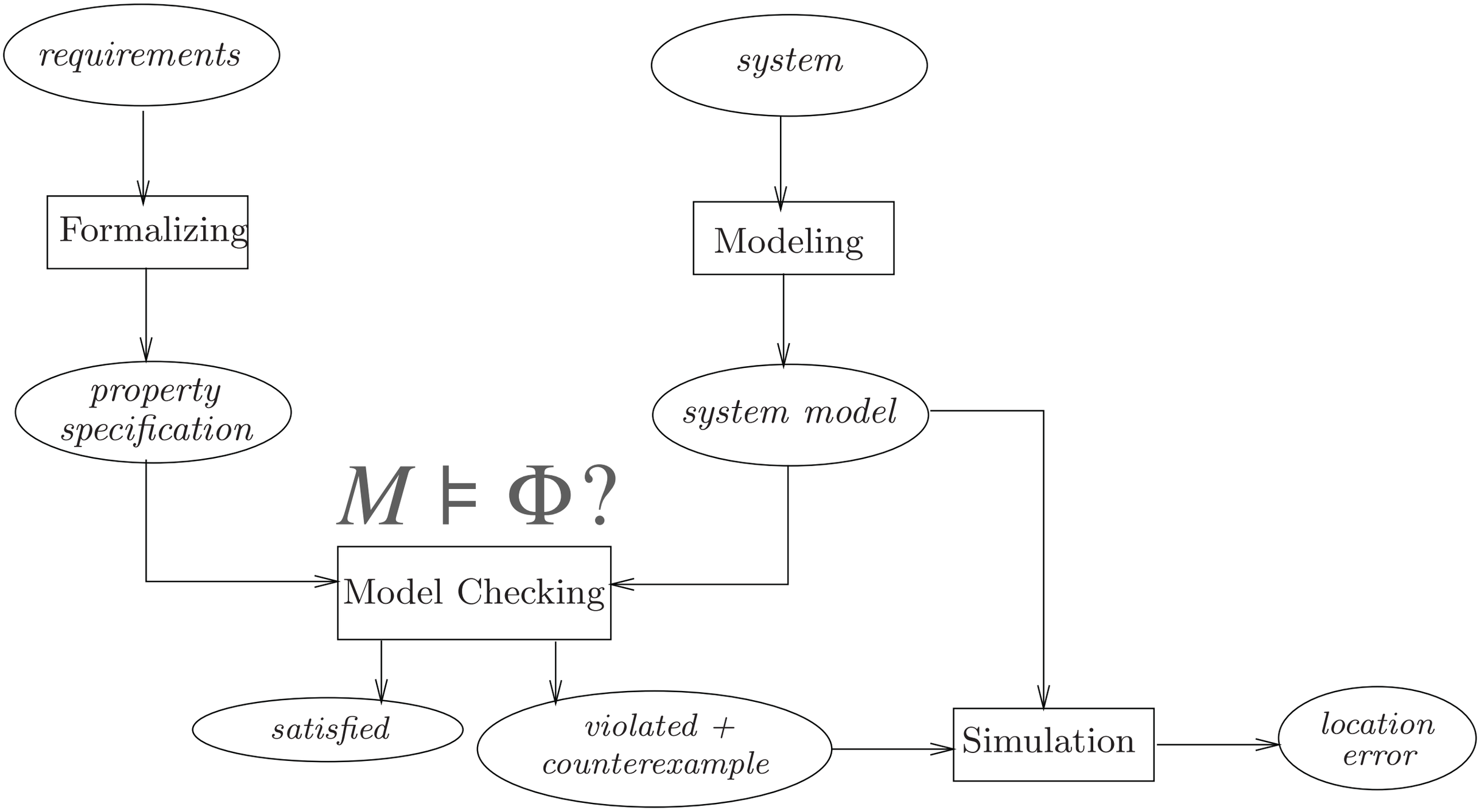
Model Checking

$$M \models \Phi ?$$

Introduction

adapted from slides by Alberto Lluch Lafuente, Technical University of Denmark, Fall 2020
based on material from the book “Principles of Model Checking” by Christel Baier and Joost-Pieter Katoen

What is Model Checking?



