

Examples

- Distributed algorithms
- Hardware protocols (cache-coherence) (Emerson and Kahlon 2003)
Reliable broadcast
Reliable broadcast

\[(8, 2, 6, 1)\]
Reliable broadcast

Diagram with nodes labeled 8, 2, 6, and 1, showing transitions labeled as wr-miss?, rd-miss??, wr-miss??, and rd-miss!!.

Nodes 8, 2, and 6 are connected with arrows indicating transitions, and node 1 is connected with an arrow labeled wr.

Additional notes:
- (8, 2, 6, 1) - rd-miss
- (7, 9, 0, 0)
Reliable broadcast

• Theorem [E., Finkel, Mayr 99] The coverability problem for broadcast protocols is decidable.

• Informally:
  Anonymous crowds with local guards are not Turing powerful

• Algorithm: Backward Search
  Abdulla et al. (LICS‘96), based on the theory of well-quasi-orders.
\( C := \text{set of dangerous conf.} \)

**Iterate** \( C := C \cup \text{pre}(C) \) until

- \( C \cap \text{Init} \neq \emptyset; \text{return} \) „unsafe“
- or
- \( C \) fixpoint; \text{return} „safe“

\[ C_1 := C \]
Backward Search

\[ C := \text{set of dangerous conf.} \]

**Iterate** \( C := C \cup \text{pre}(C) \) until

\[ C \cap \text{Init} \neq \emptyset; \text{return ”unsafe“} \]

or

\( C \) fixpoint; return ”safe“

\[ C_1 := C \]

\[ C_2 := C_1 \cup \text{pre}(C_1) \]
\( C := \text{set of dangerous conf.} \)

**Iterate** \( C := C \cup \text{pre}(C) \) until

\( C \cap \text{Init} \neq \emptyset; \text{return} \) „unsafe“

or

\( C \) fixpoint; **return** „safe“
$C := \text{set of dangerous conf.}$

**Iterate** $C := C \cup \text{pre}(C)$ until

$C \cap \text{Init} \neq \emptyset$; **return** „unsafe“

or

$C$ fixpoint; **return** „safe“
$C := \text{set of dangerous conf.}$

**Iterate** $C := C \cup \text{pre}(C)$ **until**

- $C \cap \text{Init} \neq \emptyset$; **return** „unsafe“
- or
- $C$ fixpoint; **return** „safe“

**Problems:**
- $C$ can hold infinite sets. Finite representation?
- Termination?
Finite representation: Upward-closed sets

- **Definition:** $c \leq c'$ if $c'$ has at least as many processes as $c$ in each state
Finite representation: Upward-closed sets

- **Definition:** \( c \leq c' \) if \( c' \) has at least as many processes as \( c \) in each state.
- **Observation 1:** If \( c \) is unsafe and \( c \leq c' \), then \( c' \) is unsafe. We say that the set of unsafe configurations is *upward-closed*. 
Finite representation: Upward-closed sets

- **Definition**: $c \leq c'$ if $c'$ has at least as many processes as $c$ in each state.
- **Observation 1**: If $c$ is unsafe and $c \leq c'$, then $c'$ is unsafe. We say that the set of unsafe configurations is **upward-closed**.
- **Observation 2**: If $C$ is upward-closed then so is $\text{pre}(C)$. 
Finite representation: Upward-closed sets

• **Definition:** $c \leq c'$ if $c'$ has at least as many processes as $c$ in each state.

• **Observation 1:** If $c$ is unsafe and $c \leq c'$, then $c'$ is unsafe. We say that the set of unsafe configurations is **upward-closed**.

• **Observation 2:** If $C$ is upward-closed then so is $\text{pre}(C)$.

• **Observation 3:** The union of upward-closed sets is upward-closed.
Finite representation: Upward-closed sets

- **Definition:** $c \leq c'$ if $c'$ has at least as many processes as $c$ in each state.

- **Observation 1:** If $c$ is unsafe and $c \leq c'$, then $c'$ is unsafe. We say that the set of unsafe configurations is upward-closed.

- **Observation 2:** If $C$ is upward-closed then so is $\text{pre}(C)$.

