Verificação Formal de Software

Nelma Moreira

Departamento de Ciência de Computadores da FCUP

Verificação Formal de Software
Aula 1

Software and cathedrals are much the same.
First we build them, then we
pray.

Confiabilidade
Segurança
Correção
Robustez


FM'19
3rd world congress on formal methods
FM Week
FM 2019
23rd Symposium on Formal Methods
LOPSTR 2019
29th International Symposium on Logic-Based Program Synthesis and Transformation
MPC 2019
13th International Conference on Mathematics of Program Construction
PPDP 2019
21st International Symposium on Principles and Practice of Declarative Programming
RV 2019
19th International Conference on Runtime Verification
SAS 2019
26th International Static Analysis Symposium
TAP 2019
13th International Conference on Tests and Proofs
UTP 2019
7th International Symposium on Unifying Theories of Programming
VECoS 2019
13th International Conference on Verification and Evaluation of Computer and Communication Systems
Doctoral Symposium
Industry Day
and several Workshops and Tutorials!
7 — 11 october 2019
Alfandega Porto Congress Centre Porto, Portugal
fm2019.inesctec.pt
formalmethods19
www.fmeurope.org

Uso de métodos matemáticos para modelação, cálculo e predição na especificação, desenho e análise de sistemas informáticos

URL:`www.dcc.fc.up.pt/nam/web/Teaching/vfs19/index.html`

Método de avaliação

1. realização de ??+1 trabalhos práticos (40) : ?? escritos e um de implementação.
2. exame (60)

Programa da disciplina

1. Breve introdução aos métodos formais e a técnicas de verificação formal.
2. Verificação por model checking de sistemas reativos
3. Verificação de Dedutiva de Programas

Verificação por model checking de sistemas reativos

1. Modelação de sistemas paralelos: sistemas transição
2. Paralelismo e comunicação
3. Propriedades temporais lineares: segurança, liveness e fairness
4. (Propriedades regulares)
5. Lógicas temporais: linear (LTL) e ramificada (CTL e CTL*).
6. Modelação e especificação usando um model checker
7. Algoritmos de model checking para LTL e CTL
8. Model checking simbólico: BDDs e OBDDs.
9. Técnicas de implementação de model checking.
10. Algoritmos de decisão baseados em autómatos

Verificação de Dedutiva de Programas

1. Cálculos de correcção (Lógica de Hoare)
2. Pré-condições mais fracas e algoritmos de geração de condições de verificação.
3. Geração de obrigações de prova
4. Ferramentas para a especificação, verificação e certificação de programas imperativos.

Livros recomendados

- **Principles of Model Checking** [BKL08]
- **Rigorous Software Development**, [AFPMdS11]
- Logic in Computer Science, [HR04] (Cap. 3, 4, 6)

Métodos formais em desenho de sistemas informáticos

1. Especificações formais: linguagens *Z*, *VDM*, *B*, *JML*
2. Garantir que as especificações satisfazem determinadas propriedades
3. Derivar implementações de especificações
4. Verificar implementações em relação a especificações

Exemplos de sistemas críticos e erros de software

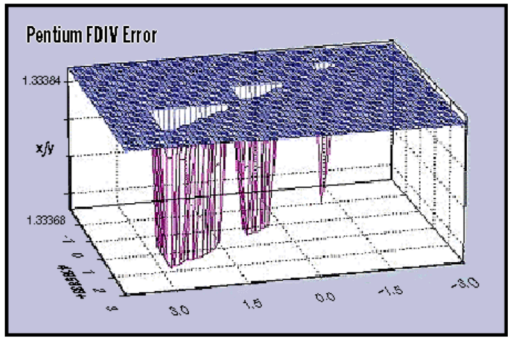
- Arianne-5, 1996
- Marte "Path Finder"
- Airbuses

Sistemas de controlo:

- Instalações Nucleares
- Controlo de tráfego
- Instrumentos médicos
- etc.

Em geral por cada 1000 linhas de código há um erro. Mas, o Windows 95 tinha pelo menos 5000 erros....

Ups!



