Model Checking

Javier Esparza and Stephan Merz

Lab. for Foundations of Computer Science, University of Edinburgh
Institut für Informatik, Universität München
9:00–10:00 Basics
   A bit of history
   A case study: the Needham-Schroeder protocol
   Linear and branching time temporal logics
10:00–10:30 Model-checking LTL I
   The automata-theoretic approach
10:30–11:00 Coffee Break
11:00–11:30 Model-checking LTL II
   On-the-fly model checking
   Partial-order techniques
11:30–12:30 Model-checking CTL
   Basic algorithms
   Binary Decision Diagrams
12:30–14:00 Lunch

14:00–15:30 Abstraction
   Basics
   Predicate Abstraction
15:30–16:00 Coffee Break
16:00–17:30 Infinite state spaces
   Sources of infinity
   Symbolic search
   Accelerations and widenings
Basics

A bit of history

A case study: the Needham-Schroeder protocol

Linear and branching time temporal logics
A bit of history

Goal: automatic verification of systems
Prerequisites: formal semantics and specification language

• In the beginning there were Input-Output Systems . . .

  Total correctness = partial correctness + termination
  Formal semantics: input-output relation
  Specification language: first-order logic.

• Late 60s: Reactive systems emerge . . .

  Reactive systems do not “compute anything”
  Termination may not be desirable (deadlock!)
  Total correctness: safety + progress + fairness . . .
  Formal semantics: Kripke structures, transition systems (~ automata)
  Specification language: Temporal logic
Temporal logic

- **Middle Ages:** analysis of modal and temporal inferences in natural language.

  Since yesterday she said she’d come tomorrow, she’ll come today.

- **Beginning of the 20th century:** Temporal logic is formalised

  Primitives: always, sometime, until, since . . .

- **1977:** Pnueli suggests to use temporal logic as specification language

  Temporal formulas are interpreted on Kripke structures
  A. Pnueli: *The Temporal Logic of Programs*. FOCS ’77

```
“System satisfies property”
formalised as
Kripke structure is model of temporal formula
```
Automatising the verification problem

Given a reactive system $S$ and a temporal formula $\phi$, give an algorithm to decide if the system satisfies the formula.

- **Late 70s, early 80s:** reduction to the validity problem
  1. Give a proof system for checking validity in the logic (e.g. axiomatization)
  2. Extract from $S$ a set of formulas $F$
  3. Prove that $F \rightarrow \phi$ is valid using the proof system
    Did not work: step 3 too expensive

- **Early 80s:** reduction to the model checking problem
  1. Construct and store the Kripke structure $\mathcal{K}$ of $S$ → restriction to finite-state systems
  2. Check if $\mathcal{K}$ is a model of $\phi$ directly through the definition

Clarke and Emerson: *Design and synthesis of synchronisation skeletons using branching time temporal logic*. LNCS 131, 1981
Quielle and Sifakis: *Specification and verification of concurrent systems in CESAR*. 5th International Symposium on Programming, 1981
Making the approach work

State explosion problem: the number of reachable states grows exponentially with the size of the system

- **Late 80s, 90s:** Attacks on the problem
  
  **Compress.** Represent sets of states succinctly: Binary decision diagrams, unfoldings.
  **Reduce.** Do not generate irrelevant states: Stubborn sets, sleep sets, ample sets.
  **Abstract.** Aggregate equivalent states: Verification diagrams, process equivalences.

- **90s, 00s:** Industrial applications
  
  Considerable success in hardware verification (e.g. Pentium arithmetic verified)
  Groups in all big companies: IBM, Intel, Lucent, Microsoft, Motorola, Siemens . . .
  Many commercial and non-commercial tools: FormalCheck, PEP, SMV, SPIN . . .
  Exciting industrial and academic jobs!

- **90s, 00s:** Extensions: Infinite state systems, software model-checking
Case study: Needham-Schroeder protocol

Establish joint secret (e.g. pair of keys) over insecure medium

1: \langle A, N_A \rangle_B

2: \langle N_A, N_B \rangle_A

3: \langle N_B \rangle_B

- secret represented by pair \langle N_A, N_B \rangle of “nonces”
- messages can be intercepted
- assume secure encryption and uncompromised keys

Is the protocol secure?
Protocol analysis by model checking

Representation as finite-state system

Finite number of agents Alice, Bob, Intruder

Finite-state model of agents
limit honest agents to single protocol run
one (pre-computed) nonce per agent
describe capabilities of intruder with limited memory

Simple network model shared communication channels

Simulate encryption pattern matching instead of computation

Protocol description in B(PN)$^2$ (Basic Petri Net Programming Notation)

Input language for the PEP tool
http://theoretica.informatik.uni-oldenburg.de/ pep/
B(PN)$^2$ model of honest agents

Model for Alice

begin
  nondeterministically choose partner
  < PartnerAKey’=KeyB OR PartnerAKey’=KeyI >;
  send initial message, modelled as a triple (key,d1,d2)
  < msg1!=PartnerKey AND msg2!=Alice AND msg3!=NonceA >;
  expect matching reply from partner
  < KeyA=msg1? ANDNonceA=msg2? AND PartnerANonce’=msg3? >;
  send final message
  < msg1!=PartnerAKey AND msg2!=PartnerANonce AND msg3!=0 >;
  declare success
  < StatusA’=1 >
end

Similar model for Bob
B(PN)$^2$ model of intruder (1)

begin
do
receive or intercept message, decrypt if possible
  < IntKey'=msg1! AND IntD1'=msg2! AND IntD2'=msg3? >;
do < IntKey=KeyI AND
  (IntD1=NonceA OR IntD2=NonceA) AND KnowNA'=1 >; exit
[] < IntKey=KeyI AND
  (IntD1=NonceB OR IntD2=NonceB) AND KnowNB'=1 >; exit
od; repeat
[] replay intercepted message
  < msg1!=IntKey AND msg2!=IntD1 AND msg3!=IntD2 >; repeat
[] do compose/fake initial message
    choose identity and nonce
    < Sender’=Alice OR Sender’=Bob OR Sender’=Intruder >;
    < (KnowNA=1 AND Nonce’=NonceA) OR
      (KnowNB=1 AND Nonce’=NonceB) OR
      Nonce’=NonceI >;
    < msg1!=KeyB AND msg2!=Sender AND msg3!=Nonce >; exit
[] fake reply if NonceA is known
    choose nonce
    < (KnowNA=1 AND Nonce’=NonceA) OR
      (KnowNB=1 AND Nonce’=NonceB) OR
      Nonce’=NonceI >;
    < KnowNA=1 AND msg1!=KeyA AND msg2!=NonceA AND msg3!=Nonce >; exit
[] fake acknowledgement if NonceB is known
    < KnowNB=1 AND msg1!=KeyB AND msg2!=NonceB AND msg3!=0 >; exit
    od; repeat
    od
end
Protocol analysis using PEP/The Model Checking Kit

Input

B(PN)² model of protocol

Property expressed as temporal logic formula

\[ G(\text{StatusA} = 1 \land \text{StatusB} = 1 \Rightarrow (\text{PartnerAKey} = \text{KeyB} \iff \text{PartnerBKey} = \text{KeyA})) \]

Verification (Sun Ultra 60, 295 MHz, 1.5 GB)

Program automatically translated into Petri net:
130 places, 5461 transitions, 8279 reachable markings

“Compressed” state space computed in about 2 minutes
Shortest run violating the property computed in about 2 seconds
Protocol bug

Alice (correctly) believes to talk with Intruder

Bob (incorrectly) believes to talk with Alice

Bug went undetected for 17 years
Three steps to model checking

1. Model abstraction of system under investigation
   - reduce number of processes
   - limit computational resources
   - increase non-determinism
   - coarser grain of atomicity

2. Validate model
   - simulation ensures existence of certain executions
   - check “obvious” properties

3. Run model checker for properties of interest
   - “true” property holds of model, and perhaps of system
   - “false” counterexample guides debugging of model and/or system
   - timeout review model, tune parameters of model checker
Kripke structures

Basic model of computation \( K = (S, I, \delta, AP, L) \)

- \( S \) system states (control, variables, channels)
- \( I \subseteq S \) initial states
- \( \delta \subseteq S \times S \) transition relation
- \( AP \) atomic propositions over states
- \( L : S \rightarrow 2^{AP} \) (labels) labelling function

All states assumed to have at least one successor

\( K \) described in modelling language (e.g., B(PN)\(^2\), Petri nets, process algebra)

Size of \( K \) usually exponential in size of description

Petri net view

- \( S \) reachable markings
- \( AP \) set of places
- \( L(M) \) set of places marked at \( M \)
Example: Petri net
Example: Kripke structures
Computations of Kripke structures

Computations of $\mathcal{K} = (S, I, \delta, AP, L)$

infinite sequences $L(s_0)L(s_1) \ldots \in S^\omega$ satisfying $s_0 \in I$ and $(s_i, s_{i+1}) \in \delta$

Petri net view

infinite sequences of markings $M_0M_1 \ldots$ starting at an initial marking and obeying the firing rule