- **Observation 3:** The union of upward-closed sets is upward-closed.

- **Consequence:** all sets computed by Backward Search are upward-closed.
Finite representation: Well quasi-orders

• Proposition: \( \leq \) is a well-quasi-order: every infinite sequence \( c_1, c_2, c_3 \ldots \) of configurations contains an infinite chain \( c_{i_1} \leq c_{i_2} \leq c_{i_3} \ldots \)
Finite representation: Well quasi-orders

• **Proposition:** $\leq$ is a **well-quasi-order**: every infinite sequence $c_1, c_2, c_3 \cdots$ of configurations contains an infinite chain $c_{i_1} \leq c_{i_2} \leq c_{i_3} \cdots$

• **Consequence:** Every upward-closed set has a finite set of minimal elements, and can be represented by it.
Termination of Backward Search

- Observation: Backward Search computes a sequence $C_0 \subseteq C_1 \subseteq C_2 \cdots$ of upward-closed sets of configurations.
- Theorem: $\bigcup_{i \geq 0} C_i = C_j$ for some $j \geq 0$. 
Termination of Backward Search

• **Observation:** Backward Search computes a sequence $C_0 \subseteq C_1 \subseteq C_2 \cdots$ of upward-closed sets of configurations.

• **Theorem:** $\bigcup_{i \geq 0} C_i = C_j$ for some $j \geq 0$.

• **Consequence:** Backward Search terminates.
Termination of Backward Search

• **Observation:** Backward Search computes a sequence $C_0 \subseteq C_1 \subseteq C_2 \cdots$ of upward-closed sets of configurations.

• **Theorem:** $\bigcup_{i \geq 0} C_i = C_j$ for some $j \geq 0$.

• **Consequence:** Backward Search terminates.

Love it!
Complexity and cut-off bound

Theorem (Schmitz and Schnoebelen 2013)

The coverability problem for broadcast protocols has non-primitive-recursive complexity, even in the leaderless case.
Complexity and cut-off bound

Theorem (Schmitz and Schnoebelen 2013)
The coverability problem for broadcast protocols has non-primitive-recursive complexity, even in the leaderless case.

Consequence: There is a family $\{P_n\}_{n=0}^{\infty}$ of broadcast protocols with $O(n)$ states such that the smallest number of processes required to reach the dangerous state is a non-primitive-recursive function of $n$. 
Theorem (Schmitz and Schnoebelen 2013)

The coverability problem for broadcast protocols has non-primitive-recursive complexity, even in the leaderless case.

Consequence: There is a family \( \{ \xi_\omega \} \) of broadcast protocols with \( \xi \) states such that the smallest number of processes required to reach the dangerous state is a non-primitive-recursive function of \( \xi \).

This round goes to me, Sherlock!
Theorem (Schmitz and Schnoebelen 2013)
The coverability problem for broadcast protocols has non-primitive-recursive complexity, even in the leaderless case.

And yet, backwards reachability is useful for verification! I’ve used it to prove properties of a dozen cache-coherence protocols: their templates have under 10 states!

G. Delzanno
Application to the MESI-protocol

- Are the states M and S mutually exclusive?
- Check if the upward-closed set with minimal element $m = 1$, $e = 0$, $s = 1$, $i = 0$ can be reached from the initial set $m = 0$, $e = 0$, $s = 0$, $i = N$
Application to the MESI-protocol

- Are the states M and S mutually exclusive?
- Check if the upward-closed set with minimal element $m = 1, e=0, s=1, i = 0$ can be reached from the initial set $m=0, e=0, s=0, i = N$

$C_0 : \quad m \geq 1 \land s \geq 1$

$C_1 = C_0 \cup \text{pre}(C_0): \quad (m \geq 1 \land s \geq 1) \lor (m = 0 \land e = 1 \land s \geq 1)$

$C_2 = C_1 \cup \text{pre}(C_1): \quad C_1$
Comm. Mechanisms: Rendez-Vous

**Rendez-vous**
- Synchronous exchange of a message between two processes (binary encounters)

**Examples**
- Biochemical systems
- Vehicular networks
- Communication protocols: equivalent to message passing with bounded channels
Rendez-vous