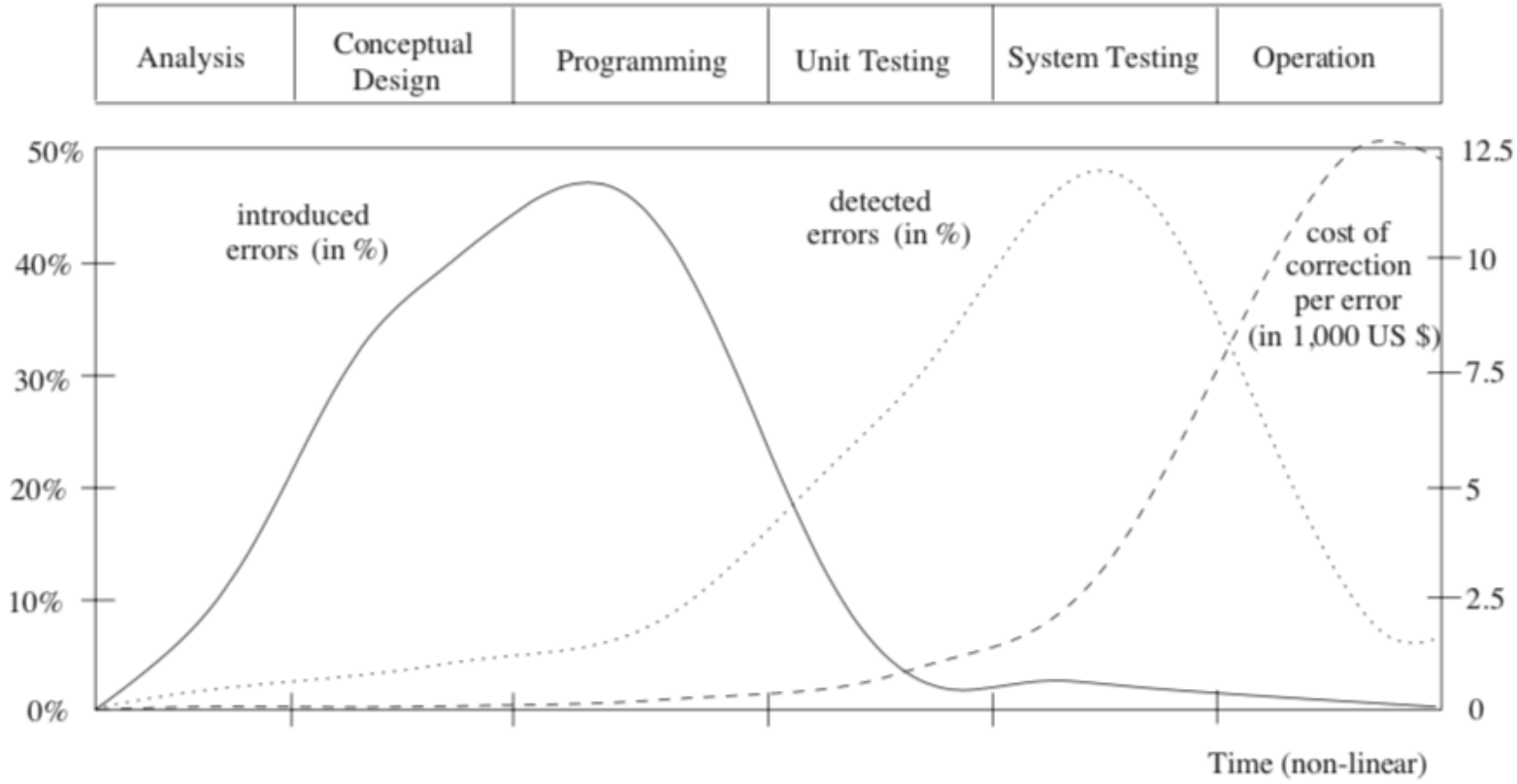
Intel Pentium bug caused loss of reputation and money.

Ariane 5 crashed within a few minutes after launch

Software bug caused Toyota to recall 1.2M Prius cars

Software race condition caused northeast blackout of 2003

Ciclo de vida do software, erros, e custos



- Como garantir ao nível de especificação o comportamento desejado: *problema de validação do modelo*.
- Como garantir (de uma especificação) uma implementação com o mesmo comportamento: *problema de relação formal entre especificação e implementação*.
- Estudo da especificação por: animação, transformação ou demonstração de propriedades
- Implementações podem:
  - ser derivadas de especificações;
  - garantir a sua correção por: construção, verificação, demonstração formal

- Como garantir ao nível de especificação o comportamento desejado: *problema de validação do modelo*.
- Como garantir (de uma especificação) uma implementação com o mesmo comportamento: *problema de relação formal entre especificação e implementação*.
- Estudo da especificação por: animação, transformação ou demonstração de propriedades
- Implementações podem:
  - ser derivadas de especificações;
  - garantir a sua correção por: construção, verificação, demonstração formal

- Como garantir ao nível de especificação o comportamento desejado: *problema de validação do modelo*.
- Como garantir (de uma especificação) uma implementação com o mesmo comportamento: *problema de relação formal entre especificação e implementação*.
- Estudo da especificação por: animação, transformação ou demonstração de propriedades
- Implementações podem:
  - ser derivadas de especificações;
  - garantir a sua correção por: construção, verificação, demonstração formal

Ferramentas formais de software

- Como garantir ao nível de especificação o comportamento desejado: *problema de validação do modelo*.
- Como garantir (de uma especificação) uma implementação com o mesmo comportamento: *problema de relação formal entre especificação e implementação*.
- Estudo da especificação por: animação, transformação ou demonstração de propriedades
- Implementações podem:
  - ser derivadas de especificações;
  - garantir a sua correção por: construção, verificação, demonstração formal

Ferramentas formais de software

- Como garantir ao nível de especificação o comportamento desejado: *problema de validação do modelo*.
- Como garantir (de uma especificação) uma implementação com o mesmo comportamento: *problema de relação formal entre especificação e implementação*.
- Estudo da especificação por: animação, transformação ou demonstração de propriedades
- Implementações podem:
  - ser derivadas de especificações;
  - garantir a sua correção por: construção, verificação, demonstração formal

Ferramentas formais de software

- Como garantir ao nível de especificação o comportamento desejado: *problema de validação do modelo*.
- Como garantir (de uma especificação) uma implementação com o mesmo comportamento: *problema de relação formal entre especificação e implementação*.
- Estudo da especificação por: animação, transformação ou demonstração de propriedades
- Implementações podem:
  - ser derivadas de especificações;
  - garantir a sua correção por: construção, verificação, demonstração formal

Noções centrais a técnicas/ferramentas de verificação

- A natureza operacional do sistema modelado: capturada por um sistema de transições
  - diferentes mecanismos implicam diferentes interpretações das noções de: estado, transição, transformação de estado
- A essência comportamental do sistema modelado, capturada por uma determinada lógica de programas.

Noções centrais a técnicas/ferramentas de verificação

- A natureza operacional do sistema modelado: capturada por um sistema de transições
  - diferentes mecanismos implicam diferentes interpretações das noções de: estado, transição, transformação de estado
- A essência comportamental do sistema modelado, capturada por uma determinada lógica de programas.