Computation tree represents all computations of $\mathcal{K}$

- nodes: system states
- edges: transitions
- paths: computations
- branching: non-determinism (e.g., interleaving)
Linear-time temporal logic (LTL)

Express time-dependent properties of system runs

Evaluated over infinite sequences of labels (computations or not)

<table>
<thead>
<tr>
<th>type</th>
<th>formula</th>
<th>$\rho \models \varphi$ iff ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>atomic</td>
<td>$p \in AP$</td>
<td>$p$ holds of $\rho_0$</td>
</tr>
<tr>
<td>boolean</td>
<td>$\neg \varphi$</td>
<td>$\rho \not\models \varphi$</td>
</tr>
<tr>
<td></td>
<td>$\varphi \lor \psi$</td>
<td>$\rho \models \varphi$ or $\rho \models \psi$</td>
</tr>
<tr>
<td>temporal</td>
<td>$X \varphi$</td>
<td>$\rho</td>
</tr>
<tr>
<td></td>
<td>$F \varphi$</td>
<td>$\rho</td>
</tr>
<tr>
<td></td>
<td>$G \varphi$</td>
<td>$\rho</td>
</tr>
<tr>
<td></td>
<td>$\varphi \text{ until } \psi$, $\varphi \mathbf{U} \psi$</td>
<td>there is $i \in \mathbb{N}$ such that $\rho</td>
</tr>
<tr>
<td></td>
<td>$\varphi \text{ unless } \psi$, $\varphi \mathbf{W} \psi$</td>
<td>$\rho \models \varphi \text{ until } \psi$ or $\rho \models G \varphi$</td>
</tr>
</tbody>
</table>

System validity: $\mathcal{K} \models \varphi$ iff $\sigma \models \varphi$ for all computations of $\mathcal{K}$
LTL: examples

Invariants

\[ \text{mutual exclusion} \]

\[ \text{deadlock freedom} \]

Response, recurrence

\[ \text{eventual access to critical section} \]

\[ \text{no starvation in critical section} \]

Reactivity, Streett

\[ \text{strong fairness} \]

Precedence

\[ \text{1-bounded overtaking} \]
Branching-time temporal logic

Include assertions about branching behavior

combine temporal modalities and quantification over paths

Example: CTL  Computation Tree Logic

\[ \text{Q T} \]

for some path \ E \ A \ X \ F \ G \ U, W \]

successor (next)
some time in the future (eventually)
always in the future (globally)
until, unless

evaluated at subtree \( K, s \models \varphi \)

system validity \( K \models \varphi \) iff \( K, s \models \varphi \) for all \( s \in I \)

Possibility properties

\[ AG \ EF \ init \] home state, resettability
Linear vs. branching time

Incomparable expressiveness of LTL and CTL

- LTL cannot express possibility properties
- CTL cannot express $F G p$

Choose your logic depending on problem requirements

More expressive logics: CTL*, $\mu$-calculus
Model-checking LTL I

The automata-theoretic approach
Büchi automata

Finite automata operating on $\omega$-words $B = (Q, I, \delta, F)$

- $Q$ finite set of states
- $I \subseteq Q$ initial states
- $\delta \subseteq Q \times \Sigma \times Q$ transition relation
- $F \subseteq Q$ accepting states

Run of $B$ on $\omega$-word $a_0 a_1 \ldots \in \Sigma^\omega$

- sequence $q_0 \xrightarrow{a_0} q_1 \xrightarrow{a_1} q_2 \ldots$
- initialization $q_0 \in I$
- consecution $(q_i, a_i, q_{i+1}) \in \delta$ for all $i \in \mathbb{N}$
- accepting $q_i \in F$ for infinitely many $i \in \mathbb{N}$

$\omega$-language defined by $B$

$\mathcal{L}(B) = \{w \in \Sigma^\omega : B$ has some accepting run on $w\}$

$\omega$-regular languages class of ($\omega$-)languages definable by Büchi automata
Büchi automata: examples

infinitely often ‘b’

infinitely often ‘ab’

eventually only ‘b’

not definable by deterministic Büchi automaton
Büchi automata: basic properties

Decidability of emptiness problem

\[ \mathcal{L}(B) \neq \emptyset \quad \text{iff} \quad \exists q_0 \in I, q \in F \text{ such that } q_0 \xrightarrow{\Sigma^*} q \xrightarrow{\Sigma^+} q \]

complexity linear in \(|Q|\) (NLOGSPACE)

Closure properties

- union standard NFA construction
- intersection “marked” product
- complement difficult construction \(O(2^n \log n)\) states
- projection \(\Sigma \rightarrow \Sigma'\)
Other kinds of $\omega$-automata

**Generalized Büchi automata** $\mathcal{B} = (Q, I, \delta, \{F_1, \ldots, F_n\})$

- run accepting iff infinitely many $q_i \in F_k$, for all $k$
- can be coded as a Büchi automaton with additional counter $(\mod n)$
- intersection definable via product automaton

**Muller automata** $\mathcal{M} = (Q, I, \delta, \mathcal{F})$

- run accepting iff set of states attained infinitely often $\in \mathcal{F}$
- special case: Streett automata, can be exponentially more succinct than Büchi automata

**Alternating automata**

- transition relation $\delta \subseteq Q \times \Sigma \times 2^Q$
- several states can be simultaneously active
- unifying framework for encoding linear-time and branching-time logics
From LTL to (generalized) Büchi automata

Basic insight

- Let $\mathcal{L}(\varphi)$ be the set of sequences of labels satisfying $\varphi$
- Construct automaton $\mathcal{B}_\varphi$ that accepts precisely $\mathcal{L}(\varphi)$
  Alphabet of $\mathcal{B}_\varphi$ is $2^{AP}$

Idea of construction

<table>
<thead>
<tr>
<th>states</th>
<th>sets of subformulas of $\varphi$ intended to be true at the next position in the sequence of labels</th>
</tr>
</thead>
<tbody>
<tr>
<td>initial states</td>
<td>states containing $\varphi$</td>
</tr>
<tr>
<td>transition relation</td>
<td>ensures satisfaction of non-temporal formulas in source state</td>
</tr>
<tr>
<td></td>
<td>replaces temporal formulas in source by others in target</td>
</tr>
<tr>
<td></td>
<td>temporal formulas decomposed according to recursion laws</td>
</tr>
<tr>
<td>$G \varphi$</td>
<td>$\equiv \varphi \land X G \varphi$</td>
</tr>
<tr>
<td>$F \varphi$</td>
<td>$\equiv \varphi \lor X F \varphi$</td>
</tr>
<tr>
<td>$\varphi$ until $\psi$</td>
<td>$\equiv \psi \lor (\varphi \land X(\varphi$ until $\psi))$</td>
</tr>
<tr>
<td>accepting states</td>
<td>defined from “eventualities” $F \varphi$ or $\varphi$ until $\psi$</td>
</tr>
</tbody>
</table>
Example: $G(p \Rightarrow F q)$

Subformulas

$\{G(p \Rightarrow F q), p \Rightarrow F q, p, F q, q\} \cup \text{negations}$

Examples of states

$\{G(p \Rightarrow F q), p \Rightarrow F q, \neg p, F q, q\}$

$\{\neg(G(p \Rightarrow F q)), p \Rightarrow F q, p, F q, \neg q\}$

$\{\neg(G(p \Rightarrow F q)), \neg(p \Rightarrow F q), p, \neg F q, \neg q\}$

Example of transitions

$\{G(p \Rightarrow F q), p \Rightarrow F q, \neg p, F q, q\} \xrightarrow{\{\neg p, q\}} \{G(p \Rightarrow F q), p \Rightarrow F q, p, F q, \neg q\}$

Sets of final states

States containing $\neg F p$ or $p$

States containing $\neg(G(p \Rightarrow F q)) \equiv F \neg(p \Rightarrow F q)$ or $\neg(p \Rightarrow F q)$
Result for the example (improved construction)

\[ \neg p \vee q \quad p \wedge \neg q \]

\[ S_0 \rightarrow S_1 \]

\[ q \quad \neg q \]

Complexity

- worst case: \( B_\varphi \) exponential in length of \( \varphi \)
- improved constructions try to avoid exponential blow-up

Application  LTL decision procedure

- \( \varphi \) satisfiable iff \( L(B_\varphi) \neq \emptyset \)
- exponential complexity (PSPACE)
Model Checking

Problem  Given $\mathcal{K}$ and $\varphi$, decide whether $\mathcal{K} \models \varphi$

Automata-theoretic solution

Consider $\mathcal{K}$ as $\omega$-automaton with all states final
Define $L(\mathcal{K}) = \text{set of computations of } \mathcal{K}$

$$\mathcal{K} \models \varphi \iff L(\mathcal{K}) \subseteq L(\varphi) \iff L(\mathcal{K}) \cap L(\neg \varphi) = \emptyset \iff L(\mathcal{K} \times B_{\neg \varphi}) = \emptyset$$

Complexity $O(|\mathcal{K}| \cdot |B_{\neg \varphi}|) = O(|\mathcal{K}| \cdot 2^{|\varphi|})$
State explosion