- Important differences with broadcast:
  - no way to reliably reach all processes
  - the crowd can no longer produce a leader

**Theorem:**
The coverability problem for rendez-vous communication and is EXPSPACE-complete.
The problem for leaderless perametrized configuration can be solved in PTIME.
Lower bound

[Lipton 1976]

A template with $O(n)$ states can simulate a counter counting up to $2^{2^n}$. 
Simulating counter machines

• “$x$ has value $n$” modelled by $n$ processes in state $1_c$
Simulating counter machines

• “$x$ has value $n$” modelled by $n$ processes in state $1_x$

• $\ell: x := x + 1; \ell'$ ... easy to simulate: rendez-vous between a leader and a “unit process”
Simulating counter machines

• “$x$ has value $n$” modelled by $n$ processes in state $1c$
• $ℓ: x := x + 1; ℓ'$ ... easy to simulate: rendez-vous between a leader and a “unit process”

• $ℓ: x := x - 1; ℓ'$ ... similar
Simulating counter machines

• "x has value n" modelled by n processes in state \( 1_c \)
• \( \ell: x := x + 1; \ell' \ldots \) easy to simulate: rendez-vous between a leader and a "unit process"

\[
\begin{align*}
\ell & \xrightarrow{inc_x!} \ell' \\
0x & \xrightarrow{inc_x?} 1x
\end{align*}
\]

• \( \ell: x := x - 1; \ell' \ldots \) similar

• **Problem**: simulate \( \ell: \text{if } x = 0 \text{ then goto } \ell' \text{ else goto } \ell'' \)
  
  Rendez-vous cannot check that no process is in state \( 1_x \)
Simulating counter machines

- Idea: Simulate only counters counting up to $k$
- Introduce for each counter $x$ a complementary counter $\bar{x}$, and maintain the invariant $x + \bar{x} = k$
Simulating counter machines

- Idea: Simulate only counters counting up to $k$
- Introduce for each counter $x$ a complementary counter $\overline{x}$, and maintain the invariant $x + \overline{x} = k$
- Simulate $\ell$: if $x = 0$ then goto $\ell'$ else goto $\ell''$ by a nondeterministic choice

$$\ell: \ \text{goto } \ell_1 \text{ or goto } \ell_2$$

$$\ell_1: \ \overline{x} := \overline{x} - k; \ \overline{x} := \overline{x} + k; \ \text{goto } \ell'$$

$$\ell_2: \ x := x - 1; \ x := x + 1; \ \text{goto } \ell''$$
Simulating counter machines

- Idea: Simulate only counters counting up to $k$
- Introduce for each counter $x$ a complementary counter $\bar{x}$, and maintain the invariant $x + \bar{x} = k$
- Simulate $\ell$: if $x = 0$ then goto $\ell'$ else goto $\ell''$ by a nondeterministic choice

$$\ell: \text{ goto } \ell_1 \text{ or } \text{ goto } \ell_2$$

$$\ell_1: \bar{x} := \bar{x} - k; \bar{x} := \bar{x} + k; \text{ goto } \ell'$$

$$\ell_2: x := x - 1; x := x + 1; \text{ goto } \ell''$$

- The execution of this code can get stuck, but if it terminates it faithfully simulates the zero-test.
Simulating counter machines

- Idea: Simulate only counters counting up to $k$
- Introduce for each counter $x$, a complementary counter $\bar{x}$, and maintain the invariant $x + \bar{x} = k$
- Simulate $\ell$: if $x = 0$ then goto $\ell' \ldots$ else goto $\ell''$ by a nondeterministic choice

\[
\begin{align*}
\ell : & \quad \text{goto } \ell_1 \text{ or goto } \ell_2 \\
\ell_1 : & \quad \bar{x} := \bar{x} - k; \bar{x} := \bar{x} + k; \text{ goto } \ell' \\
\ell_2 : & \quad x := x - 1; x := x + 1; \text{ goto } \ell'' 
\end{align*}
\]

- The execution of this code can get stuck, but if it terminates it faithfully simulates the zero-test.