Noções centrais a técnicas/ferramentas de verificação

- A natureza operacional do sistema modelado: capturada por um sistema de transições
  - diferentes mecanismos implicam diferentes interpretações das noções de: estado, transição, transformação de estado
- A essência comportamental do sistema modelado, capturada por uma determinada lógica de programas.

Principais abordagens relativamente à especificação

- O comportamento do sistema é descrito com base em: operações; mecanismos disponíves; acções a realizar.

  Linguagens de especificação deste tipo designam-se como baseadas no estado ou modelo

- O comportamento do sistema é descrito com base em: dados manipulados; a forma como os dados mudam; a forma como os dados se relacionam

  Tais especificações são chamadas de especificações algébrica ou axiomáticas

Principais abordagens relativamente à especificação

- O comportamento do sistema é descrito com base em: operações; mecanismos disponíves; acções a realizar.

  Linguagens de especificação deste tipo designam-se como baseadas no estado ou modelo

- O comportamento do sistema é descrito com base em: dados manipulados; a forma como os dados mudam; a forma como os dados se relacionam

  Tais especificações são chamadas de especificações algébrica ou axiomáticas

Especificações baseadas no estado

- Capacidade para descrever a noção de estado
- Descrição de como as operações do sistema modificam o estado
- Formalização baseada em:
  - matemática discreta;
  - teoria de conjuntos;
  - teoria de categorias;
  - lógica

Especificações baseadas no estado

- Capacidade para descrever a noção de estado
- Descrição de como as operações do sistema modificam o estado
- Formalização baseada em:
  - matemática discreta;
  - teoria de conjuntos;
  - teoria de categorias;
  - lógica

Especificações baseadas no estado

- Capacidade para descrever a noção de estado
- Descrição de como as operações do sistema modificam o estado
- Formalização baseada em:
  - matemática discreta;
  - teoria de conjuntos;
  - teoria de categorias;
  - lógica

Especificações baseadas no estado

- Capacidade para descrever a noção de estado
- Descrição de como as operações do sistema modificam o estado
- Formalização baseada em:
  - matemática discreta;
  - teoria de conjuntos;
  - teoria de categorias;
  - lógica

Especificações baseadas no estado

- Capacidade para descrever a noção de estado
- Descrição de como as operações do sistema modificam o estado
- Formalização baseada em:
  - matemática discreta;
  - teoria de conjuntos;
  - teoria de categorias;
  - lógica

Especificações baseadas no estado

- Capacidade para descrever a noção de estado
- Descrição de como as operações do sistema modificam o estado
- Formalização baseada em:
  - matemática discreta;
  - teoria de conjuntos;
  - teoria de categorias;
  - lógica

Especificações baseadas no estado

- Capacidade para descrever a noção de estado
- Descrição de como as operações do sistema modificam o estado
- Formalização baseada em:
  - matemática discreta;
  - teoria de conjuntos;
  - teoria de categorias;
  - lógica

Especificações baseadas no estado

- Máquinas abstractas: sistema descrito por estados e por um conjunto finito de regras de transição entre estados.
- Teoria de categorias e conjuntos: estados descritos em termos de estruturas matemáticas (conjuntos, relações ou funções); transições expressas como invariantes, pré-condições e pós-condições.
- Modelos baseados em autómatos: para modelar sistemas com um comportamento concorrente; definir como o sistema reaje a estímulos e eventos; particularmente adequado para especificar sistemas reactivos, concorrentes ou de comunicação assim como protocolos.
- Linguagens modelo para sistemas de tempo-real (*cyberphysical*): capacidade para modelar conceitos físicos tais como tempo (ou duração), temperatura, inclinação, altitude, etc; sistemas de controlo que reajem a ambientes dinâmicos.

Especificações baseadas no estado

- Máquinas abstractas: sistema descrito por estados e por um conjunto finito de regras de transição entre estados.
- Teoria de categorias e conjuntos: estados descritos em termos de estruturas matemáticas (conjuntos, relações ou funções); transições expressas como invariantes, pré-condições e pós-condições.
- Modelos baseados em autómatos: para modelar sistemas com um comportamento concorrente; definir como o sistema reaje a estímulos e eventos; particularmente adequado para especificar sistemas reactivos, concorrentes ou de comunicação assim como protocolos.
- Linguagens modelo para sistemas de tempo-real (*cyberphysical*): capacidade para modelar conceitos físicos tais como tempo (ou duração), temperatura, inclinação, altitude, etc; sistemas de controlo que reajem a ambientes dinâmicos.

Especificações baseadas no estado

- Máquinas abstractas: sistema descrito por estados e por um conjunto finito de regras de transição entre estados.
- Teoria de categorias e conjuntos: estados descritos em termos de estruturas matemáticas (conjuntos, relações ou funções); transições expressas como invariantes, pré-condições e pós-condições.
- Modelos baseados em autómatos: para modelar sistemas com um comportamento concorrente; definir como o sistema reaje a estímulos e eventos; particularmente adequado para especificar sistemas reactivos, concorrentes ou de comunicação assim como protocolos.
- Linguagens modelo para sistemas de tempo-real (*cyberphysical*): capacidade para modelar conceitos físicos tais como tempo (ou duração), temperatura, inclinação, altitude, etc; sistemas de controlo que reajem a ambientes dinâmicos.

Especificações baseadas no estado

- Máquinas abstractas: sistema descrito por estados e por um conjunto finito de regras de transição entre estados.
- Teoria de categorias e conjuntos: estados descritos em termos de estruturas matemáticas (conjuntos, relações ou funções); transições expressas como invariantes, pré-condições e pós-condições.
- Modelos baseados em autómatos: para modelar sistemas com um comportamento concorrente; definir como o sistema reaje a estímulos e eventos; particularmente adequado para especificar sistemas reactivos, concorrentes ou de comunicação assim como protocolos.
- Linguagens modelo para sistemas de tempo-real (*cyberphysical*): capacidade para modelar conceitos físicos tais como tempo (ou duração), temperatura, inclinação, altitude, etc; sistemas de controlo que reajem a ambientes dinâmicos.