$\mathcal{K} \times B_{\neg \varphi}$ is too big to be computed effectively

Problems start around $10^6$ states

Solutions

- **Reduce**: ignore irrelevant portions of $\mathcal{K} \times B_{\neg \varphi}$
- **Compress**: construct compact representation of $\mathcal{K} \times B_{\neg \varphi}$
- **Abstract**: see section on abstraction
Model-checking LTL II

On-the-fly model checking

Partial-order techniques
On-the-fly LTL model checking

Basic insight

– Construct only reachable states of $\mathcal{K} \times B_{\neg \varphi}$
– Stop if a word in $\mathcal{L}(\mathcal{K} \times B_{\neg \varphi})$ (acceptance cycle) is found

Setup

– Consider pairs $(s, q)$ of states of $\mathcal{K}$ and $B_{\neg \varphi}$

  • initial pairs both components initial
  • successors joint execution of $\mathcal{K}$ and $B_{\neg \varphi}$
  • accepting pairs second component accepting for $B_{\neg \varphi}$

“On-the-fly” search for acceptance cycles [Courcoubetis et al 1992]

– depth-first search for accepting pair reachable from itself
– interleave state generation and search for cycle
– stack of pairs whose successors need to be explored (contains counterexample)
– hashtable of pairs already seen (in current search mode)
On-the-fly LTL model checking

def dfs(boolean search_cycle) {
    p = top(stack);
    foreach (q in successors(p)) {
        if (search_cycle and (q == seed))
            report acceptance cycle and exit;
        if (((q, search_cycle) not in visited) {
            enter (q, search_cycle) into visited;
            push q onto stack;
            dfs(search_cycle);
            if (not search_cycle and (q is accepting)) {
                seed = q; dfs(true);
            }
        }
    }
    pop(stack);
}
// initialization
visited = emptyset(); stack = emptystack(); seed = null;
foreach initial pair p {
    push p onto stack;
    enter (q, false) into visited;
    dfs(false)
}
Partial-order reduction (Petri net view)

Transitions $t$, $u$ are **independent** if $(\bullet t \cup t^\bullet) \cap (\bullet u \cup u^\bullet) = \emptyset$

**Examples**

- assignments to different variables of values that do not depend on the other variable
- sending and receiving on a channel that is neither empty nor full

**Idea: avoid exploring independent transitions** . . .

. . . is correct if the property cannot distinguish their order and every transition is eventually considered

. . . may lead to exponential reduction in part of system explored

**Practical issues**

Select at each new state an appropriate subset of the enabled transitions
Selecting a smallest subset is untractable
Linear or quadratic suboptimal algorithms
Different techniques: stubborn sets, sleep sets, ample sets
Stubborn sets

A set $U$ of transitions is stubborn at a marking $M$ if

- for every $t \in U$, and every $\sigma \in (T \setminus U)^*$

  $$M \xrightarrow{\sigma t} M' \text{ implies } M \xrightarrow{t\sigma} M'$$

- either no transition is enabled at $M$, or there is $t \in U$ such that for every $\sigma \in (T \setminus U)^*$

  $$M \xrightarrow{\sigma} \text{ implies } M \xrightarrow{\sigma t} M'$$

Reduced transition systems constructed using stubborn sets contain all deadlock states and preserve existence of infinite paths

**Efficiently** constructing a stubborn set at a marking $M$:

- start with $U = \{t\}$ for some $t$ enabled at $M$
- if $t \in U$ and $t$ enabled, then add $(\bullet t)\bullet$ (or $\bullet(\bullet t)$) to $U$
- if $t \in U$ and $t$ not enabled, then take $p \in \bullet t$ such that $M(p) = 0$ and add $\bullet p$ to $U$

More complicated definitions for preservation of LTL properties
Examples

Deadlock freedom can be decided by exploring only six states

Needham-Schroeder: property checked by PROD after examining 942 states (out of 8279)
Partial-order compression (Petri net view)

Based on “true concurrency” theory

Unfolding of a Petri net

Obtained through “unrolling”
Acyclic, possibly infinite net
Equivalent to the original net for all sensible equivalence notions

Checking procedure for a property $\varphi$

Generate a Petri net $N \times B_{\neg \varphi}$ with “final places”
Generate a finite prefix of the unfolding of $N \times B_{\neg \varphi}$ to decide if $\mathcal{L}(N \times B_{\neg \varphi}) = \emptyset$
Prefix can be exponentially more compact than $K \times B_{\neg \varphi}$
Examples

Needham-Schroeder: prefix with 3871 events
Model Checking CTL

Basic Algorithms

Binary Decision Diagrams
Computation Tree Logic (CTL)

Branching structure and temporal modalities

<table>
<thead>
<tr>
<th>Type</th>
<th>Formula $\varphi$</th>
<th>$\mathcal{K}, s_0 \models \varphi$ iff ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>atomic</td>
<td>$p \in AP$</td>
<td>$p$ holds of $s_0$</td>
</tr>
<tr>
<td>propositional</td>
<td>$\neg \varphi$</td>
<td>$\mathcal{K}, s_0 \not\models \varphi$</td>
</tr>
<tr>
<td></td>
<td>$\varphi \lor \psi$</td>
<td>$\mathcal{K}, s_0 \models \varphi$ or $\mathcal{K}, s_0 \models \psi$</td>
</tr>
<tr>
<td>temporal</td>
<td>$\text{EX} \varphi$</td>
<td>exists path $s_0 s_1 \ldots$ s.t. $\mathcal{K}, s_1 \models \varphi$</td>
</tr>
<tr>
<td></td>
<td>$\text{AF} \varphi$</td>
<td>for all paths $s_0 s_1 \ldots$ exists $i \in \mathbb{N}$ s.t. $\mathcal{K}, s_i \models \varphi$</td>
</tr>
<tr>
<td></td>
<td>$\varphi \text{EU} \psi$</td>
<td>exists path $s_0 s_1 \ldots$ and $i \in \mathbb{N}$ s.t. $\mathcal{K}, s_i \models \psi$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>and $\mathcal{K}, s_j \models \varphi$ for all $0 \leq j &lt; i$</td>
</tr>
</tbody>
</table>

invariants $\ \text{AG} \neg (\text{crit}_1 \wedge \text{crit}_2)$

home state, resettability $\ \text{AG EF} \ reset$
CTL model checking

Idea: label states with formulas they satisfy

Recall system validity:

\[ \mathcal{K} \models \varphi \iff \mathcal{K}, s \models \varphi \text{ for all } s \in I \]

iff \[ I \subseteq \lbrack \varphi \rbrack_{\mathcal{K}} \]

where \[ \lbrack \varphi \rbrack_{\mathcal{K}} = \text{def} \{ s \in S \mid \mathcal{K}, s \models \varphi \} \]

Model checking requires:

- algorithm to compute \[ \lbrack \varphi \rbrack_{\mathcal{K}} \]
- data structures to represent and manipulate sets of states
Bottom-up calculation of $[[\varphi]]_K$

**Observation:** all CTL formulas definable from \(EX\), \(EG\), and \(EU\), e.g.

\[
\begin{align*}
AX \varphi & \equiv \neg EX \neg \varphi & EF \varphi & \equiv \text{true} \ EU \varphi \\
AG \varphi & \equiv \neg EF \neg \varphi & AF \varphi & \equiv \neg EG \neg \varphi
\end{align*}
\]

**simple cases:** reformulation of CTL semantics

\[
\begin{align*}
[[p]]_K & = \{ s \in S \mid p \in L(s) \} \text{ for } p \in AP \\
[[\neg \psi]]_K & = S \setminus [[\psi]]_K \\
[[\psi_1 \lor \psi_2]]_K & = [[\psi_1]]_K \cup [[\psi_2]]_K \\
[[EX \psi]]_K & = \delta^{-1}([[\psi]]_K) =_{\text{def}} \{ s \in S \mid t \in [[\psi]]_K \text{ for some } t \text{ s.t. } (s, t) \in \delta \}
\end{align*}
\]

**missing cases:** $[[EG \varphi]]_K$, $[[\varphi \ EU \psi]]_K$
Calculation of $\llbracket EG \varphi \rrbracket_\mathcal{K}$

Observe recursion law

$$EG \varphi \equiv \varphi \land EX EG \varphi$$

In fact:

$\llbracket EG \varphi \rrbracket_\mathcal{K}$ is the greatest “solution” of $X = \llbracket \varphi \rrbracket_\mathcal{K} \cap \delta^{-1}(X)$ in $(2^S, \subseteq)$

Proof.

- Recursion law implies that $\llbracket EG \varphi \rrbracket_\mathcal{K}$ is a solution.
- Assume $M = \llbracket \varphi \rrbracket_\mathcal{K} \cap \delta^{-1}(M)$ for $M \subseteq S$, show $M \subseteq \llbracket EG \varphi \rrbracket_\mathcal{K}$. Assume $s_0 \in M$.

1. $s_0 \in \llbracket \varphi \rrbracket_\mathcal{K}$ implies $\mathcal{K}, s_0 \models \varphi$.
2. $s_0 \in \delta^{-1}(M)$ implies there is $s_1 \in M$ s.t. $(s_0, s_1) \in \delta$.