Model Checking process

$$M \models \Phi ?$$

Model Checking process

$$M \models \Phi ?$$

- *Modelling* phase:
  - model the system under consideration using the model description language of the model checker at hand;
  - as a first sanity check and quick assessment of the model perform some simulations;
  - formalise the property to be checked using the property specification language.

Model Checking process

$$M \models \Phi ?$$

- *Modelling* phase:
  - model the system under consideration using the model description language of the model checker at hand;
  - as a first sanity check and quick assessment of the model perform some simulations;
  - formalise the property to be checked using the property specification language.

- *Running* phase:
  - run the model checker to check the validity of the property in the system model.

Model Checking process

$$M \models \Phi ?$$

- *Modelling* phase:
  - model the system under consideration using the model description language of the model checker at hand;
  - as a first sanity check and quick assessment of the model perform some simulations;
  - formalise the property to be checked using the property specification language.

- *Running* phase:
  - run the model checker to check the validity of the property in the system model.

- *Analysis* phase:
  - property satisfied? → check next property (if any);
  - property violated? →
    1. analyse generated counterexample by simulation;
    2. refine the model, design or property;
    3. repeat the entire procedure.
  - out of memory? → try to reduce the model and try again.

System

refine/implement

Requirements

abstract model

$M \models \Phi$?

some logic here

Finite-state transition System

some smart algorithm here

yes ☺

may be 😐

no ☹

Strengths and Weaknesses

Strengths and Weaknesses

- *General* verification approach, applicable to a wide range of applications (e.g. embedded systems, software engineering, hardware design);

Strengths and Weaknesses

- *General* verification approach, applicable to a wide range of applications (e.g. embedded systems, software engineering, hardware design);

Strengths and Weaknesses

- *General* verification approach, applicable to a wide range of applications (e.g. embedded systems, software engineering, hardware design);
- Supports *partial* verification, i.e. properties can be checked individually, allowing to focus on essential properties first (no complete requirement specification needed);

Strengths and Weaknesses

- *General* verification approach, applicable to a wide range of applications (e.g. embedded systems, software engineering, hardware design);
- Supports *partial* verification, i.e. properties can be checked individually, allowing to focus on essential properties first (no complete requirement specification needed);

Strengths and Weaknesses

- *General* verification approach, applicable to a wide range of applications (e.g. embedded systems, software engineering, hardware design);
- Supports *partial* verification, i.e. properties can be checked individually, allowing to focus on essential properties first (no complete requirement specification needed);
- Not vulnerable to likelihood that an error is exposed (contrary to testing, simulation);

Strengths and Weaknesses

- *General* verification approach, applicable to a wide range of applications (e.g. embedded systems, software engineering, hardware design);
- Supports *partial* verification, i.e. properties can be checked individually, allowing to focus on essential properties first (no complete requirement specification needed);
- Not vulnerable to likelihood that an error is exposed (contrary to testing, simulation);

Strengths and Weaknesses

- *General* verification approach, applicable to a wide range of applications (e.g. embedded systems, software engineering, hardware design);
- Supports *partial* verification, i.e. properties can be checked individually, allowing to focus on essential properties first (no complete requirement specification needed);
- Not vulnerable to likelihood that an error is exposed (contrary to testing, simulation);
- Provides *diagnostic info* in case a property is invalidated: very useful for debugging;

Strengths and Weaknesses

- *General* verification approach, applicable to a wide range of applications (e.g. embedded systems, software engineering, hardware design);
- Supports *partial* verification, i.e. properties can be checked individually, allowing to focus on essential properties first (no complete requirement specification needed);
- Not vulnerable to likelihood that an error is exposed (contrary to testing, simulation);
- Provides *diagnostic info* in case a property is invalidated: very useful for debugging;

Strengths and Weaknesses

- *General* verification approach, applicable to a wide range of applications (e.g. embedded systems, software engineering, hardware design);
- Supports *partial* verification, i.e. properties can be checked individually, allowing to focus on essential properties first (no complete requirement specification needed);
- Not vulnerable to likelihood that an error is exposed (contrary to testing, simulation);
- Provides *diagnostic info* in case a property is invalidated: very useful for debugging;
- **Potentially a "push-button" technology**, requiring neither a high degree of user interaction nor a high degree of expertise;