Exemplos

- Máquinas abstractas:
  - ASM_Gopher serviu de base a uma formalisação da linguagem Java;
  - O método B (e a sua linguagem de especificação B), cuja metodologia está próxima à modelação orientada a objectos. Deu origem a várias implementações: Atelier B, BRILLANT, ProB, Rodin, etc...
- Teoria de categorias e conjuntos:
  - Os métodos formais Z e VDM são baseados em lógica de predicados e teoria de conjuntos. Estiveram na base de outros sistemas: RAISE, Alloy (estende a linguagem Z para incorporar análise parcial)
  - Specware, Charity são formalismos baseados em teoria de categorias.

Exemplos

- Máquinas abstractas:
  - ASM_Gopher serviu de base a uma formalisação da linguagem Java;
  - O método B (e a sua linguagem de especificação B), cuja metodologia está próxima à modelação orientada a objectos. Deu origem a várias implementações: Atelier B, BRILLANT, ProB, Rodin, etc...
- Teoria de categorias e conjuntos:
  - Os métodos formais Z e VDM são baseados em lógica de predicados e teoria de conjuntos. Estiveram na base de outros sistemas: RAISE, Alloy (estende a linguagem Z para incorporar análise parcial)
  - Specware, Charity são formalismos baseados em teoria de categorias.

Exemplos

- Máquinas abstractas:
  - ASM_Gopher serviu de base a uma formalisação da linguagem Java;
  - O método B (e a sua linguagem de especificação B), cuja metodologia está próxima à modelação orientada a objectos. Deu origem a várias implementações: Atelier B, BRILLANT, ProB, Rodin, etc...
- Teoria de categorias e conjuntos:
  - Os métodos formais Z e VDM são baseados em lógica de predicados e teoria de conjuntos. Estiveram na base de outros sistemas: RAISE, Alloy (estende a linguagem Z para incorporar análise parcial)
  - Specware, Charity são formalismos baseados em teoria de categorias.

Exemplos

- Máquinas abstractas:
  - ASM_Gopher serviu de base a uma formalisação da linguagem Java;
  - O método B (e a sua linguagem de especificação B), cuja metodologia está próxima à modelação orientada a objectos. Deu origem a várias implementações: Atelier B, BRILLANT, ProB, Rodin, etc...
- Teoria de categorias e conjuntos:
  - Os métodos formais Z e VDM são baseados em lógica de predicados e teoria de conjuntos. Estiveram na base de outros sistemas: RAISE, Alloy (estende a linguagem Z para incorporar análise parcial)
  - Specware, Charity são formalismos baseados em teoria de categorias.

Exemplos

- Máquinas abstractas:
  - ASM_Gopher serviu de base a uma formalisação da linguagem Java;
  - O método B (e a sua linguagem de especificação B), cuja metodologia está próxima à modelação orientada a objectos. Deu origem a várias implementações: Atelier B, BRILLANT, ProB, Rodin, etc...
- Teoria de categorias e conjuntos:
  - Os métodos formais Z e VDM são baseados em lógica de predicados e teoria de conjuntos. Estiveram na base de outros sistemas: RAISE, Alloy (estende a linguagem Z para incorporar análise parcial)
  - Specware, Charity são formalismos baseados em teoria de categorias.

Exemplos

- Máquinas abstractas:
  - ASM_Gopher serviu de base a uma formalisação da linguagem Java;
  - O método B (e a sua linguagem de especificação B), cuja metodologia está próxima à modelação orientada a objectos. Deu origem a várias implementações: Atelier B, BRILLANT, ProB, Rodin, etc...
- Teoria de categorias e conjuntos:
  - Os métodos formais Z e VDM são baseados em lógica de predicados e teoria de conjuntos. Estiveram na base de outros sistemas: RAISE, Alloy (estende a linguagem Z para incorporar análise parcial)
  - Specware, Charity são formalismos baseados em teoria de categorias.

Exemplos

- Linguagens modelo para sistemas de tempo-real:
  - a linguagem de fluxo de dados Lustre e o ambiente de modelação gráfico SCADE (baseado em Lustre), permitem exprimir concorrência síncrona em termos de fluxo de dados
  - ferramentas de model-checking como Uppaal ou Kronos, baseadas em automatos temporizados (*timed-automata*).
  - ferramentas como Hytech e KeYmaera, baseadas em autómatos híbridos com o intuito de modelar sistemas dinâmicos com comportamento interactivo. (e.g.*sistemas de transportes*)

Exemplos

- Linguagens modelo para sistemas de tempo-real:
  - a linguagem de fluxo de dados Lustre e o ambiente de modelação gráfico SCADE (baseado em Lustre), permitem exprimir concorrência síncrona em termos de fluxo de dados
  - ferramentas de model-checking como Uppaal ou Kronos, baseadas em automatos temporizados (*timed-automata*).
  - ferramentas como Hytech e KeYmaera, baseadas em autómatos híbridos com o intuito de modelar sistemas dinâmicos com comportamento interactivo. (e.g.*sistemas de transportes*)

Exemplos

- Linguagens modelo para sistemas de tempo-real:
  - a linguagem de fluxo de dados Lustre e o ambiente de modelação gráfico SCADE (baseado em Lustre), permitem exprimir concorrência síncrona em termos de fluxo de dados
  - ferramentas de model-checking como Uppaal ou Kronos, baseadas em automatos temporizados (*timed-automata*).
  - ferramentas como Hytech e KeYmaera, baseadas em autómatos híbridos com o intuito de modelar sistemas dinâmicos com comportamento interactivo. (e.g.*sistemas de transportes*)