Inductively obtain path $s_0, s_1, \ldots$ of states satisfying $\varphi$.

This proves $\mathcal{K}, s_0 \models EG \varphi$ and thus $s_0 \in \llbracket EG \varphi \rrbracket_\mathcal{K}$. 

46
Kleene’s fixed point theorem implies:

\[
\left[ [\mathsf{EG} \varphi]_K \right] \quad \text{can be computed as the limit of}
\]

\[
S, \; \pi(S), \; \pi(\pi(S)), \; \ldots \quad \text{for} \quad \pi : \begin{cases} 
2^S \rightarrow 2^S \\
X \mapsto [\varphi]_K \cap \delta^{-1}(X)
\end{cases}
\]

**Convergence:** obvious, because \( S \) is finite
Computation of greatest fixed point (1)

Compute $[E G y]$

$$\pi^0(S) = S$$
Computation of greatest fixed point (2)

Compute \([\text{EG} \ y]\)

$$\pi^1(S) = \llbracket y \rrbracket_K \cap \delta^{-1}(S)$$
Compute \[ [EG\ y] \]

\[
\pi^2(S) = [[y]_K \cap \delta^{-1}(\pi^1(S))]
\]
Computation of greatest fixed point (4)

Compute \([\text{EG } y]\)

\[
\pi^3(S) = \llbracket y \rrbracket_K \cap \delta^{-1}(\pi^2(S)) = \pi^2(S) : \llbracket \text{EG } y \rrbracket_K = \{s_0, s_2, s_4\}
\]
Calculation of $[[\varphi \text{ EU } \psi]]_K$

Similarly:

$$\varphi \text{ EU } \psi \equiv \psi \lor (\varphi \land \text{EX}(\varphi \text{ EU } \psi))$$

$$[[\varphi \text{ EU } \psi]]_K$$ is the smallest solution of $X = [[\psi]]_K \cup ([[\varphi]]_K \cap \delta^{-1}(X))$

Computation: calculate the limit of

$$\emptyset, \pi(\emptyset), \pi(\pi(\emptyset)), \ldots$$ for $\pi : \begin{cases} 2^S \to 2^S \\ X \mapsto [[\psi]]_K \cup ([[\varphi]]_K \cap \delta^{-1}(X)) \end{cases}$
Compute \[ \left[ \mathcal{EF}( (x = z) \land (x \neq y) ) \right]_k \]

\[ \pi^0(\emptyset) = \emptyset \]
Compute \[
\llbracket \mathbf{EF}((x = z) \land (x \neq y)) \rrbracket
\]

\[\pi^1(\emptyset) = \llbracket (x = z) \land (x \neq y) \rrbracket_K \cup \delta^{-1}(\emptyset)\]
Compute \[ \mu \exists \ 
\begin{equation} \begin{aligned} \left\lbrack \text{EF}((x = z) \land (x \neq y))\right\rbrack_K \end{aligned} \end{equation} \]

\[ \pi^2(\emptyset) = \left\lbrack (x = z) \land (x \neq y) \right\rbrack_K \cup \delta^{-1}(\pi^1(\emptyset)) \]

55
Computation of least fixed point (4)

Compute $\llbracket \text{EF}((x = z) \land (x \neq y)) \rrbracket_K$

$\pi^3(\emptyset) = \llbracket (x = z) \land (x \neq y) \rrbracket_K \cup \delta^{-1}(\pi^2(\emptyset))$
Compute \( \llbracket \text{EF}( (x = z) \land (x \neq y)) \rrbracket_K \)

\[ \pi^4(\emptyset) = \llbracket (x = z) \land (x \neq y) \rrbracket_K \cup \delta^{-1}(\pi^3(\emptyset)) = \pi^3(\emptyset): \]

\[ \llbracket \text{EF}( (x = z) \land (x \neq y)) \rrbracket_K = \{s_4, s_5, s_6, s_7\} \]
Complexity issues

Complexity of fixed point algorithm:  \( O(|\varphi| \cdot |S| \cdot (|S| + |\delta|)) \)

Improved algorithm  [Clarke, Emerson, Sistla 1986]

- Computation of \([\mathcal{E}G \varphi]_K\)
  1. restrict \(K\) to states satisfying \(\varphi\)
  2. compute SCCs of restricted graph
  3. find states from which some SCC is reachable, using backward search

- Computation of \([\varphi \mathcal{E}U \psi]_K\) can similarly be reduced to backward search

Complexity:  \( O(|\varphi| \cdot (|S| + |\delta|)) \)  linear in size of model and formula
Fairness constraints

Recall limited expressiveness of CTL: fairness conditions not expressible

Instead: modify semantics and model checking algorithm

FairCTL: exclude “unfair” paths, e.g.

\[ \mathcal{K}, s_0 \models \text{EG}_f \varphi \iff \text{there exists fair path } s_0, s_1, \ldots \text{ s.t. } \mathcal{K}, s_i \models \varphi \text{ for all } i \]

\[ \mathcal{K}, s_0 \models \text{AG}_f \varphi \iff \mathcal{K}, s_i \models \varphi \text{ holds for all fair paths } s_0, s_1, \ldots \text{ and all } i \]

Fairness conditions specified by additional constraints

SMV: indicate CTL formulas that must hold infinitely often along a fair path

Key property: suffix closure

path \( s_0, s_1, s_2, \ldots \) is fair iff \( s_n, s_{n+1}, s_{n+2}, \ldots \) is fair (for all \( n \))
Model checking FairCTL

Observe: \( \text{EG}_f \text{true} \) holds at \( s \) iff there is some fair path from \( s \)

Suffix closure ensures

\[
\begin{align*}
\text{EX}_f \varphi & \equiv \text{EX}(\varphi \land \text{EG}_f \text{true}) \\
\varphi \text{ EU}_f \psi & \equiv \varphi \text{ EU} (\psi \land \text{EG}_f \text{true})
\end{align*}
\]

Therefore: need only modify algorithm to compute \( \lbrack \text{EG}_f \varphi \rbrack_K \)

assume \( k \) SMV-style fairness constraints: \( \psi_1 \land \ldots \land \psi_k \)

1. restrict \( K \) to states satisfying \( \varphi \)
2. compute SCCs of restricted graph
3. remove SCCs that do not contain a state satisfying \( \psi_j \), for some \( j \)
4. \( \lbrack \text{EG}_f \varphi \rbrack_K \) consists of states from which some (fair) SCC is reachable

Complexity: \( O(|\varphi| \cdot (|S| + |\delta|) \cdot k) \) still linear in the size of the model
Symbolic CTL model checking

Compress: data structures for model checking algorithm

compact representation of sets $[\phi]_{\mathcal{K}} \subseteq S$ and relation $\delta \subseteq S \times S$

Operations required

- Boolean operations on sets union, intersection, complement
- inverse image operation $\delta^{-1}(M)$
- comparison detect termination of fixed point computation

BDDs (binary decision diagrams) [Bryant 1986]

- widely used data structure for boolean functions
- compact, canonical dag representation of binary decision trees
- can represent large sets of regular structure
Compact set representations

Assume states are valuations of Boolean variables $x_0, x_1, y_0, y_1$

Example: set of states such that sum $x_1x_0 \oplus y_1y_0$ produces carry

- explicit enumeration \{ $\overline{x_0}x_1\overline{y}_0y_1, x_0\overline{x}_1y_0y_1, x_0x_1\overline{y}_0y_1, x_0x_1y_0\overline{y}_1, x_0x_1y_0y_1$ \}
- decision tree set elements correspond to paths leading to 1
- BDD dag obtained by removing redundant nodes and sharing equal subtrees
BDD implementation

Constructors

- constant BDDs  \textbf{true, false}
  \[
  \begin{array}{c}
  1 \\
  0 \\
  \end{array}
  \]

- inner nodes  \( BDD(v, f, g) \)

Observe global invariants:

- along any path, variables occur in same order (if at all)
- subdags of inner node are always distinct
- avoid reallocation of equivalent BDD nodes (use hash table)

Therefore:

- BDD uniquely determined by Boolean function
- equivalence checking reduces to testing pointer equality
Boolean operations for BDDs

**basic operation** \( \text{ite}(f, g, h) = (f \land g) \lor (\neg f \land h) \)  

“if then else”

all Boolean connectives definable from \( \text{ite} \) and constants

**recursive computation**

\[
\text{ite}(\text{true}, g, h) = g \quad \text{ite}(\text{false}, g, h) = h
\]

Else: let \( \nu \) be “smallest” variable in \( f, g, h \)

\[
\text{ite}(f, g, h) = \nu \land \text{ite}(f|_{\nu=\text{true}}, g|_{\nu=\text{true}}, h|_{\nu=\text{true}}) \lor
\]

\[
\neg \nu \land \text{ite}(f|_{\nu=\text{false}}, g|_{\nu=\text{false}}, h|_{\nu=\text{false}})
\]

\[
= \begin{cases} 
\text{ite}(f|_{\nu=\text{true}}, \ldots) & \text{if } \text{ite}(f|_{\nu=\text{true}}, \ldots) = \text{ite}(f|_{\nu=\text{false}}, \ldots) \\
\text{BDD}(\nu, \text{ite}(f|_{\nu=\text{true}}, \ldots), \text{ite}(f|_{\nu=\text{false}}, \ldots)) & \text{otherwise}
\end{cases}
\]

