Aula 1

1 Disciplina

Verificação Formal de Software

URL:http://www.dcc.fc.up.pt/~ nam/web/Teaching/VFS1213/index.html
Escolaridade: 3T e 1.5TP (para dúvidas/realização de trabalhos)

Método de avaliação

1. realização e apresentação de trabalhos práticos (individuais) (5) ($Trab$).
2. resumo escrito e apresentação de um tema/sistema/artigo científico/implementação ($Exame$).

Aprovação

$$(Trab * 15 + Exame * 5)/20 \geq 9.5, \quad Exame \geq 2$$

2 Programa

Programa da disciplina

Verificação por modelos de sistemas reactivos (*model checking*)

1. Sistemas reactivos
2. Lógicas temporais:
   (a) linear (LTL)
   (b) ramificada (CTL e CTL*)
3. *Model checkers*: NUSMV
4. *Model checking* simbólico: BDDs e OBDDs
5. Algoritmos de *model checking*
6. Algoritmos de decisão baseados em autómatos finitos

Programa da disciplina

Verificação de Programas

1. Cálculos de correcção (Lógica de Hoare)
2. Geração de obrigações de prova
3. Ferramentas para a especificação, verificação e certificação de programas imperativos: geradores de condições de verificação.
4. Why3

Programa da disciplina

Demonstração automática
(introdução)

1. Demonstradores automáticos de teoremas baseados em SAT-DPLL e teorias decidíveis: inteiros, reais, "arrays", etc.
2. SMTs e algoritmos decisão.

3 Bibliografia

Bibliografia

Livros recomendados

- **Logic in Computer Science**, [HR04] (Cap. 3, 4, 6)
- Principles of Model Checking. [BKL08]
- Model Checking. Edmund Clarke, [JGP99]
- Systems and Software Verification, [BBF+01]
- **Rigorous Software Development**, [AFPMdS11]
- Specification and Verification I, Mike Gordon.
- Decision Procedures (SMT solvers)[]

4 Métodos formais em desenho de sistemas informáticos

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

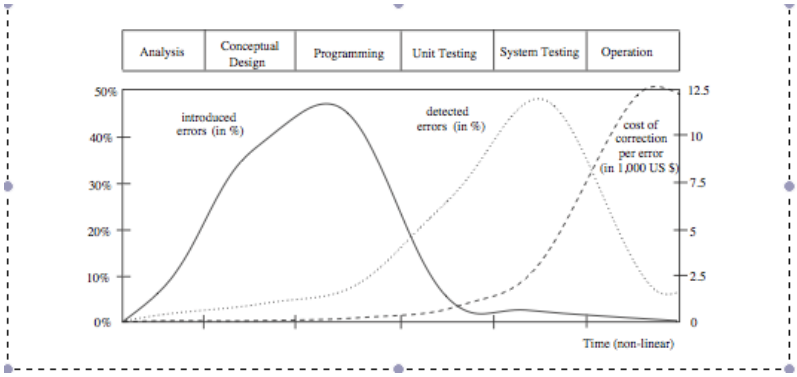
Exemplos de sistesmas críticos e erros de software

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

Sistemas de controlo:

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

Ciclo de vida do software, erros, e custos



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 de 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.

Principais abordagens relativamente a especificaçao

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

- 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); transicções 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: 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 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 automátos de tempo.
  - ferramentas como Hytech e Keymaera, baseadas em autómatos híbridos com o intuito de modelar sistemas dinâmicos com comportamento interactivo.

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.

Modelos Declarativos

- 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).

5 Verificação

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

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.