Exemplos

- Linguagens modelo para sistemas de tempo-real:
  - a linguagem de fluxo de dados Lustre e o ambiente de modelação gráfico SCADE (baseado em Lustre), permitem exprimir concorrência síncrona em termos de fluxo de dados
  - ferramentas de model-checking como Uppaal ou Kronos, baseadas em automatos temporizados (*timed-automata*).
  - ferramentas como Hytech e KeYmaera, baseadas em autómatos híbridos com o intuito de modelar sistemas dinâmicos com comportamento interactivo. (e.g.*sistemas de transportes*)

Especificações algébricas

- Consistem num conjunto de declarações, assinaturas de funções e axiomas que declaram o comportamento básico de cada símbolo funcional.
- Exemplos de ferramentas e linguagens:
  - CASL, OBJ, Clear, Larch, ACT-ONE são ferramentas baseadas em especificações algébricas
  - LOTOS - contém primitivas CCS (Calculus of Communicating Systems), que permite especificar comportamentos de sistemas concorrentes.

Especificações algébricas

- Consistem num conjunto de declarações, assinaturas de funções e axiomas que declaram o comportamento básico de cada símbolo funcional.
- Exemplos de ferramentas e linguagens:
  - CASL, OBJ, Clear, Larch, ACT-ONE são ferramentas baseadas em especificações algébricas
  - LOTOS - contém primitivas CCS (Calculus of Communicating Systems), que permite especificar comportamentos de sistemas concorrentes.

Especificações algébricas

- Consistem num conjunto de declarações, assinaturas de funções e axiomas que declaram o comportamento básico de cada símbolo funcional.
- Exemplos de ferramentas e linguagens:
  - CASL, OBJ, Clear, Larch, ACT-ONE são ferramentas baseadas em especificações algébricas
  - LOTOS - contém primitivas CCS (Calculus of Communicating Systems), que permite especificar comportamentos de sistemas concorrentes.

Especificações algébricas

- Consistem num conjunto de declarações, assinaturas de funções e axiomas que declaram o comportamento básico de cada símbolo funcional.
- Exemplos de ferramentas e linguagens:
  - CASL, OBJ, Clear, Larch, ACT-ONE são ferramentas baseadas em especificações algébricas
  - LOTOS - contém primitivas CCS (Calculus of Communicating Systems), que permite especificar comportamentos de sistemas concorrentes.

- Inclui linguagens baseadas em lógica, linguagens funcionais, linguagens de reescrita e linguagens de formalização de semânticas
  - Linguagens baseadas em lógica como o Prolog: baseadas na noção de predicado.
  - Linguagens funcionais cuja linguagem base é o λ-calculus: Scheme, SML, Haskell e OCaml; assistentes de prova como: ACL2, Coq, PVS, HOL, Isabelle e Agda, são todos baseados em variantes tipadas e extensões ao λ-calculus. Pelo isomorfismo de Curry-Howard as linguagens têm associados sistemas de prova baseados em tipos que também podem ser usadas como linguagens lógicas de ordem-superior.
  - Sistemas de reescrita como ELAN ou SPIKE: o comportamento dos símbolos funcionais é descrito por sistemas equacionais, a execução é baseada noção de redução (como no λ-calculus).

- Inclui linguagens baseadas em lógica, linguagens funcionais, linguagens de reescrita e linguagens de formalização de semânticas
  - Linguagens baseadas em lógica como o Prolog: baseadas na noção de predicado.
  - Linguagens funcionais cuja linguagem base é o λ-calculus: Scheme, SML, Haskell e OCaml; assistentes de prova como: ACL2, Coq, PVS, HOL, Isabelle e Agda, são todos baseados em variantes tipadas e extensões ao λ-calculus. Pelo isomorfismo de Curry-Howard as linguagens têm associados sistemas de prova baseados em tipos que também podem ser usadas como linguagens lógicas de ordem-superior.
  - Sistemas de reescrita como ELAN ou SPIKE: o comportamento dos símbolos funcionais é descrito por sistemas equacionais, a execução é baseada noção de redução (como no λ-calculus).

- Inclui linguagens baseadas em lógica, linguagens funcionais, linguagens de reescrita e linguagens de formalização de semânticas
  - Linguagens baseadas em lógica como o Prolog: baseadas na noção de predicado.
  - Linguagens funcionais cuja linguagem base é o λ-calculus: Scheme, SML, Haskell e OCaml; assistentes de prova como: ACL2, Coq, PVS, HOL, Isabelle e Agda, são todos baseados em variantes tipadas e extensões ao λ-calculus. Pelo isomorfismo de Curry-Howard as linguagens têm associados sistemas de prova baseados em tipos que também podem ser usadas como linguagens lógicas de ordem-superior.
  - Sistemas de reescrita como ELAN ou SPIKE: o comportamento dos símbolos funcionais é descrito por sistemas equacionais, a execução é baseada noção de redução (como no λ-calculus).

- Inclui linguagens baseadas em lógica, linguagens funcionais, linguagens de reescrita e linguagens de formalização de semânticas
  - Linguagens baseadas em lógica como o Prolog: baseadas na noção de predicado.
  - Linguagens funcionais cuja linguagem base é o λ-calculus: Scheme, SML, Haskell e OCaml; assistentes de prova como: ACL2, Coq, PVS, HOL, Isabelle e Agda, são todos baseados em variantes tipadas e extensões ao λ-calculus. Pelo isomorfismo de Curry-Howard as linguagens têm associados sistemas de prova baseados em tipos que também podem ser usadas como linguagens lógicas de ordem-superior.
  - Sistemas de reescrita como ELAN ou SPIKE: o comportamento dos símbolos funcionais é descrito por sistemas equacionais, a execução é baseada noção de redução (como no λ-calculus).