**Cofactor** \( f|_{\nu=\text{true}}, f|_{\nu=\text{false}} \) for \( \nu \) at most head variable of \( f \)

equals left or right sub-dag of \( f \) if \( \nu \) is head variable, otherwise equals \( f \)

**Complexity:** \( O(|f| \cdot |g| \cdot |h|) \) if recomputation is avoided by hashing
BDD implementation: quantifiers

projection \( \exists x : \varphi \) = (\( \varphi \mid_{x=true} \lor \varphi \mid_{x=false} \))

quantification over head variable

\[ \exists x : BDD(x, f, g) = \exists x : (x \land f) \lor (\neg x \land g) \text{ [Def. BDD]} \]

\[ = (true \land f) \lor (\neg true \land g) \lor (false \land f) \lor (\neg false \land g) \text{ [note: x does not occur in f,g]} \]

\[ = f \lor g \]

general case: quantification over several variables

\[ \exists x : BDD(y, f, g) = \begin{cases} 
BDD(y, \exists x : f, \exists x : g) & \text{if } y \notin x \\
(\exists x : f) \lor (\exists x : g) & \text{otherwise}
\end{cases} \]

universal quantification: similar

Complexity: worst case exponential, but usually works well in practice
BDDs: variable ordering

Variable ordering can drastically affect BDD sizes

example:

exponential growth in $n$

linear growth in $n$

determining optimal variable ordering is NP-hard

Heuristics

- manual ordering cluster dependent variables
- automatic strategies based on steepest-ascent or similar techniques
- some structures (e.g. multipliers, queues) do not admit compact BDD representation
Symbolic CTL model checking: implementation

Symbolic representation

- state space \( S \) vector of (Boolean) state variables \( x \)
- initial states \( I \) BDD over \( x \)
- transition relation \( \delta \) BDD over \( x, x', \) perhaps split conjunctively
- sets \( \{ \varphi \}_K \) BDDs over \( x \)

Operations

- set operations Boolean operations on BDDs
- pre-image \( \delta^{-1}(M) = \exists x' : \delta \land M' \)
- set comparison pointer comparison

Complexity can be exponential in size of BDD representing \( \delta \)

Results

- systems with huge potential state spaces \( (10^{120}) \) states) have been analysed
- particularly successful for synchronous hardware with short data paths
Abstraction techniques

Basics

Predicate Abstraction

Extensions for liveness
State explosion problem

Exponential increase of reachable states with system size

Partial solutions

- **reduce** partial-order, symmetry: explore only relevant part of state space
- **compress** unfoldings, BDDs: efficient data structures

**But:** \(10^{100}\) potential states are generated by just 300 bits

What about larger systems?

- **hardware** register files, execution pipelines
- **software** usually unbounded state size

Ad hoc approach

- **analyse small instances** 2 cache lines, 3 potential data values, etc.

How do you make sure that you’ll catch the bug?
Abstraction

Idea

- compute “abstract system” $\overline{\mathcal{K}}$ (finite, small)
- infer properties of $\mathcal{K}$ from properties of $\overline{\mathcal{K}}$

Issues

- how to obtain and present abstract model?
- full automation or user interaction?
- what if $\overline{\mathcal{K}} \not\models \varphi$ (“false negatives”)?

Predicate abstraction: abstraction determined by predicates over concrete state space

- predicates of interest indicated by the user
- subsumes other abstraction techniques
- intuitive presentation of abstract model
Example: dining mathematicians

mutual exclusion for two processes (synchronization via integer variable $n$)

```
int $n > 0$

loop
  $t_0 :$ await even($n$);
  $e_0 :$ $n := n \div 2$
endloop

parallel

loop
  $t_1 :$ await $\neg$even($n$);
  $e_1 :$ $n := 3 \times n + 1$
endloop
```

abstract representation: control state, parity

$\Box$($n > 0$)

$\Box \neg (at
e_0 \land at
e_1)$

$\Box F at
e_0$

$\Box F at
e_1$ can not be verified
Predicate diagrams

Fix set $AP$ of atomic propositions

$\overline{AP}$ denotes set of propositions in $AP$ and their negations

Presentation of abstraction as transition system $\overline{A} = (\overline{S}, \overline{l}, \overline{\delta})$

finite set $\overline{S} \subseteq 2^{\overline{AP}}$ of nodes (let $\overline{s} \in \overline{S}$ also denote conjunction of literals)

Verification conditions for correctness of abstraction

– **initialization:** initial nodes of $\overline{A}$ cover initial states of $K$

  \[
  \bigvee_{\overline{s} \in \overline{l}} \overline{s} \Rightarrow \bigvee_{s \in l} L(s)
  \]

– **consecution:** transitions of $\overline{A}$ cover possible transitions of $K$

  \[(\overline{s}, \overline{t}) \in \overline{\delta} \text{ if } L(s) \Rightarrow \overline{s} \text{ and } L(t) \Rightarrow \overline{t} \text{ for some } (s, t) \in \delta\]

Note: extra initial states or transitions preserves correctness
Preservation of properties

Correctness of abstraction implies:

– all computations of $\mathcal{K}$ represented as computations of $\overline{\mathcal{A}}$
– properties of $\mathcal{K}$ can be inferred from those of $\overline{\mathcal{A}}$

$$\overline{\mathcal{A}} \models \varphi \quad \iff \quad \mathcal{K} \models \varphi$$

for all LTL (actually, ACTL*) formulas $\varphi$ over $AP$
– $\overline{\mathcal{A}} \models \varphi$ established by model checking: consider atomic propositions as Boolean variables

$\overline{\mathcal{A}}$ may contain additional computations

– $\overline{\mathcal{A}} \not\models \varphi$ need not imply $\mathcal{K} \not\models \varphi$
– counter example often suggests how to improve the abstraction
– spurious loops invalidate liveness properties (cf. “dining mathematicians”)

Strengthening abstractions

– split nodes extend set $AP$ of atomic propositions
– break cycles represent information for liveness properties
Correct abstraction by elimination

- assume $\mathcal{K}$ being given by initial condition $Init$ and transition relation $Next$
- start with full graph over $2^{AP}$
- remove node $\bar{s}$ from $\bar{I}$ if $\models Init \Rightarrow \neg \bar{s}$
- remove edge $(\bar{s}, \bar{t})$ from $\bar{G}$ if $\models \bar{s} \land Next \Rightarrow \neg \bar{t}'$

Implementation: use theorem prover

- try to prove implications using automatic tactic with limited resources
  - many “local” goals instead of “global” property
  - unproven implications: approximation, perhaps good enough
  - drawback: $2^{|AP|}$ states, $2^{2^{|AP|}}$ proof attempts

Optimized implementation in PVS

H. Saïdi and N. Shankar: Abstract and model check while you prove. [CAV’99, LNCS 1633]
Generating predicate diagrams (2)

Compute abstraction by symbolic evaluation

- reduce: generate only reachable abstract states
- compilation approach: borrow from abstract interpretation

Formally: Galois connection

$$\alpha$$

sets of states $\xrightarrow{\gamma}$ Boolean algebra of predicates

Implementation

- rewrite $s \land \text{Next}$ into disjunction $t_1' \lor \ldots \lor t_n'$ of successor states
- sample rules for “dining mathematicians”

$$\text{even}(x), \text{even}(y) \Rightarrow \text{even}(x + y) \quad \text{even}(x), \neg\text{even}(y) \Rightarrow \neg\text{even}(x + y)$$

$$x \in \text{Nat}, x > 0, \text{even}(x) \Rightarrow x \text{ div } 2 > 0 \quad \text{even}(0) \quad \neg\text{even}(1)$$
Example: bakery algorithm

Lamport’s mutual-exclusion protocol  (2 processes, “atomic” version)

\[ \text{int } t_1 = 0, \ t_2 = 0 \quad (* \text{“queueing tickets” } *) \]

\begin{align*}
\text{loop} & \\
\quad l_1 : & \quad \text{“noncritical section”}; \\
\quad l_2 : & \quad t_1 := t_2 + 1; \\
\quad l_3 : & \quad \text{await } t_2 = 0 \lor t_1 \leq t_2; \\
\quad l_4 : & \quad \text{“critical section”}; \\
\quad l_5 : & \quad t_1 := 0 \\
\text{endloop} & \\
\end{align*}

\begin{align*}
\text{loop} & \\
\quad m_1 : & \quad \text{“noncritical section”}; \\
\quad m_2 : & \quad t_2 := t_1 + 1; \\
\quad m_3 : & \quad \text{await } t_1 = 0 \lor \neg (t_1 \leq t_2); \\
\quad m_4 : & \quad \text{“critical section”}; \\
\quad m_5 : & \quad t_2 := 0 \\
\text{endloop} & \\
\end{align*}