Cheat!!! You can’t directly simulate me! You only know how to simulate $\bar{x} := \bar{x} - 1$!
Simulating counter machines

• Question: By how large a number $k$ can templates with a total of $O(n)$ states decrease a counter $\overline{x}$?
Simulating counter machines

- **Question:** By how large a number $k$ can templates with a total of $O(n)$ states decrease a counter $\bar{x}$?
- **First answer:** $k = O(n)$
  - One leader template:
Simulating counter machines

- **Question:** By how large a number $k$ can templates with a total of $O(n)$ states decrease a counter $\bar{c}$?
- **Better answer:** $k = 2^{O(n)}$
- **Iterated doubling:**

```
ω

\[ \omega \rightarrow g_0! \rightarrow \text{done}_0? \rightarrow \text{go}_1! \rightarrow \text{go}_1 ! \rightarrow \text{done}_1? \rightarrow \text{done}_1? \rightarrow \text{done}_0! \rightarrow \text{...} \rightarrow \text{go}_{n-1}! \rightarrow \text{go}_n! \rightarrow \text{done}_n? \rightarrow \text{done}_n? \rightarrow \text{done}_{n-1}! \rightarrow \text{...} \rightarrow \text{dec}_x! \rightarrow \text{done}_n! \]```
Simulating counter machines

- **Question:** By how large a number $k$ can templates with a total of $O(n)$ states decrease a counter $\overline{x}$?
- **Best answer (Lipton):** $k = 2^{2^n}$
Simulating counter machines

• **Question:** By how large a number $k$ can templates with a total of $O(n)$ states decrease a counter $\bar{x}$?

• Best answer (Lipton): $k = 2^{2^n}$

• Iterated **squaring**
Iterative squaring

• Given templates simulating $\overline{x} := \overline{x} - m$ as

  \[
  \text{for } i = m \text{ to } 1 \\
  \quad \overline{x} := \overline{x} - 1 \\
  \text{endfor}
  \]

Lipton constructs templates simulating $\overline{x} := \overline{x} - m^2$ as

  \[
  \text{for } j = m \text{ to } 1 \\
  \quad \text{for } i = m \text{ to } 1 \\
  \quad \quad \overline{x} := \overline{x} - 1 \\
  \quad \text{endfor} \\
  \text{endfor}
  \]
Upper bound

Lower bound [Lipton 1976]

Upper bound [Rackoff 1978]:

If the goal state is coverable, then it is coverable in an instance with $2^{2^n}$ processes.
Fix a template $T$ with $n$ states $q_1, \ldots, q_n$. For every configuration $c$ of $T$ (initial or not!), let $\ell(c)$ be the length of the shortest sequence covering $q_1$ (if it exists). Let $f(n)$ be the maximum of all $\ell(c)$. How fast can $f(n)$ grow with $n$?
Upper bound

• Define $g(n, i)$ as $f(n)$, but starting from “ω-configurations” of the form $(k_1, \ldots, k_i, \omega, \ldots, \omega)$ where ω means „arbitrarily many“.

We are interested in $g(n, n) = f(n)$

• We have $g(n, 0) = 1$

• Rackoff shows by induction:

$$g(n, i) = g(n, i - 1)^i + g(n, i - 1)$$

• A little math gives $g(n, n) \leq 2^{2^n}$
Upper bound

Given a sequence $c_0 \rightarrow c_1 \rightarrow \cdots \rightarrow c_K$ such that

• $c_0 = (k_1, \ldots, k_i, \omega, \ldots, \omega)$ and $c_K$ covers $q_d$,

find another sequence $c_0 \rightarrow c'_1 \rightarrow \cdots \rightarrow c'_K$, such that

• $c'_{K'}$ covers $q_d$, and $K' \leq g(n, i) = g(n, i-1)^i + g(n, i-1)$

All $c_i$ different
Upper bound

Case 1

No number here greater than $g(n, i - 1)$
Upper bound

Case 1

\[ K \leq g(n, i - 1)^i \leq g(n, i) \]
Upper bound

Case 2

$m > g(n, i - 1)$
Upper bound

Case 2

$L \leq g(n, i - 1)^i$

Same argument as in Case 1

$m > g(n, i - 1)$
Upper bound

Case 2