Pretende-se verificar a correção de sistemas informáticos e programas:

1. asíncronos/síncronos
2. analógico/digital hardware
3. mono/multi processadores
4. linguagens: imperativas, funcionais, lógicas, oo
5. sequencial ou multi-*threaded*
6. sistemas de operação convencionais ou tempo real
7. sistemas embebidos
8. sistemas distribuídos

Pretende-se verificar a correção de sistemas informáticos e programas:

1. asíncronos/síncronos
2. analógico/digital hardware
3. mono/multi processadores
4. linguagens: imperativas, funcionais, lógicas, oo
5. sequencial ou multi-*threaded*
6. sistemas de operação convencionais ou tempo real
7. sistemas embebidos
8. sistemas distribuídos

Pretende-se verificar a correção de sistemas informáticos e programas:

1. asíncronos/síncronos
2. analógico/digital hardware
3. mono/multi processadores
4. linguagens: imperativas, funcionais, lógicas, oo
5. sequencial ou multi-*threaded*
6. sistemas de operação convencionais ou tempo real
7. sistemas embebidos
8. sistemas distribuídos

Verificação

Pretende-se verificar a correção de sistemas informáticos e programas:

1. asíncronos/síncronos
2. analógico/digital hardware
3. mono/multi processadores
4. linguagens: imperativas, funcionais, lógicas, oo
5. sequencial ou multi-*threaded*
6. sistemas de operação convencionais ou tempo real
7. sistemas embebidos
8. sistemas distribuídos

Pretende-se verificar a correção de sistemas informáticos e programas:

1. asíncronos/síncronos
2. analógico/digital hardware
3. mono/multi processadores
4. linguagens: imperativas, funcionais, lógicas, oo
5. sequencial ou multi-*threaded*
6. sistemas de operação convencionais ou tempo real
7. sistemas embebidos
8. sistemas distribuídos

Pretende-se verificar a correção de sistemas informáticos e programas:

1. asíncronos/síncronos
2. analógico/digital hardware
3. mono/multi processadores
4. linguagens: imperativas, funcionais, lógicas, oo
5. sequencial ou multi-*threaded*
6. sistemas de operação convencionais ou tempo real
7. sistemas embebidos
8. sistemas distribuídos

Pretende-se verificar a correção de sistemas informáticos e programas:

1. asíncronos/síncronos
2. analógico/digital hardware
3. mono/multi processadores
4. linguagens: imperativas, funcionais, lógicas, oo
5. sequencial ou multi-*threaded*
6. sistemas de operação convencionais ou tempo real
7. sistemas embebidos
8. sistemas distribuídos

Pretende-se verificar a correção de sistemas informáticos e programas:

1. asíncronos/síncronos
2. analógico/digital hardware
3. mono/multi processadores
4. linguagens: imperativas, funcionais, lógicas, oo
5. sequencial ou multi-*threaded*
6. sistemas de operação convencionais ou tempo real
7. sistemas embebidos
8. sistemas distribuídos

Tipos de sistemas informáticos

1. Transformacionais: lêem dados de entrada e produzem um resultado; devem terminar. Exemplo: compiladores
2. Interactivos: interagem com o utilizador via acções/reações; normalmente não terminam.
3. Reactivos: a interação é determinada pelo ambiente (restrições de tempo real); são normalmente implementados de forma concorrente. Exemplo: acessos a uma base de dados de voos; contrololador de comboios.

Tipos de sistemas informáticos

1. Transformacionais: lêem dados de entrada e produzem um resultado; devem terminar. Exemplo: compiladores
2. Interactivos: interagem com o utilizador via acções/reações; normalmente não terminam.
3. Reactivos: a interação é determinada pelo ambiente (restrições de tempo real); são normalmente implementados de forma concorrente. Exemplo: acessos a uma base de dados de voos; contrololador de comboios.

Tipos de sistemas informáticos

1. Transformacionais: lêem dados de entrada e produzem um resultado; devem terminar. Exemplo: compiladores
2. Interactivos: interagem com o utilizador via acções/reações; normalmente não terminam.
3. Reactivos: a interação é determinada pelo ambiente (restrições de tempo real); são normalmente implementados de forma concorrente. Exemplo: acessos a uma base de dados de voos; contrololador de comboios.

Plataforma para a modelação de sistemas

Linguagem de especificação para descrever as propriedades que se quer verificar

Método de verificação não basta simular são necessárias demonstrações rigorosas usando lógica.

- "em papel, à mão"
- com alguma automação
- usando ferramentas computacionais automáticas ou interactivas

Plataforma para a modelação de sistemas

Linguagem de especificação para descrever as propriedades que se quer verificar

Método de verificação não basta simular são necessárias demonstrações rigorosas usando lógica.

- "em papel, à mão"
- com alguma automação
- usando ferramentas computacionais automáticas ou interactivas

Plataforma para a modelação de sistemas

Linguagem de especificação para descrever as propriedades que se quer verificar

Método de verificação não basta simular são necessárias demonstrações rigorosas usando lógica.

- "em papel, à mão"
- com alguma automação
- usando ferramentas computacionais automáticas ou interactivas

Plataforma para a modelação de sistemas

Linguagem de especificação para descrever as propriedades que se quer verificar

Método de verificação não basta simular são necessárias demonstrações rigorosas usando lógica.

- "em papel, à mão"
- com alguma automação
- usando ferramentas computacionais automáticas ou interactivas

Plataforma para a modelação de sistemas

Linguagem de especificação para descrever as propriedades que se quer verificar

Método de verificação não basta simular são necessárias demonstrações rigorosas usando lógica.