Note: ticket values can grow arbitrarily large

Predicates of interest

- control state
- \( t_1 = 0, \ t_2 = 0, \ t_1 \leq t_2 \)
Symbolic evaluation produces the following diagram (only control state indicated)

mutual exclusion \( G \neg(at\ l_4 \land at\ m_4) \)

response \( G(at\ l_3 \Rightarrow \neg F at\ l_4) \)

precedence
\( G(at\ l_3 \Rightarrow \neg at\ m_4 \ W at\ m_4 \ W \neg at\ m_4 \ W at\ l_4) \)

all properties verified from single diagram
Predicates on-the-fly

Symbolic evaluation can fail due to insufficient information

Bakery example: computing successors of

\[ \bar{n} \overset{\text{def}}{=} \{ \text{at } l_3, \text{ at } m_3, t_1 \neq 0, t_2 \neq 0 \} \]

fails because guard \( g \equiv t_1 \leq t_2 \) cannot be evaluated

Solution: reconsider predecessors of \( \bar{n} \)

– for every predecessor \( \bar{m} \) in the diagram, try to establish

\[ \bar{m} \land \text{Next} \land \bar{n}' \Rightarrow \left\{ \begin{array}{c} g \\ \neg g \end{array} \right\} \]

– add \((\neg)g\) to the node label of \( \bar{n} \) as appropriate
– possibly split node \( \bar{n} \)
Predicates on-the-fly: Bakery example

Predicate $t_1 \leq t_2$ need not be supplied by the user

inferred predicates added precisely where necessary
Strengthening for liveness

Boolean abstractions often cannot prove liveness properties

- predicate diagram usually contains cycles that do not correspond to “concrete” computations
- “dining mathematicians” example: liveness for process 1 could not be verified

Standard techniques to establish liveness properties

- fairness conditions  action taken infinitely often if sufficiently often enabled
- well-founded orderings  exclude cycles that correspond to infinite descent

These need to be represented in the abstraction!
Representing fairness conditions

Annotate (some) transitions in $\bar{\delta}$ with actions $A \in Act$

- formally, transitions are now triples $\bar{\delta} \subseteq \bar{S} \times Act \times \bar{S}$
- assume actions are described by characteristic predicate over $(x, x')$

Correctness conditions $(\bar{s}, A, \bar{t}) \in \bar{\delta}$ implies:

- enabledness: action $A$ is enabled at $\bar{s}$
  \[ \bar{s} \Rightarrow \exists x' : A \]

- effect: represent all possible $A$-successors
  \[ \bar{s} \land A \Rightarrow \bigvee_{(\bar{s},A,\bar{t}) \in \bar{\delta}} \bar{t}' \]
Model checking under fairness assumptions

Instrument abstract transition system \( \overline{A} \)

add Boolean variables \( en_A \) and \( taken_A \) for every action \( A \in Act \):

- enabledness \( en_A \) true at states that have outgoing edge \( (s, A, t) \in \delta \)
- execution \( taken_A \) true when previous transition may have been caused by \( A \)

Weaken property to prove

Deduce \( K \models \varphi \) from

\[
\overline{A} \models \bigwedge_{A \in Act} \left\{ WF(A), SF(A) \right\} \Rightarrow \varphi
\]

for actions \( A \in Act \) with weak (resp., strong) fairness assumption where

\[
WF(A) \overset{\text{def}}{=} FG en_A \Rightarrow GF taken_A
\]

\[
SF(A) \overset{\text{def}}{=} GF en_A \Rightarrow GF taken_A
\]
Representing well-founded orderings

Reconsider “dining mathematicians”

No computation of “concrete” system cycles between left-hand nodes

\[ n > 0, \text{even}(n) \]

\[ n > 0, \neg\text{even}(n) \]

\[ n > 0, \text{even}(n) \]

\[ n > 0, \neg\text{even}(n) \]

\[ n \] stays positive and even \ldots

\ldots but is infinitely often divided by 2

Note: every finite-state abstraction must contain similar cycle!
Ordering annotations

Represent descent w.r.t. well-founded ordering in $\overline{A}$

- let $t$ be (concrete-level) term and $\prec$ be well-founded ordering on domain of $t$
- label edge $(\overline{m}, A, \overline{n}) \in \overline{\delta}$ by $(t, \prec)$ (resp., $(t, \preceq)$) if
  \[
  \overline{m} \land A \land \overline{n}' \Rightarrow \begin{cases} 
  t' \prec t \\
  t' \preceq t 
  \end{cases}
  \]

Use edge annotations in model checking

- exclude computations of $\overline{A}$ that correspond to infinite descent of $t$ in $K$:
  Deduce $K \models \varphi$ from $\overline{A} \models (G F \neg \neg t' \prec t \Rightarrow G F \neg \neg t' \preceq t) \Rightarrow \varphi$
- “$t' \prec t$” represented by auxiliary Boolean variables
Dining mathematicians completed

Diagram annotated with ordering information

\[
\begin{align*}
\text{at } t_0, \text{ at } t_1 & \quad n > 0, \text{ even}(n) \\
\text{at } t_0, \text{ at } t_1 & \quad n > 0, \text{ even}(n) \\
\text{at } e_0, \text{ at } t_1 & \quad n > 0, \text{ even}(n) \\
\text{at } t_0, \text{ at } e_1 & \quad n > 0, \text{ even}(n) \\
\text{at } t_0, \text{ at } e_1 & \quad n > 0, \text{ even}(n)
\end{align*}
\]

\[
\begin{align*}
(n, \leq) & \quad (n, <)
\end{align*}
\]

Justification

- at \( t_0 \land \text{even}(n) \land \text{Next} \Rightarrow n' = n \\
- at \ e_0 \land \text{even}(n) \land n > 0 \land \text{Next} \Rightarrow n' = n \div 2

G(n > 0) \\
G \neg(\text{at } e_0 \land \text{at } e_1) \\
G \neg \text{at } e_0 \\
G \neg \text{at } e_1 \quad \text{can not be verified also}
Case study: self-stabilizing protocol

Dijkstra’s algorithm for self-stabilization

\[\text{var } v : \text{array } [0 .. N] \text{ of } [0 .. M]\]

\[v[0] = v[N] \quad \text{→} \quad v[0] := (v[0] + 1) \mod (M + 1)\]

\[\bigwedge_{i < N} \quad v[i + 1] \neq v[i] \quad \text{→} \quad v[i + 1] := v[i]\]

Stable configurations: precisely one process can move
Protocol formalized in Isabelle

types
  proc = nat
  vals = nat
  state = proc => vals
consts
  N, M :: nat
rules
  N_pos "0 < N"
  M_atleast_N "N ≤ M"
constdefs
  Proc :: proc set  
  Proc ≡ {i . i ≤ N}
  Vals :: vals set  
  Vals ≡ {i . i ≤ M}
act :: [proc, state, state] => bool  
  act i v w ≡
    (case i of
       0 => (v 0 = v N) ∧ (w 0 = (Suc (v 0)) mod (Suc M))
     | Suc j => (v j ≠ v (Suc j)) ∧ (w (Suc j) = v j))
       ∧ (∀ k. k ≠ i ⇒ (w k = v k))
enab :: [proc, state] => bool  
  enab i v ≡ ∃ w. act i v w
Idea of correctness

1. Once the ring has stabilized, it will remain stable.

2. Assume process 0’s value \( v[0] \) is different from all other register values.
   - Process 0 cannot move until \( v[0] \) appears in register \( N \).
   - \( v[0] \) will eventually spread along the ring.
   - When \( v[0] \) has reached register \( N \), the configuration is stable.

3. Some value \( k \leq M \) does not occur in any register \( v[i + 1] \).
   - Observe \( N \leq M \) and use pigeonhole principle.
   - This value cannot be introduced into the ring except by an action of process 0.
   - Because every move disables that process, process 0 must make infinitely many moves.
   - Therefore its register will eventually contain \( k \), and we have reached case 2.
Concepts for correctness formalized

prefix: initial ring segment with same register values

\[
\text{prefix} \quad :: \quad \text{state} \Rightarrow \text{proc set} \\
\text{prefix } v \equiv \{ \ i. \ i \in \text{Proc} \land (\forall j. \ j \leq i \Rightarrow v j = v 0) \ \}\]

othVals: values of registers outside the prefix

\[
\text{othVals} \quad :: \quad \text{state} \Rightarrow \text{vals set} \\
\text{othVals } v \equiv v``(\text{Proc} \setminus (\text{prefix } v))
\]

minfree: distance to least value not contained in othVals

\[
\text{minfree} \quad :: \quad \text{state} \Rightarrow \text{vals} \\
\text{minfree } v \equiv \text{LEAST } n. \ n \in \text{Vals} \\
\quad \land (v 0 + n) \mod (\text{Suc } M) \notin \text{othVals } v
\]

enabs: bit vector indicating enabledness of nonzero processes

ordered lexicographically
Predicate diagram for Dijkstra’s protocol

\( q_1 \)

\[ \exists i : i+1 \in \text{Proc} \land \text{enab}(i+1) \]

\( q_{11} \)

\( \minfree > 0 \)

\( q_{12} \)

\( \minfree = 0 \)

\( q_2 \)

\[ i \in \text{Proc} \land \text{act}(i) \]

\( q_3 \)

\[ \text{stable} \]

\( \neg \exists i : i+1 \in \text{Proc} \land \text{enab}(i+1) \)

\( F \ G \text{ stable} \quad \text{verifiable by model checking} \)
Summary

Semi-automatic construction of abstraction followed by model checking

Combination of model checking, theorem proving, and abstract interpretation

Challenge: integrate tools (SAL project at SRI, Stanford, Berkeley, Grenoble)

Identify useful abstractions that can be generated automatically:

- parameterized systems (Manna, Sipma ’99; Baukus, Lakhnech, Stahl ’00)

see also part of tutorial on infinite state spaces
Infinite State Spaces

Sources of infinity

Symbolic search: forward and backward

Accelerations and widenings
Sources of infinity

Data manipulation: unbounded counters, integer variables, lists . . .