\[ L \leq g(n, i - 1)^i \]

No number here greater than \( g(n, i - 1) \)

\[ \leq g(n, i - 1) \]

Induction hypothesis

\[ m > g(n, i - 1) \]
Case 2

$L \leq g(n, i - 1)^i$

$\leq g(n, i - 1)$

$\omega$

$m > g(n, i - 1)$

$m' > m - g(n, i - 1) > 0$

Enough processes!
Upper bound

Case 2

\[ \leq g(n, i - 1)^i + g(n, i - 1) = g(n, i) \]

\[ m > g(n, i - 1) \]

\[ m' > m - g(n, i - 1) > 0 \]
Rendez-vous

Unfortunately, for us verifiers this upper bound is algorithmically pretty useless ...
Theorem [Bozzelli, Ganty 2012]: Backwards reachability runs in double exponential time for rendez-vous systems.
But backwards algorithms often generate too many unreachable states! Can’t you come up with a forward exploration algorithm?

- Graph with $\omega$-configurations as nodes

- Graph with $\omega$-configurations as nodes
- Initial $\omega$-configuration

for $C = (1, 0, N_1, N_2): (1, 0, \omega, \omega)$

- Graph with $\omega$-configurations as nodes
- Initial $\omega$-configuration
  
  For $C = (1,0,N_1,N_2)$: $(1,0,\omega,\omega)$

- Construct a „forward reachability graph“ using $\omega - 1 = \omega$

- Graph with \(\omega\)-configurations as nodes
- Initial \(\omega\)-configuration
  
  for \(C = (1,0,N_1,N_2)\): \((1,0,\omega,\omega)\)

- Construct a „forward reachability graph“ using \(\omega - 1 = \omega\)

- Graph with $\omega$-configurations as nodes
- Initial $\omega$-configuration for $\mathcal{C} = (1, 0, N_1, N_2): (1, 0, \omega, \omega)$
- Construct a „forward reachability graph“ using $\omega - 1 = \omega$

$$
\begin{align*}
q_1 & \xrightarrow{a!} q_2 \\
q_3 & \xrightarrow{a?} q_4
\end{align*}
$$

$(1,0,2,1) \xrightarrow{a} (0,1,1,2)$

- Graph with $\omega$-configurations as nodes
- Initial $\omega$-configuration
  
  for $\mathcal{C} = (1, 0, N_1, N_2)$: $(1, 0, \omega, \omega)$

- Construct a „forward reachability graph“ using $\omega - 1 = \omega$

$\begin{align*}
q_1 \xrightarrow{a!} q_2 \\
q_3 \xrightarrow{a?} q_4^o
\end{align*}$

$(1, 0, 2, 1) \xrightarrow{a} (0, 1, 1, 2)$

$(1, 0, \omega, 1) \xrightarrow{a} (0, 1, \omega, 2)$

- Graph with $\omega$-configurations as nodes
- Initial $\omega$-configuration for $C = (1,0,1) = N_2$:
- Construct a "forward reachability graph" using $\omega$

Problem: termination!!

$(1,0,\omega,1) \xrightarrow{a} (0,1,\omega,2)$

$(1,0,1) \rightarrow (0,1,1,2)$
Forward Algorithms

- Karp-Miller graph: „Accelerate“ the construction:
  If \((\omega, 1, 1, 0) \rightarrow \cdots \rightarrow (\omega, 2, 1, 2)\)
  then \((\omega, 1, 1, 0) \rightarrow \cdots \rightarrow (\omega, 2, 1, 2)\)
  \(\rightarrow \cdots \rightarrow (\omega, 3, 1, 4)\)
  \(\rightarrow \cdots \rightarrow (\omega, 4, 1, 6)\)
  \(\cdots\)
Forward Algorithms

• Karp-Miller graph: „Accelerate“ the construction:
  
  If \((\omega, 1, 1, 0) \rightarrow \cdots \rightarrow (\omega, 2, 1, 2)\)
  
  then \((\omega, 1, 1, 0) \rightarrow \cdots \rightarrow (\omega, 2, 1, 2)\)
  
  \begin{align*}
  &\rightarrow \cdots \rightarrow (\omega, 3, 1, 4) \\
  &\rightarrow \cdots \rightarrow (\omega, 4, 1, 6) \\
  &\cdots
  \end{align*}

  Replace \((\omega, 2, 1, 2)\) by \((\omega, \omega, 1, \omega)\)
Forward Algorithms

• Karp-Miller graph: „Accelerate“ the construction:

\[
\begin{align*}
\text{If } (\omega,1,1,0) \rightarrow & \cdots \rightarrow (\omega,2,1,2) \\
\text{then } (\omega,1,1,0) \rightarrow & \cdots \rightarrow (\omega,2,1,2) \\