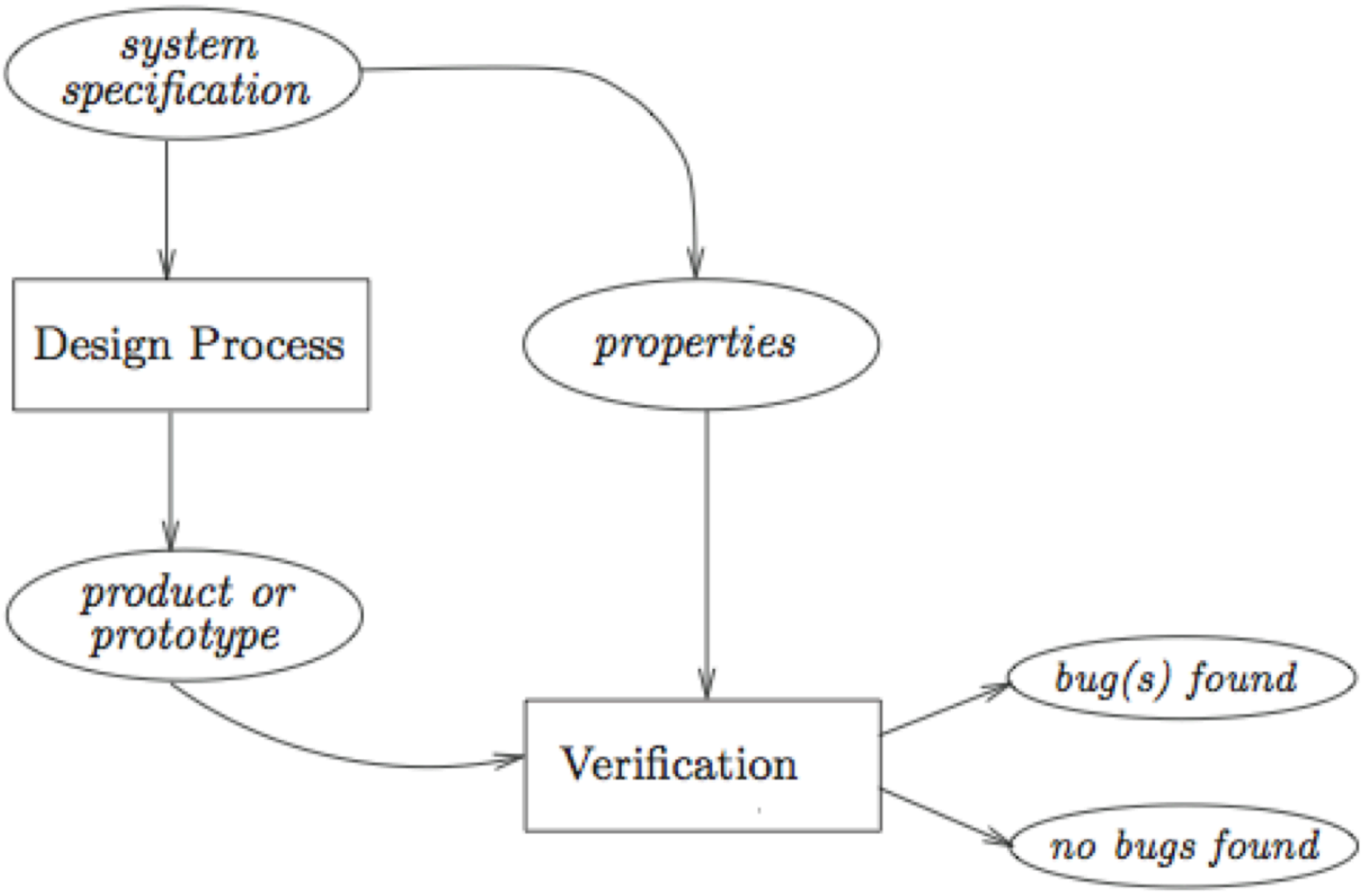
- "em papel, à mão"
- com alguma automação
- usando ferramentas computacionais automáticas ou interactivas

Plataforma para a modelação de sistemas

Linguagem de especificação para descrever as propriedades que se quer verificar

Método de verificação não basta simular são necessárias demonstrações rigorosas usando lógica.

- "em papel, à mão"
- com alguma automação
- usando ferramentas computacionais automáticas ou interactivas


system specification
Design Process
properties
product or prototype
Verification
bug(s) found
no bugs found

Abordagens para a Verificação

Sistema dedutivos vs. Modelos
SD: O sistema é descrito por um conjunto de fórmulas Γ, a especificação é uma fórmula φ, e pretende-se $\Gamma \vdash \varphi$ (em geral semi-automático)
M: O sistema é descrito por um modelo $\mathcal{M}$, a especificação é uma fórmula φ, e pretende-se $\mathcal{M} \models \varphi$. (em geral automático para modelos finitos).

Grau de automatização automático/assistido por computador

Completude vs. Propriedades A especificação descreve uma propriedade ou todo o comportamento do sistema.

Domínio Hardware/Software; sequencial ou concorrente; funcional (calcula um valor a partir dos dados) ou reactivo (sistemas de operação, embebidos, hardware, etc)

Estática vs Dinâmica A verificação é feita antes ou em tempo de execução.

Abordagens para a Verificação

Sistema dedutivos vs. Modelos

SD: O sistema é descrito por um conjunto de fórmulas Γ, a especificação é uma fórmula φ, e pretende-se $\Gamma \vdash \varphi$ (em geral semi-automático)
M: O sistema é descrito por um modelo $\mathcal{M}$, a especificação é uma fórmula φ, e pretende-se $\mathcal{M} \models \varphi$. (em geral automático para modelos finitos).