Control structures: procedures → stack, process creation → bag

Asynchronous communication: unbounded FIFO queues

Parameters: number of processes, of input gates, of buffers, . . .

Real-time: discrete or dense domains
A bit of history

• Late 80s, early 90s: First theoretical papers
  
  Decidability/Undecidability results for Place/Transition Petri nets
  Efficient model-checking algorithms for context-free processes
  Region construction for timed automata

• 90s: Research program
  
  1. Decidability analysis
  2. Design of algorithms or semi-algorithms
  3. Design of implementations
  4. Tools
  5. Applications

• Late 90s, 00s: General techniques emerge
  
  Automata-theoretic approach to model-checking
  Symbolic reachability
  Accelerations and widenings
Parametrized protocols

Defined for $n$ processes.

Correctness: the desired properties hold for every $n$

Processes modelled as communicating finite automata

For each value of $n$ the system has a finite state space (only one source of infinity)

Turing powerful, and so further restrictions sensible:

Broadcast Protocols
Broadcast protocols

Introduced by Emerson and Namjoshi in LICS ’98

All processes execute the same algorithm, i.e., all finite automata are identical

Processes are undistinguishable (no IDs)

Communication mechanisms:

  **Rendezvous**: two processes exchange a message and move to new states

  **Broadcasts**: a process sends a message to all others

  all processes move to new states
Syntax

- `a!!`: broadcast a message along (channel) `a`
- `a??`: receive a broadcasted message along `a`
- `b!`: send a message to one process along `b`
- `b?`: receive a message from one process along `b`
- `c`: change state without communicating with anybody
Semantics

The global state of a broadcast protocol is completely determined by the number of processes in each state.

Configuration: mapping \( S \rightarrow \mathbb{N} \), seen as element of \( \mathbb{N}^n \), where \( n = |S| \)

Semantics for each \( n \): finite transition system

- configurations as nodes
- channel names as transition labels

In our example:

\[
\begin{align*}
(3, 1, 2) \xrightarrow{c} (4, 0, 2) \text{ (silent move)} \\
(3, 1, 2) \xrightarrow{b} (3, 2, 1) \text{ (rendezvous)} \\
(3, 1, 2) \xrightarrow{a} (2, 1, 3) \text{ (broadcast)}
\end{align*}
\]
Parametrized configuration: *partial* mapping \( p : Q \to \mathbb{N} \)

- Intuition: “configuration with holes”
- Formally: set of configurations (total mappings matching \( p \))

(Infinite) transition system of the broadcast protocol:

- Fix an initial parametrized configuration \( p_0 \).
- Take the union of all finite transition systems \( K_c \) for each configuration \( c \in p_0 \)
A MESI-protocol
The automata-theoretic approach

System $S \Rightarrow$ Kripke structure $\mathcal{K} \Rightarrow$ Languages $\mathcal{L}(\mathcal{K})$, $\mathcal{L}_\omega(\mathcal{K})$

of finite and infinite computations

If systems closed under product with automata then $B_{\neg \phi} \times \mathcal{K} \Rightarrow S_{\neg \phi}$

Safety and liveness problems reducible to

- **Reachability**
  
  Given: system $S$, sets $I$ and $F$ of initial and final configurations of $\mathcal{K}$
  
  To decide: if $F$ can be reached from $I$, i.e., if there exist $i \in I$ and $f \in F$ such that $i \rightarrow f$

- **Repeated reachability**
  
  Given: System $S$, sets $I$ and $F$ of initial and final configurations of $S$
  
  To decide: if $F$ can be repeatedly reached from $I$, i.e. if there exist $i \in I$ and $f_1, f_2, \ldots \in F$
  
  such that $i \rightarrow f_1 \rightarrow f_2 \ldots$

Shape of $I$ and $F$ depend on the class of atomic propositions
Repeated reachability is undecidable even for very simple sets $I$ and $F$

It is undecidable if there is a value of $n$ such that for this value the broadcast protocol has an infinite computation

Reachability is decidable for upward-closed sets $I$ and $F$

$U$ is an upward-closed set of configurations if

$$c \in U \text{ and } c' \geq c \text{ implies } c' \in U$$

where $\geq$ is the pointwise order on $\mathbb{N}^n$.

Safety property: upward-closed set $D$ of dangerous configurations

Example: in the MESI protocol the states $M$ and $S$ should be mutually exclusive

$$D = \{(m, e, s, i) \mid m \geq 1 \land s \geq 1\}$$
Symbolic search: forward and backward

Let $C$ denote a (possibly infinite) set of configurations

Forward search

$\text{post}(C) =$ immediate successors of $C$

Initialize $C := I$

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

$C \cap F \neq \emptyset$; return “reachable”, or

a fixpoint is reached; return “non-reachable”

Backward search

$\text{pre}(C) =$ immediate predecessors of $C$

Initialize $C := F$

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

$C \cap I \neq \emptyset$; return “reachable”, or

a fixpoint is reached; return “non-reachable”

Problem: when are the procedures effective?
Forward search effective if . . .

there is a family $C$ of sets such that

1. each $C \in C$ has a **symbolic** finite representation;
2. $I \in C$;
3. if $C \in C$, then $C \cup \text{post}(C) \in C$;
4. emptiness of $C \cap F$ is decidable;
5. $C_1 = C_2$ is decidable (to check if fixpoint has been reached); and
6. any chain $C_1 \subseteq C_2 \subseteq C_3 \ldots$ reaches a fixpoint after finitely many steps

(1)—(5) guarantee partial correctness, (6) guarantees termination

For backward search substitute $\text{post}(C)$ by $\text{pre}(C)$ and exchange $I$ and $F$

**Important difference:** backward search starts from $F$ instead from $I$; $I$ and $F$ may have different properties!
Forward search in broadcast protocols

$C$ must contain all parametrized configurations.

Satisfies (1)—(5) but not (6). Termination fails in very simple cases.

\[ (\sqcup, 0) \xrightarrow{a} (\sqcup, 1) \xrightarrow{a} (\sqcup, 2) \xrightarrow{a} \ldots \]
Backward search in broadcast protocols

[Abdulla, Cerãns, Jonsson, Tsay ’96], [E., Finkel, Mayr ’99]

The family of all upward-closed sets satisfies (1)—(6)

1. An upward-closed set can be represented by its set of minimal elements w.r.t. the pointwise order \( \leq \) (Dickson’s Lemma)

3. If \( U \) is upward-closed then so is \( U \cup \text{pre}(U) \).

\[
\begin{align*}
c & \xrightarrow{a} u \in U \\
\leq & \leq \\
c' & \xrightarrow{a} u' \in U
\end{align*}
\]

6. Any chain \( U_1 \subseteq U_2 \subseteq U_3 \ldots \) of upwards closed sets reaches a fixpoint after finitely many steps (Dickson’s lemma + some reasoning)
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 p-configuration

$$m = 0, e = 0, s = 0, i = \sqcup.$$ 

Proceed as follows:

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

$$U \cup \text{pre}(U): \quad (m \geq 1 \land s \geq 1) \lor$$

$$(m = 0 \land e = 1 \land s \geq 1)$$

$$U \cup \text{pre}(U) \cup \text{pre}^2(U): \quad U \cup \text{pre}(U)$$
Other models

FIFO-automata with lossy channels

[Abdulla and Jonsson ’93], [Abdulla, Bouajjani, Jonsson ’98]
Configuration: pair \((q, w)\), where \(q\) state and \(w\) vector of words representing the queue contents
Class \(C\): upward-closed sets with the subsequence order
Backward search satisfies (1)—(6)

Timed automata

[Alur and Dill ’94]
Configuration: pair \((q, x)\), where \(q\) state and \(x\) vector of real numbers
Class \(C\): regions
Forward search satisfies (1)—(6)
Accelerations and widenings: setup

\[ post[\sigma](C) = \text{set of configurations reached from } C \text{ by the sequence } \sigma \]

Compute a **symbolic reachability graph** with elements of \( C \) as nodes:

- Add \( I \) as first node
- For each node \( C \) and each label \( a \), add an edge \( C \xrightarrow{a} post[a](C) \)
Accelerations

Replace $C \xrightarrow{\sigma} \text{post}[\sigma](C)$ by $C \xrightarrow{\sigma} X$, where $X$ satisfies

1. $\text{post}[\sigma](C) \subseteq X$, and
2. $X$ contains only reachable configurations