& \rightarrow \cdots \rightarrow (\omega,3,1,4) \\
& \rightarrow \cdots \rightarrow (\omega,4,1,6) \\
& \cdots
\end{align*}
\]

Replace \((\omega,2,1,2)\) by \((\omega,\omega,1,\omega)\)

Observe: the replacement is „safe“ with respect to coverability, all configurations under \((\omega,\omega,1,\omega)\) are coverable
Forward Algorithms

• **Theorem (Karp, Miller 69):** The Karp-Miller graph is always finite.
Forward Algorithms

• **Theorem (Karp, Miller 69):** The Karp-Miller graph is always finite.

• **Theorem (Karp, Miller 69):** A state is coverable iff it is covered by some node of the Karp-Miller graph.
Forward Algorithms

- **Theorem (Karp, Miller 69):** The Karp-Miller graph is always finite.
- **Theorem (Karp, Miller 69):** A state is coverable iff it is covered by some node of the Karp-Miller graph.
- **Theorem (Mayr, Meyer 81):** The Karp-Miller graph can have non-primitive recursive size.
Forward Algorithms

• Theorem (Karp, Miller 69): The Karp-Miller graph is always finite.
• Theorem (Karp, Miller 69): A state is coverable iff it is covered by some node of the Karp-Miller graph.
• Theorem (Mayr, Meyer 81): The Karp-Miller graph can have non-primitive recursive size.
• So forward-search more expensive than backward-search?
Forward Algorithms

- Expand, Enlarge, Check (Geeraerts et al. 2004)
  - The Karp-Miller acceleration is „exact“ with respect to coverability: It only introduces an $\omega$ when it is safe to do so.
  - Construct instead a sequence of „underapproximations“ and „overapproximations“:
    - the $i$-th underapproximation contains the state spaces of all instances with at most $i$ processes.
    - the $i$-th overapproximation identifies „more than $i$ processes“ with „arbitrarily many“. 
Forward Algorithms

• **Theorem (Geeraerts et al. 2004):** The Expand-Enlarge-Check algorithm terminates.
  – If $q_d$ is coverable, then some underapproximation discovers it.
  – If $q_d$ is not coverable, then let $K$ be the largest number (not $\omega$!) in the (finite) Karp-Miller graph. The overapproximation for $i = K$ is at least as precise as the Karp-Miller graph.
Forward Algorithms

- **Theorem (Geeraerts et al. 2004):** The Expand-Enlarge-Check algorithm terminates.
  - If \( q_d \) is coverable, then some underapproximation discovers it.
  - If \( q_d \) is not coverable, then let \( K \) be the largest number (not \( \omega \)) in the (finite) Karp-Miller graph. The overapproximation for \( i = K \) is at least as precise as the Karp-Miller graph.

- **Theorem [Majumdar, Zhang 2013]:** The EEC algorithm solves coverability in exponential space.
The Leaderless Case

• Karp-Miller graph for a leaderless parametrized configuration:
  – Initial $\omega$-configuration of the form $(\omega, \ldots, \omega, 0, \ldots, 0)$
The Leaderless Case

• Karp-Miller graph for a leaderless parametrized configuration:
  – Initial $\omega$-configuration of the form $(\omega, \ldots, \omega, 0, \ldots, 0)$

\[
\begin{align*}
q_1 & \xrightarrow{a!} q_2 \\
\omega & \xrightarrow{a} (\ldots, \omega, \omega, \omega, 0, \ldots) & \rightarrow (\ldots, \omega, \omega, \omega, 1, \ldots)
\end{align*}
\]

\[
\begin{align*}
q_3 & \xrightarrow{a?} q_4 \\
\omega & \xrightarrow{a} 0
\end{align*}
\]
The Leaderless Case

- Karp-Miller graph for a leaderless parametrized configuration:
  - Initial $\omega$-configuration of the form $(\omega, \ldots, \omega, 0, \ldots, 0)$