Grau de automatização automático/assistido por computador

Completude vs. Propriedades A especificação descreve uma propriedade ou todo o comportamento do sistema.

Domínio Hardware/Software; sequencial ou concorrente; funcional (calcula um valor a partir dos dados) ou reactivo (sistemas de operação, embebidos, hardware, etc)

Estática vs Dinâmica A verificação é feita antes ou em tempo de execução.

Abordagens para a Verificação

Sistema dedutivos vs. Modelos

SD: O sistema é descrito por um conjunto de fórmulas Γ, a especificação é uma fórmula φ, e pretende-se $\Gamma \vdash \varphi$ (em geral semi-automático)
M: O sistema é descrito por um modelo $\mathcal{M}$, a especificação é uma fórmula φ, e pretende-se $\mathcal{M} \models \varphi$. (em geral automático para modelos finitos).

Grau de automatização automático/assistido por computador

Completude vs. Propriedades A especificação descreve uma propriedade ou todo o comportamento do sistema.

Domínio Hardware/Software; sequencial ou concorrente; funcional (calcula um valor a partir dos dados) ou reactivo (sistemas de operação, embebidos, hardware, etc)

Estática vs Dinâmica A verificação é feita antes ou em tempo de execução.

Abordagens para a Verificação

Sistema dedutivos vs. Modelos

SD: O sistema é descrito por um conjunto de fórmulas Γ, a especificação é uma fórmula φ, e pretende-se $\Gamma \vdash \varphi$ (em geral semi-automático)
M: O sistema é descrito por um modelo $\mathcal{M}$, a especificação é uma fórmula φ, e pretende-se $\mathcal{M} \models \varphi$. (em geral automático para modelos finitos).

Grau de automatização automático/assistido por computador

Completude vs. Propriedades A especificação descreve uma propriedade ou todo o comportamento do sistema.

Domínio Hardware/Software; sequencial ou concorrente; funcional (calcula um valor a partir dos dados) ou reactivo (sistemas de operação, embebidos, hardware, etc)

Estática vs Dinâmica A verificação é feita antes ou em tempo de execução.

Abordagens para a Verificação

Sistema dedutivos vs. Modelos

SD: O sistema é descrito por um conjunto de fórmulas Γ, a especificação é uma fórmula φ, e pretende-se $\Gamma \vdash \varphi$ (em geral semi-automático)
M: O sistema é descrito por um modelo $\mathcal{M}$, a especificação é uma fórmula φ, e pretende-se $\mathcal{M} \models \varphi$. (em geral automático para modelos finitos).

Grau de automatização automático/assistido por computador

Completude vs. Propriedades A especificação descreve uma propriedade ou todo o comportamento do sistema.

Domínio Hardware/Software; sequencial ou concorrente; funcional (calcula um valor a partir dos dados) ou reactivo (sistemas de operação, embebidos, hardware, etc)

Estática vs Dinâmica A verificação é feita antes ou em tempo de execução.

Model checking (Verificação de modelos)

Automático, Baseado em modelos, Verificação de propriedades, Sistemas concorrentes e reactivos, dinâmicos

Verificação de programas

Interactivo, Sistemas dedutivos, Verificação de propriedades, Programas que terminam

Mas estas abordagens não são estritas: podemos misturar as técnicas... por exemplo em sistemas embebidos, em sistema portadores de prova (*proof carrying code*).

Model checking (Verificação de modelos)

Automático, Baseado em modelos, Verificação de propriedades, Sistemas concorrentes e reactivos, dinâmicos

Verificação de programas

Interactivo, Sistemas dedutivos, Verificação de propriedades, Programas que terminam

Mas estas abordagens não são estritas: podemos misturar as técnicas... por exemplo em sistemas embebidos, em sistema portadores de prova (*proof carrying code*).

Podemos classificar os métodos de prova em três categorias:

- Provas não assistidas computacionalmente: as demonstrações são efectuadas à mão e podem ser descritas informalmente.
- Ferramentas que introduzem um sistema formal para formular demonstrações: o uso de linguagem natural não é permitido.
- Ferramentas de prova computacionais.

Lógicas:

- lógica proposicional, lógica de primeira ordem, lógica de ordem superior
- lógica clássica versus lógica intuicionistica
- lógicas temporais

Ferramentas de prova

- Sistemas de demonstração automática: dependem da decidibilidade de fragmentos da lógica que lhes serve de base
  - ELAN: reescrita de primeira ordem
  - ACS2: lógica de primeira ordem
  - Resolutores SMT (Satisfability Module Theory): Yices, CVC3, Z3, Alt-Ergo, Simplify: algoritmos decisão para inteiros, reais, "arrays", etc.
  - Ao contrário de model-checkers permitem racionalizar acerca de conjuntos infinitos
- Sistemas de demonstração assistida: consideram lógicas mais expressivas e potencialmente não-decidíveis
  - Combinam duas capacidades: verificação de provas; construção assistida de provas
  - Na maior parte dos assistentes de prova as demonstrações são construídas interactivamente aplicando um conjunto de tácticas: case, elim, change, rewrite, simpl, discriminate, injection, induction.


José Bacelar Almeida, Maria João Frade, Jorge Sousa Pinto, and Simão Melo de Sousa.
Rigorous Software Development: An Introduction to Program Verification.
Springer, 2011.


Christel Baier, Joost-Pieter Katoen, and Kim Guldstrand Larsen.
Principles of Model Checking.
MIT Press, 2008.


Michael Huth and Mark Ryan.
Logic in Computer Science: Modelling and reasoning about systems.
CUP, 2004.