Condition (1) guarantees the acceleration

Condition (2) guarantees that only reachable configurations are computed
Acceleration through loops

A loop is a sequence \( C \xrightarrow{\sigma} post[\sigma](C) \) such that

\[
C \xrightarrow{\sigma} post[\sigma](C) \xrightarrow{\sigma} post[\sigma^2](C) \xrightarrow{\sigma} post[\sigma^3](C) \ldots
\]

Syntactic loops (e.g. \( s \xrightarrow{a!} s \) in FIFO-systems)

Semantic loops defined through simulations: \( C_1 \) is simulated by \( C_2 \)
\[
\downarrow a \hspace{1cm} \downarrow a
\]
\( C'_1 \) is simulated by \( C'_2 \)

If \( post[\sigma](C) \) simulates \( C \), then \( C \xrightarrow{\sigma} post[\sigma](C) \) is a loop

Example: \( M \xrightarrow{\sigma} M \geq M \) in Petri nets
Acceleration: given a loop \( C \xrightarrow{\sigma} \text{post}[\sigma](C) \), replace \( \text{post}[\sigma](C) \) by

\[
X = \text{post}[\sigma^*](C) = C \cup \text{post}[\sigma](C) \cup \text{post}[\sigma^2](C) \cup \ldots
\]

**Problem:** find a class of loops such that \( \text{post}[\sigma^*](C) \) belongs to \( C \)
Class $C$: parametrized configurations

Class of loops: given by the following simulation

\[
\begin{align*}
\text{If } \sqsubseteq > n \text{ for all } n & \text{ then } \quad p_1 &\leq p_2 \\
\downarrow a &\quad \downarrow a \\
p_1' &\leq p_2'
\end{align*}
\]

So if $C \leq post[\sigma](C)$ then $post[\sigma](C)$ simulates $C$

$post[\sigma^*](\rho)$ may not be a parametrized configuration
Counter machines [Boigelot, Wolper 94]

Configuration: pair \((q, n_1, \ldots, n_k)\), where \(q\) state \(n_1, \ldots, n_k\) integers

Class \(C\): Presburger sets

Class of loops: syntactic

Pushdown automata [Bouajjani, E., Maler '97]

Configuration: pair \((q, w)\), where \(q\) state and \(w\) stack content

Class \(C\): regular sets

Class of loops: through semantic loops \((q, aw) \stackrel{\sigma}{\rightarrow} (q, aw'w)\)

Acceleration guarantees termination for both forward and backward search!
Other models II

FIFO-automata with lossy channels [Abdulla, Bouajjani, Jonsson ’98]

Configuration: pair \((q, w)\), where \(s\) state and \(w\) vector of words representing the contents of the queues

Class \(C\): regular sets represented by simple regular expressions

Class of loops: arbitrary

Other examples

FIFO-automata with perfect channels [Boigelot, Godefroid ], [Bouajjani, Habermehl ’98]
Arrays of parallel processes [Abdulla, Bouajjani, Jonsson, Nilsson ’99]
Widenings

Accurate widenings

Replace \( C \xrightarrow{\sigma} post[a](C) \) by \( C \xrightarrow{\sigma} X \), where \( X \) satisfies

\[
\begin{align*}
(1) & \quad post[a](C) \subseteq X, \text{ and} \\
(2') & \quad X \text{ contains only reachable final configurations}
\end{align*}
\]

Notice that \( X \) may contain unreachable non-final configurations!

Inaccurate widenings

Replace \( C \xrightarrow{\sigma} post[a](C) \) by \( C \xrightarrow{\sigma} X \), where \( X \) satisfies

\[
(1) \quad post[a](C) \subseteq X
\]

If no configuration of the graph belongs to \( F \), then no reachable configuration belongs to \( F \)
If some configuration of the graph belongs to \( F \), no information is gained
Accurate widenings in broadcast protocols

Fact: \( \text{post}[\sigma](p) = T_\sigma(p) \) for a linear transformation \( T_\sigma(p) = M_\sigma \cdot x + b_\sigma \)

It follows: \( \text{post}[\sigma^*](p) = \bigcup_{n \geq 0} T^*_\sigma(p) \)

Accurate widening: widen \( \text{post}[\sigma^*](p) \) to \( \text{lub}\{T^n_\sigma(p) \mid n \geq 0\} \)

Theorem: if the set \( F \) is upward closed, this widening is accurate
Does widening lead to termination?

For arbitrary broadcast protocols: NO  [E., Finkel, Mayr ’99]

Example in which the acceleration doesn’t have any effect:

\[
p_0 = (\sqcup, 0, 0)
\]

For rendezvous communication only: YES  
[Karp and Miller ’69], [German and Sistla ’92]
Implementing backwards reachability

Linear constraints as finite representation of sets of configurations.

The variable $x_i$ represents the number of processes in state $q_i$

Set of configurations $\rightarrow$ set of constraints over $x = \langle x_1, \ldots, x_n \rangle$

(interpreted disjunctively)

Immediate predecessors computed symbolically

Union and intersection $\rightarrow$ disjunction and conjunction

Containment test $\rightarrow$ entailment
Label $a$ \(\longrightarrow\) linear transformation with guard.

In our example

- Guard $G_a$: $x_1 \geq 1$

- Linear transformation $M_a x + b_a$:

\[
M_a = \begin{pmatrix}
0 & 0 & 1 \\
0 & 0 & 0 \\
1 & 1 & 0
\end{pmatrix}, \quad b_a = \begin{pmatrix}
0 \\
1 \\
-1
\end{pmatrix}
\]

Symbolic computation of $pre$ must satisfy

\[
pre(\Phi) \equiv \bigvee_{a \in \Sigma, \phi \in \Phi} G_a \land \phi[x / M_a x + b_a]
\]
Which class of constraints?

Able to express all upward-closed sets

Efficient computation of \( \text{pre} \)

Efficient entailment test

Entailment test co-NP-complete for arbitrary constraints
Natural candidates

\textbf{L-constraints}

Conjunction of inequations of shape $x_1 + \ldots + x_n \geq c$

Closed under broadcast transformations.

Entailment co-Np-complete even for single constraints

\textbf{WA-constraints}

Conjunction of inequations of shape $x_i \geq c$

Entailment is polynomial (quadratic)

Not closed under broadcast transformations.

L-constraints equivalent to sets of WA-constraints, but with exponential blow-up:

\[ x_{i_1} + \ldots + x_{i_m} \geq c \equiv \bigvee_{c_1 + \ldots + c_n = c} x_{i_1} \geq c_1 \land x_{i_2} \geq c_2 \land \ldots \land x_{i_m} \geq c_m \]
Using WA-constraints

[Delzanno and Raskin ’00]

Represent the constraint \( x_1 \geq c_1 \land \ldots \land x_n \geq c_n \) by \((c_1, \ldots, c_n)\)

Use sharing trees to represent sets of constraints

A sharing tree is an acyclic graph with one root and one terminal node such that

- all nodes of layer \( i \) have successors in the layer \( i + 1 \)
- a node cannot have two successors with the same label
- two nodes with the same label in the same layer do not have the same set of successors
A small Petri net experiment [Teruel ’98]
Deadlock-free states are the predecessors of an upward-closed set

Deadlock-free initial markings:

\[
\begin{align*}
P_1 & \geq 10, P_2 \geq 1, P_3 \geq 2 & P_1 & \geq 8, P_3 \geq 3 & P_1 & \geq 12, P_3 \geq 2 \\
P_1 & \geq 6, P_2 \geq 5, P_3 \geq 2 & P_1 & \geq 8, P_3 \geq 1, P_4 \geq 1 & P_1 & \geq 6, P_4 \geq 2 \\
P_1 & \geq 6, P_2 \geq 1, P_3 \geq 1, P_4 \geq 1
\end{align*}
\]

Computation time (Sun Ultra Sparc):

<table>
<thead>
<tr>
<th></th>
<th>Sharing trees</th>
<th>HyTech</th>
<th>Presburger</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>39s</td>
<td>&gt; 24h</td>
<td>19h50m</td>
</tr>
</tbody>
</table>
Using $L$-constraints

[Delzanno, E., Podelski ’99], [Delzanno ’00]

First simplification: entailment need only be computed for single constraints

Replace

\[
\text{until } \text{Entail}(\Phi, \text{old}_\Phi)
\]

by the stronger condition

\[
\text{until } \forall \phi \in \Phi \exists \psi \in \text{old}_\Phi : \text{Entail}(\phi, \psi)
\]

Possibly slower, but still guaranteed termination

But entailment for $L$-constraints co-Np-complete even for single constraints!

Second simplification: interpret entailment over the reals

Again, stronger until-condition which does not spoil termination
Case studies (by G. Delzanno)

Broadcast protocols must be extended with more complicated guards.

Termination guarantee gets lost

Berkeley RISC, Illinois, Xerox PARC Dragon, DEC Firefly

At most 7 iterations and below 100 seconds (SPARC5, Pentium 133)

Futurebus +

8 steps and 200 seconds (Pentium 133)
Conclusions

Decidability analysis very advanced

Many algorithms useful in practice

In the next years: improve implementations, integrate in tools.

Challenge: several sources of infinity.