Introduction to Model Checking

Radu Mateescu
Inria – Univ. Grenoble Alpes – LIG
What is model checking?

“Model checking is the method by which a desired behavioral property of a reactive system is verified over a given system (the model) through exhaustive enumeration (explicit or implicit) of all the states reachable by the system and the behaviors that traverse through them.”

Amir Pnueli

Foreword to Model Checking
[Clarke-Grumberg-Peled-00]
Basic model checking flow

- **system description**
- **property**
- **model (state space)**
- **intermediate form**
- **model checker**
- **encoding and resolution of the verification problem**
- **verdict & diagnostic**

*Model Checking - FMF, LAAS, 16/10/2014*
Running example
(action-based version)

Two-cell buffer with unreliable transmission

9 states, 20 transitions

PUT 0/1  GET 0/1

action-based setting (Labelled Transition System)
Running example
(state-based version)

Keep the contents of states and the transitions between them

state-based setting
(Kripke structure)
States vs actions

State-based
- White box spec style
- Predicates on state variables
- Stuttering equivalence
- Partial order reductions

Action-based
- Black box spec style
- Predicates on actions/events
- Weak bisimulations
- Compositionality (congruences w.r.t. $||$)

Kripke transition systems (KTS)
state variables and actions
Specification of temporal properties

Temporal logic [Pnueli-77]:

formalism for describing evolutions of program states over (logical) time

– Atomic propositions over states
– Propositional logic operators (or, and, not, …)
– Tense operators (neXt, Until, Previous, Since, Once, …)
– Interpreted on state spaces

High-level specification style:

abstraction and modularity
Properties on states and branches

(CTL – Computation Tree Logic)

- $\mathbf{X} \varphi, \mathbf{E} [\varphi_1 \mathbf{U} \varphi_2], \mathbf{A} [\varphi_1 \mathbf{U} \varphi_2]$
- $\mathbf{EF} \varphi = \mathbf{E} [\mathbf{true} \mathbf{U} \varphi]$
  (potentiality)
- $\mathbf{AG} \varphi = \neg \mathbf{EF} \neg \varphi$
  (invariance)
- $\mathbf{AF} \varphi = \mathbf{A} [\mathbf{true} \mathbf{U} \varphi]$
  (inevitability)
- $\mathbf{EG} \varphi = \neg \mathbf{AF} \neg \varphi$
  (trajectory)

- $\mathbf{AG} (s_0^* \Rightarrow \mathbf{EF} s^*0)$ **ok**
- $\mathbf{AG} (s_0^* \Rightarrow \mathbf{AF} s^*0)$ **ko**
Properties on states and paths

(LTL – Linear Temporal Logic)

\[ X \psi, \psi_1 U \psi_2 \]
\[ F \psi = \text{true} U \psi \] 
(eventually)
\[ G \psi = \neg F \neg \psi \] 
(globally)
\[ \psi_1 R \psi_2 = \neg (\neg \psi_1 U \neg \psi_2) \] 
(release)

\[ GF (s0_0 V s1_0 V s_0 V s_1) \quad \text{ok} \]
\[ FG s_0 V s_1 \quad \text{ko} \]
The two logics are incomparable.
Linear-time vs branching-time
Properties on actions

(ACTL – Action-based CTL)

AG\textsubscript{true} [PUT\textsubscript{0}]
E [true\textsubscript{true} U\textsubscript{GET0} true]
ok

AG\textsubscript{true} [PUT\textsubscript{0}]
A [true\textsubscript{true} U\textsubscript{GET0} true]
k0
Properties on actions

(L\(\mu\) – modal \(\mu\)-calculus)

“Assembly language” for temporal operators

– Modalities and fixed point operators
– Hierarchy of fragments \(L\mu_k\) with alternation depth \(k\)
– Captures virtually all existing TL operators

\[ E [\varphi_1 U \varphi_2] = \mu X . \varphi_2 V (\varphi_1 \land < \text{true} > X) \]  \hspace{1cm} \text{(CTL)}

\[ AFG \varphi = \neg \nu X . \mu Y . (\neg \varphi \land X) V < \text{true} > Y \]  \hspace{1cm} \text{(LTL)}
State-based vs action-based

F-LTL, LTL, μ-ACTL, µ-CTL, F, L, μ,CTL, LTS, LTL, ACTL, ACTL*, CTL, HML, HMLR

branching-time
Extensions with regular features

Regular expressions / automata

– Natural description of regular paths

Safety: FIFO buffer policy

\[
\text{true}^* \cdot \text{PUT}_0 \cdot (\neg \text{GET})^* \cdot \text{PUT}_1 \cdot (\neg \text{PUT})^* \cdot \text{GET}_1 \cdot (\neg \text{PUT})^* \cdot \text{GET}_0 \] \text{false} \\
(PDL)

\forall X . ([\text{PUT}_0] \forall Y . (([\text{PUT}_1] \forall Z . (([\text{GET}_1] \forall W . ([\text{GET}_0] \text{false} \land (\neg \text{PUT}) W) \land (\neg \text{PUT}) Z) \land (\neg \text{GET}_0) Y) \land \text{true} X)

(Lμ₁)
Extensions with data

- Handling of data values present in states/actions

**Safety**: capacity of (reliable) 2-buffer

\[ \text{true}^*. (\text{PUT} \ . \ (\neg \text{GET})^*) \{3\} \] \text{false}

- Parametric formulas (stable w.r.t. model)