Strengths and Weaknesses

- *General* verification approach, applicable to a wide range of applications (e.g. embedded systems, software engineering, hardware design);
- Supports *partial* verification, i.e. properties can be checked individually, allowing to focus on essential properties first (no complete requirement specification needed);
- Not vulnerable to likelihood that an error is exposed (contrary to testing, simulation);
- Provides *diagnostic info* in case a property is invalidated: very useful for debugging;
- **Potentially a "push-button" technology**, requiring neither a high degree of user interaction nor a high degree of expertise;

Strengths and Weaknesses

- *General* verification approach, applicable to a wide range of applications (e.g. embedded systems, software engineering, hardware design);
- Supports *partial* verification, i.e. properties can be checked individually, allowing to focus on essential properties first (no complete requirement specification needed);
- Not vulnerable to likelihood that an error is exposed (contrary to testing, simulation);
- Provides *diagnostic info* in case a property is invalidated: very useful for debugging;
- **Potentially a "push-button" technology**, requiring neither a high degree of user interaction nor a high degree of expertise;
- Enjoys a rapidly increasing *interest by industry*: several companies have started their in-house verification labs (e.g. Amazon), frequent job offers with required skills in model checking, and commercial model checkers have become available;

Strengths and Weaknesses

- *General* verification approach, applicable to a wide range of applications (e.g. embedded systems, software engineering, hardware design);
- Supports *partial* verification, i.e. properties can be checked individually, allowing to focus on essential properties first (no complete requirement specification needed);
- Not vulnerable to likelihood that an error is exposed (contrary to testing, simulation);
- Provides *diagnostic info* in case a property is invalidated: very useful for debugging;
- **Potentially a "push-button" technology**, requiring neither a high degree of user interaction nor a high degree of expertise;
- Enjoys a rapidly increasing *interest by industry*: several companies have started their in-house verification labs (e.g. Amazon), frequent job offers with required skills in model checking, and commercial model checkers have become available;

Strengths and Weaknesses

- *General* verification approach, applicable to a wide range of applications (e.g. embedded systems, software engineering, hardware design);
- Supports *partial* verification, i.e. properties can be checked individually, allowing to focus on essential properties first (no complete requirement specification needed);
- Not vulnerable to likelihood that an error is exposed (contrary to testing, simulation);
- Provides *diagnostic info* in case a property is invalidated: very useful for debugging;
- **Potentially a "push-button" technology**, requiring neither a high degree of user interaction nor a high degree of expertise;
- Enjoys a rapidly increasing *interest by industry*: several companies have started their in-house verification labs (e.g. Amazon), frequent job offers with required skills in model checking, and commercial model checkers have become available;
- Easy *integration* in existing development cycles: its learning curve is not very steep, and empirical studies indicate that it may lead to shorter development times;

Strengths and Weaknesses

- *General* verification approach, applicable to a wide range of applications (e.g. embedded systems, software engineering, hardware design);
- Supports *partial* verification, i.e. properties can be checked individually, allowing to focus on essential properties first (no complete requirement specification needed);
- Not vulnerable to likelihood that an error is exposed (contrary to testing, simulation);
- Provides *diagnostic info* in case a property is invalidated: very useful for debugging;
- **Potentially a "push-button" technology**, requiring neither a high degree of user interaction nor a high degree of expertise;
- Enjoys a rapidly increasing *interest by industry*: several companies have started their in-house verification labs (e.g. Amazon), frequent job offers with required skills in model checking, and commercial model checkers have become available;
- Easy *integration* in existing development cycles: its learning curve is not very steep, and empirical studies indicate that it may lead to shorter development times;

Strengths and Weaknesses

- *General* verification approach, applicable to a wide range of applications (e.g. embedded systems, software engineering, hardware design);
- Supports *partial* verification, i.e. properties can be checked individually, allowing to focus on essential properties first (no complete requirement specification needed);
- Not vulnerable to likelihood that an error is exposed (contrary to testing, simulation);
- Provides *diagnostic info* in case a property is invalidated: very useful for debugging;
- **Potentially a "push-button" technology**, requiring neither a high degree of user interaction nor a high degree of expertise;
- Enjoys a rapidly increasing *interest by industry*: several companies have started their in-house verification labs (e.g. Amazon), frequent job offers with required skills in model checking, and commercial model checkers have become available;
- Easy *integration* in existing development cycles: its learning curve is not very steep, and empirical studies indicate that it may lead to shorter development times;
- *Sound and mathematical underpinning*, based on theory of graph algorithms, data structures and logic.