$$
\begin{align*}
q_1 & \quad a! \quad q_2 \\
\omega & \quad \omega \\
q_3 & \quad a? \quad q_4 \\
\omega & \quad 0
\end{align*}
$$

$(\ldots, \omega, \omega, \omega, 0, \ldots) \xrightarrow{a} (\ldots, \omega, \omega, \omega, 1, \ldots)$

is accelerated to

$(\ldots, \omega, \omega, \omega, 0, \ldots) \xrightarrow{a} (\ldots, \omega, \omega, \omega, \omega, \omega, \ldots)$
The Leaderless Case

- Karp-Miller graph for a leaderless parametrized configuration:
  - Initial $\omega$-configuration of the form $(\omega, \ldots, \omega, 0, \ldots, 0)$
    
    $\omega 
    \xrightarrow{a!} \omega 
    
    (\ldots, \omega, \omega, \omega, 0, \ldots) \xrightarrow{a} (\ldots, \omega, \omega, \omega, 1, \ldots)$
    
    is accelerated to
    
    $\omega 
    \xrightarrow{a?} 0 
    
    (\ldots, \omega, \omega, \omega, 0, \ldots) \xrightarrow{a} (\ldots, \omega, \omega, \omega, \omega, \omega, \ldots)$
    
    - So every $\omega$-configuration of the graph contains only 0's and $\omega$'s.
The Leaderless Case

Fact 1: Every $\omega$-configuration of the graph contains only 0's and $\omega$'s.
The Leaderless Case

Fact 1: Every $\omega$-configuration of the graph contains only 0´s and $\omega$´s.

Consequence: The graph has at most $2^n$ states
The Leaderless Case

Fact 1: Every $\omega$-configuration of the graph contains only 0’s and $\omega$’s.

Consequence: The graph has at most $2^n$ states

Fact 2: Along every path the $\omega$‘s increase monotonically
Fact 1: Every $\omega$-configuration of the graph contains only 0´s and $\omega$´s.

Consequence: The graph has at most $2^n$ states

Fact 2: Along every path the $\omega$´s increase monotonically

Consequence: Every simple path of the graph has at most length $n$, and so the coverability problem is in NP
Fact 3: If $(\ldots, 0, \omega, 0, \ldots)$ then $(\ldots, \omega, \omega, 0, \ldots)$ $(\ldots, 0, \omega, \omega, \ldots)$
The Leaderless Case

Fact 3: If 

\[(\ldots, 0, \omega, 0, \ldots)\]

then 

\[(\ldots, 0, \omega, 0, \ldots)\]

Consequence: We can compute the set of \(\omega\)'s reachable after 1, 2, \ldots \(n\) steps in \(\text{PTIME}\)
The Leaderless Case

Theorem (German, Sistla 92): The coverability problem for leaderless parametrized configurations is solvable in PTIME.
Comm. Mechanisms: Shared memory

Shared memory
- A lock for every shared variable
- Process owning the lock can perform reads and writes

Examples
- Multithreaded programs
Shared memory

- Shared memory communication can be simulated by rendez-vous communication, and vice versa.
Shared memory

• Shared memory communication can be simulated by rendez-vous communication, and vice versa.
  – Shared memory → Rendez-vous:
    Consider each shared variable as a leader template with one state for each possible memory value

Theorem: The coverability problem for shared-memory systems is EXPSPACE-complete
Shared memory

• Shared memory communication can be simulated by rendez-vous communication, and vice versa.
  – Shared memory → Rendez-vous:
    Consider each shared variable as a leader template with one state for each possible memory value
  – Rendez-vous → Shared memory: one memory value per action, plus one for „currently empty“
Shared memory

• Shared memory communication can be simulated by rendez-vous communication, and vice versa.
  – Shared memory → Rendez-vous:
    Consider each shared variable as a leader template with one state for each possible memory value
  – Rendez-vous → Shared memory: one memory value per action, plus one for „currently empty“

• **Theorem:** The coverability problem for shared-memory systems is EXPSPACE-complete
The Leaderless Case

- However: the simulation does not preserve „leaderlessness“

**Theorem:** The coverability problem for shared-memory systems EXPSPACE-complete for leaderless initial configurations
Lock-free shared memory

– Concurrent reads and writes allowed
– Interleaving semantics
Comm.Mech: Lock-free Shared Memory
\[ w_d(1) \]
\[ r_d(2) \]
\[ r_d(1) \]
\[ r_d(3) \]

\[ N = 2 \]

\[ 0 \]

\[ w_c(1) \]
\[ w_c(2) \]
\[ w_c(3) \]

\[ r_c(1) \]

\[ r_c(1) \]
\[ r_c(1) \]
\[ r_c(1) \]
The diagram shows a flowchart with nodes labeled with $r_d(1)$, $r_d(2)$, $r_d(3)$, $w_d(1)$, $w_c(1)$, $w_c(2)$, and $w_c(3)$. The nodes are connected by arrows indicating the flow of information or processes, with labels indicating the steps or operations involved in the process.
Theorem (E., Ganty, Majumdar 2013, 2015)
The coverability problem for lock-free shared memory is NP-complete.
In the leaderless case, the problem is polynomial.
Lock-free Shared Memory

- Configuration: triple \((q, v, C)\), where
  - \(q\) : state of the leader
  - \(v\) : current value of the store
  - \(C\) : number of processes in each state of contributor
Lock-free Shared Memory

- We construct the Karp-Miller graph:

  If \([q, v, (\omega, 1, 1, 0)] \rightarrow \cdots \rightarrow [q, v, (\omega, 2, 1, 2)]\)

  then \([q, v, (\omega, 1, 1, 0)] \rightarrow \cdots \rightarrow [q, v, (\omega, 2, 1, 2)]\)

  \[\rightarrow \cdots \rightarrow [q, v, (\omega, 3, 1, 4)]\]

  \[\rightarrow \cdots \rightarrow [q, v, (\omega, 4, 1, 6)]\]

  \[\cdots\]

  Replace \([q, v, (\omega, 2, 1, 2)]\) by \([q, v(\omega, \omega, 1, \omega)]\)
Fact 1: Every $\omega$-configuration of the graph contains only 0’s and $\omega$’s.
Fact 1: Every $\omega$-configuration of the graph contains only 0's and $\omega$'s.

\[ [q, v, (\omega, 0, ...)] \rightarrow [q, v, (\omega, 1, ...)] \]
\[ \rightarrow [q, v, (\omega, 2, ...)] \]
\[ \rightarrow [q, v, (\omega, 3, ...)] \ldots \]

Replace \( [q, v, (\omega, 1, ...)] \) by \( [q, v(\omega, \omega, ...)] \)
Fact 2: In every run, the $\omega$’s grow monotonically.
Fact 2: In every run, the $\omega$’s grow monotonically.

Consequence: Every simple path of the Karp-Miller graph has length $n_l \cdot n_v \cdot n_c$ where

- $n_l$ number of states of leader template
- $n_v$ number of values
- $n_c$ number of states of contributor template

Therefore: coverability is in NP.
Lock-free Shared Memory

Compare with: The coverability problem for a fixed number of contributor processes is PSPACE-complete.
Theorem (E., Ganty, Majumdar 2013)

The problem remains NP-complete if the template is a polytime Turing machine.

This means we cannot distribute an exponentially long computation onto exponentially many machines so that each machine only does polynomial work.
Not covered work and open questions

Termination
Not covered work and open questions

Termination

Temporal logics
Not covered work and open questions

- Termination
- Temporal logics
- Implementations
Not covered work and open questions

And if the processes know $N$?

Termination

Temporal logics

Implementations
That's all!