Verificação

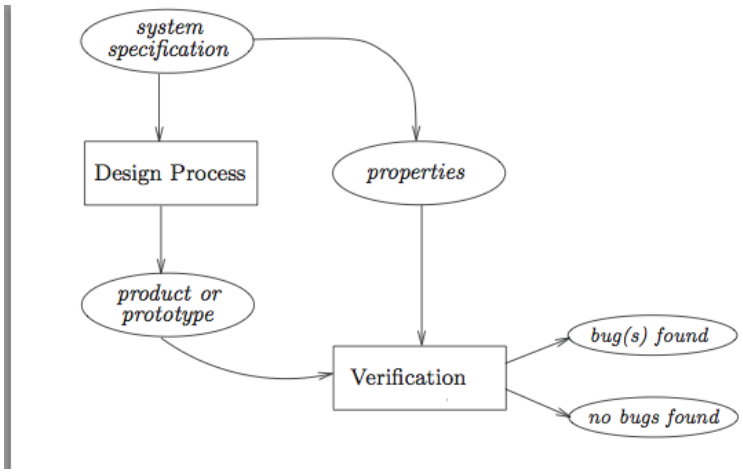
Verificação formal

Plataforma para a modelação de sistemas

Linguagem de especificação para descrever as propriedades que se quer verificar

Método de verificação

Verificação



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 $\Gamma \vdash \phi$ (em geral semi-automático)

M: O sistema é descrito por um modelo $\mathcal{M}$, a especificação é uma fórmula ϕ, e pretende-se $\mathcal{M} \models \phi$. (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.

Métodos de Verificação

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*).

Especificação e Prova

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.

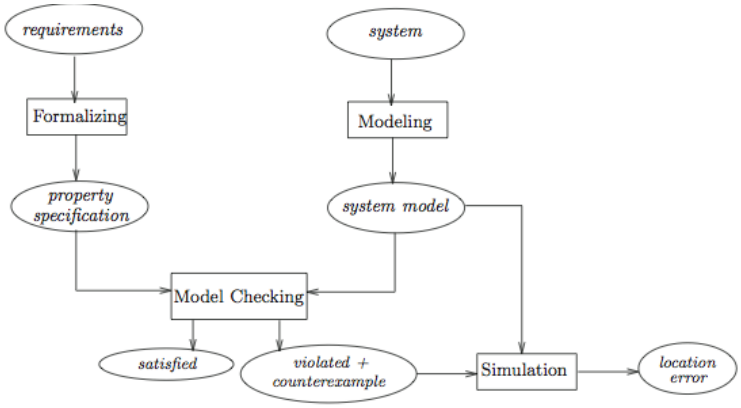
Lógicas:

- 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.

Model checking



Model checking

Os sistema reactivos são descritos pelas suas propriedades temporais. Uma especificação é verificada se for satisfeita ao longo da execução do sistema.

Assim, o *Model checking* é baseado em Lógicas temporais.

Um modelo temporal $\mathcal{M}$ é constituído por um conjunto de estados e transições entre eles (sistema de transições).

Uma fórmula ϕ pode ser **V** nalguns estado e **F** noutros.

A noção de verdade não é absoluta, mas sim dinâmica.

Dado um estado s, pretende-se saber se:

$$\mathcal{M}, s \models \phi$$

O verificador de modelos (*model checker*) é um algoritmo que decide este problema (responde sim ou não).

Propriedades

***Reachability* (estados atingíveis)** propriedades que garantem que um dado estado pode ser atingido

***Safety* (segurança)** propriedades que têm de se verificar para todas as computações e em todos os estados

***Liveness* (vivacidade)** indicam que determinados estados do sistema têm de ser atingidos.

Persistência indicam que para todas as computações a partir de um dado estado uma certa propriedade tem de se verificar.

***Fairness* (razoabilidade)** indicam que uma dada propriedade tem de se verificar um número não finito de vezes. Exemplo: nenhum processo é esquecido sempre por um temporizador.

6 Bibliografia

Referências

[AFPMdS11] José Bacelar Almeida, Maria João Frade, Jorge Sousa Pinto, and Simão Melo de Sousa. *Rigorous Software Development: An Introduction to Program Verification*. Springer, 2011.

[BBF+01] B. Berard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, and P. Schnoebelen. *Systems and Software Verification*. Springer-Verlag, 2001.

[BKL08] Christel Baier, Joost-Pieter Katoen, and Kim Guldstrand Larsen. *Principles of Model Checking*. MIT Press, 2008.

[HR04] Michael Huth and Mark Ryan. *Logic in Computer Science: Modelling and reasoning about systems*. CUP, 2004.

[JGP99] Edmund M. Clarke Jr, Orna Grumberg, and Doron A. Peled. *Model Checking*. The MIT Press, 1999.