Strengths and *Weaknesses*

Strengths and *Weaknesses*

- Mainly appropriate to *control-intensive* applications and less suited for data-intensive applications as data typically ranges over infinite domains;

Strengths and *Weaknesses*

- Mainly appropriate to *control-intensive* applications and less suited for data-intensive applications as data typically ranges over infinite domains;

Strengths and *Weaknesses*

- Mainly appropriate to *control-intensive* applications and less suited for data-intensive applications as data typically ranges over infinite domains;

- Applicability subject to *decidability* issues: for infinite-state systems or reasoning about abstract data types (which requires undecidable or semi-decidable logics), model checking is in general not effectively computable;

Strengths and *Weaknesses*

- Mainly appropriate to *control-intensive* applications and less suited for data-intensive applications as data typically ranges over infinite domains;

- Applicability subject to *decidability* issues: for infinite-state systems or reasoning about abstract data types (which requires undecidable or semi-decidable logics), model checking is in general not effectively computable;

Strengths and *Weaknesses*

- Mainly appropriate to *control-intensive* applications and less suited for data-intensive applications as data typically ranges over infinite domains;

- Applicability subject to *decidability* issues: for infinite-state systems or reasoning about abstract data types (which requires undecidable or semi-decidable logics), model checking is in general not effectively computable;

- One verifies a *system model*, and not the actual system (product or prototype) itself: **any obtained result is thus as good as the system model** (need complementary techniques, like testing, to find fabrication faults (for HW) or coding errors (for SW);

Strengths and *Weaknesses*

- Mainly appropriate to *control-intensive* applications and less suited for data-intensive applications as data typically ranges over infinite domains;

- Applicability subject to *decidability* issues: for infinite-state systems or reasoning about abstract data types (which requires undecidable or semi-decidable logics), model checking is in general not effectively computable;

- One verifies a *system model*, and not the actual system (product or prototype) itself: **any obtained result is thus as good as the system model** (need complementary techniques, like testing, to find fabrication faults (for HW) or coding errors (for SW);

Strengths and *Weaknesses*

- Mainly appropriate to *control-intensive* applications and less suited for data-intensive applications as data typically ranges over infinite domains;

- Applicability subject to *decidability* issues: for infinite-state systems or reasoning about abstract data types (which requires undecidable or semi-decidable logics), model checking is in general not effectively computable;

- One verifies a *system model*, and not the actual system (product or prototype) itself: **any obtained result is thus as good as the system model** (need complementary techniques, like testing, to find fabrication faults (for HW) or coding errors (for SW);

- Checks only *requirements actually stated*, i.e. **no guarantee of completeness**, since the validity of properties that are not checked cannot be judged;

Strengths and *Weaknesses*

- Mainly appropriate to *control-intensive* applications and less suited for data-intensive applications as data typically ranges over infinite domains;

- Applicability subject to *decidability* issues: for infinite-state systems or reasoning about abstract data types (which requires undecidable or semi-decidable logics), model checking is in general not effectively computable;

- One verifies a *system model*, and not the actual system (product or prototype) itself: **any obtained result is thus as good as the system model** (need complementary techniques, like testing, to find fabrication faults (for HW) or coding errors (for SW);

- Checks only *requirements actually stated*, i.e. **no guarantee of completeness**, since the validity of properties that are not checked cannot be judged;

Strengths and *Weaknesses*

- Mainly appropriate to *control-intensive* applications and less suited for data-intensive applications as data typically ranges over infinite domains;

- Applicability subject to *decidability* issues: for infinite-state systems or reasoning about abstract data types (which requires undecidable or semi-decidable logics), model checking is in general not effectively computable;

- One verifies a *system model*, and not the actual system (product or prototype) itself: **any obtained result is thus as good as the system model** (need complementary techniques, like testing, to find fabrication faults (for HW) or coding errors (for SW);