**Response**: fair reachability of message delivery

\[ \text{true}^*. \{\text{PUT} \ ?m:nat\} \ < \text{true}^*. \{\text{GET} \ !m\} \ > \text{true} \]
Ergonomic extensions
(regular constructs and data handling)
Expressiveness and complexity

Lμ₁
|ϕ|·|M|

Lμ₂
|ϕ|²·|M|²

PDL
|ϕ|·|M|

PDL-Δ
|ϕ|·|M|

CTL
|ϕ|·|M|

CTL*
2|ϕ|·|M|

LTL
2|ϕ|·|M|

FMM - Model Checking - LAAS, 16/10/2014
Quantitative properties

Time
(TA, TPN)

Rates
(CTMC, MDP)

Probabilities
(DTMC)
E<> s_1 && (c == 1)
Temporal logic zoo

- PCTL
- EAGLE
- MITL
- FOLμ
- LTL
- F-LTL
- νTL
- XTL
- RICO
- ETL
- μ-ACTL
- μ-ACTL
- ECTL*
- PSL
- BRTL
- RegCTL
- PDL
- PDL-Δ
- TCTL
- CSL
- Sugar
- ACTL
- HML
- HMLR
- MCL
- EAGLE
- ECTL*
- CTL*
- CTL
- LTL
- νCTL
- νTL
- P unary
- Lμ
- Hμ
- Tμ
How to choose the right TL?

- Nature of the system and its properties:
  - linear / branching
  - functional / quantitative
  - state / action
  - discrete / continuous

- Expressiveness vs model checking complexity
  - Tradeoff is often made in the available tools

- User-friendliness
  - Built-in ergonomic extensions (regexps, data)
  - Tools often provide libraries of derived operators
  - Use of property pattern libraries [Dwyer-et-al-99]
State space explosion

Exponential growth of the state space with the number of parallel processes

Model checking holy grail: (endless?) fight against state space explosion
On-the-fly model checking
(linear-time, state-based – LTL/SPIN)

Promela program

Compilation

implicit KS

Partial order reduction

synchronous product

product BA
\[ L(KS \times A_{\neg \varphi}) = L(KS) \cap L(A_{\neg \varphi}) \]

emptiness check

verdict & counterexample (lasso)

see the BA zoo at www.spot.lip6.fr

LTL formula (\(\varphi\))

negation and translation

Büchi automaton \(A_{\neg \varphi}\)
On-the-fly model checking
(branching-time, action-based – MCL/CADP/Evaluator)

- LNT specification
  - compilation
    - implicit LTS
      - Caesar_Solve
        - Open/Caesar environment
      - parameterized HMLR
        - encoding
        - optimisation
        - On-the-fly activities
    - parameterized BES
      - instantiation & resolution
      - translation
        - MCL formula
      - verdict & diagnostic
Symbolic model checking
(branching-time, state-based logics – CTL/nuSMV)

- formal description
  - compilation
  - symbolic fixed point iteration
    - dynamic variable reordering
    - fairness constraint handling
    - L_\mu encoding (predicate transformer)

- verdict & diagnostic

- symbolic KS (BDD)
  - L_\mu encoding (predicate transformer)
Other ways to fight state explosion

- Bounded model checking
  - Symbolic partial exploration, use of SAT/SMT solvers

- Parallel and distributed model checking
  - Explicit / symbolic, linear / branching

- Compositional verification
  - Assume-guarantee / partial model checking

- Runtime verification
  - TL formulas $\rightarrow$ monitors $\rightarrow$ check execution traces

- Statistical model checking
Model checkers landscape
(partial view)

- SPIN (explicit/parallel)
- SPOT (explicit/symbolic)
- DIVINE (explicit/distributed)
- LTSmin (explicit/distributed)

- nuSMV (symbolic)
- TLA+ (symbolic)
- LTSA (explicit)
- JACK (explicit/symbolic)

- LTSmin
- TLTA
- F-LTL
- μ-ACTL

- UPPAAL (symbolic)
- TINA (symbolic)
- PRISM (explicit/symbolic)
- MRMC (explicit/symbolic)
- MODEST (explicit/symbolic)
- TINA (symbolic)
- PRISM (explicit/symbolic)
- MRMC (explicit/symbolic)
- MODEST (explicit/symbolic)

- Timed CTL
- Timed LTL
- PCTL
- CSL
- MCL
Model checking in the design process

- Choose the right modeling language and TL
- Model the *essential* aspects of the system
- Start with on-the-fly (parallel) verification:
  - Fast detection of errors
  - Debug based on counterexamples
- When no more errors found / no memory left:
  - Use symbolic / compositional / distributed verification
  - Use abstraction whenever possible
What to do next?

- Regular increase of model checking capabilities
  - Bounded model checking, SAT/SMT techniques
- Several stable tools (and many others!)
  - Industrial success stories for each method / tool
- Model checking interoperates with other techniques
  (static analysis, theorem proving, ...)
- Ideally, one should be able to apply smoothly several verification techniques on the same system description
  ➔ need for languages / models / tools interoperability
Some references

[Schnoebelen-et-al-99] *Vérification de logiciels*

[Clarke-Grumberg-Peled-00] *Model Checking*

[Baier-Katoen-08] *Principles of Model Checking*

+ many articles on the various model checkers
Thank you