- Checks only *requirements actually stated*, i.e. **no guarantee of completeness**, since the validity of properties that are not checked cannot be judged;

- Suffers from *state space explosion* problem: the number of states needed to model the system accurately may easily exceed the amount of available computer memory (despite the development of several very effective methods to combat this problem, models of realistic systems may still be too large to fit in memory);

Strengths and *Weaknesses*

- Mainly appropriate to *control-intensive* applications and less suited for data-intensive applications as data typically ranges over infinite domains;

- Applicability subject to *decidability* issues: for infinite-state systems or reasoning about abstract data types (which requires undecidable or semi-decidable logics), model checking is in general not effectively computable;

- One verifies a *system model*, and not the actual system (product or prototype) itself: **any obtained result is thus as good as the system model** (need complementary techniques, like testing, to find fabrication faults (for HW) or coding errors (for SW);

- Checks only *requirements actually stated*, i.e. **no guarantee of completeness**, since the validity of properties that are not checked cannot be judged;

- Suffers from *state space explosion* problem: the number of states needed to model the system accurately may easily exceed the amount of available computer memory (despite the development of several very effective methods to combat this problem, models of realistic systems may still be too large to fit in memory);

Strengths and *Weaknesses*

- Mainly appropriate to *control-intensive* applications and less suited for data-intensive applications as data typically ranges over infinite domains;

- Applicability subject to *decidability* issues: for infinite-state systems or reasoning about abstract data types (which requires undecidable or semi-decidable logics), model checking is in general not effectively computable;

- One verifies a *system model*, and not the actual system (product or prototype) itself: **any obtained result is thus as good as the system model** (need complementary techniques, like testing, to find fabrication faults (for HW) or coding errors (for SW);

- Checks only *requirements actually stated*, i.e. **no guarantee of completeness**, since the validity of properties that are not checked cannot be judged;

- Suffers from *state space explosion* problem: the number of states needed to model the system accurately may easily exceed the amount of available computer memory (despite the development of several very effective methods to combat this problem, models of realistic systems may still be too large to fit in memory);

- Its usage requires some *expertise* in finding appropriate abstractions to obtain smaller system models and to state properties in the logical formalism used;

Strengths and *Weaknesses*

- Mainly appropriate to *control-intensive* applications and less suited for data-intensive applications as data typically ranges over infinite domains;

- Applicability subject to *decidability* issues: for infinite-state systems or reasoning about abstract data types (which requires undecidable or semi-decidable logics), model checking is in general not effectively computable;

- One verifies a *system model*, and not the actual system (product or prototype) itself: **any obtained result is thus as good as the system model** (need complementary techniques, like testing, to find fabrication faults (for HW) or coding errors (for SW);

- Checks only *requirements actually stated*, i.e. **no guarantee of completeness**, since the validity of properties that are not checked cannot be judged;

- Suffers from *state space explosion* problem: the number of states needed to model the system accurately may easily exceed the amount of available computer memory (despite the development of several very effective methods to combat this problem, models of realistic systems may still be too large to fit in memory);

- Its usage requires some *expertise* in finding appropriate abstractions to obtain smaller system models and to state properties in the logical formalism used;

Strengths and *Weaknesses*

- Mainly appropriate to *control-intensive* applications and less suited for data-intensive applications as data typically ranges over infinite domains;

- Applicability subject to *decidability* issues: for infinite-state systems or reasoning about abstract data types (which requires undecidable or semi-decidable logics), model checking is in general not effectively computable;

- One verifies a *system model*, and not the actual system (product or prototype) itself: **any obtained result is thus as good as the system model** (need complementary techniques, like testing, to find fabrication faults (for HW) or coding errors (for SW);

- Checks only *requirements actually stated*, i.e. **no guarantee of completeness**, since the validity of properties that are not checked cannot be judged;

- Suffers from *state space explosion* problem: the number of states needed to model the system accurately may easily exceed the amount of available computer memory (despite the development of several very effective methods to combat this problem, models of realistic systems may still be too large to fit in memory);

- Its usage requires some *expertise* in finding appropriate abstractions to obtain smaller system models and to state properties in the logical formalism used;

- Correct results not guaranteed: as any other tool, it may contain *software defects*;