Mestrado em Engenharia Informática,
Universidade do Minho

Falsification of Hybrid Programs

PROJETO EM MÉTODOS FORMAIS
DE PROGRAMAÇÃO

Grupo:
João Duarte (PG60110)
Luís Silva (PG60279)

Contexto e Problema

Programas Híbridos:

Os sistemas ciber-físicos integram algoritmos de controlo de software com ambientes físicos dinâmicos

Problemas:

Desafio Black-Box:
Não temos acesso aos seus modelos internos

Explosão do Estado:
Torna o teste exaustivo computacionalmente inviável

Objetivo:

Falsificação como “adversário algorítmico”, de forma a expor vulnerabilidades de forma sistemática

A Abordagem - Falsificação

O que é a Falsificação?
Ferramenta que adota uma postura proativa, agindo como um adversário que tenta encontrar o contraexemplo que quebra a segurança

Metodologia Black-Box:
Abordagem que trata o sistema como uma caixa-negra e explora o ambiente operacional à procura de inputs de perturbação ou configurações iniciais que levem o sistema a violar as suas propriedades definidas

Objetivos do Projeto

Métrica Quantitativa:
Estabelecer uma métrica para avaliar o nível de violação de uma especificação de segurança durante uma simulação

Integração no Lince 2.0:
Extensão do parser para interpretação de especificações LTL complexas e comandos para a procuras automáticas

Benchmarking:
Comparação entre uma estratégia aleatória de Monte Carlo e uma abordagem otimizada por Hill Climbing

Caso de Estudo (Adaptive Cruise Control)

Modelo:
Controlador que ajusta dinamicamente a aceleração do veículo seguidor em relação a um líder

Input program

```
1  // Adaptive Cruise Control (ACC)
2  p:=0; v:=0;  // follower
3  pl:=50; vl:=10; // leader
4  al:=0;
5  while true {
6    // decide to speed up (acc=2) or brake (acc=-2
7    if (v-8)^2 + 4*(p-pl+v-9) < 0
8    then p'=v, v'= 2, pl'=vl, vl'=al for 1;
9    else p'=v, v'=-2, pl'=vl, vl'=al for 1;
10 }
11 ----
```

Figura 2 - Programa ACC

Extensão do Lince

- **Tradução LTL:** Conversão de especificações booleanas em métricas de custo contínuo (robustez), permitindo quantificar a proximidade da falha

- **Parser Hierárquico:** Arquitetura de quatro camadas que garante a interpretação rigorosa de operadores temporais complexos e aninhados

- **Manipulação Dinâmica:** Implementação de comandos de pesquisa (range) e propriedades (property) para injeção de parâmetros em tempo real, possibilitando testes automatizados.

```
val ltlUnary: P[LTL] = (
  (char('G') *> sps *> recurse).map(LTL.Globally(_)) |
  (char('F') *> sps *> recurse).map(LTL.Finally(_)) |
  (char('X') *> sps *> recurse).map(LTL.Next(_)) |
  ((char('!') | char('~')) *> sps *> recurse).map(LTL.Not(_)) |
  ltlAtomic
)

val ltlTemporal: P[LTL] = ltlUnary.flatMap { lhs =>
  val until = (string("U") *> sps *> ltlUnary).map(rhs => LTL.Until(lhs, rhs))
  val release = (string("R") *> sps *> ltlUnary).map(rhs => LTL.Release(lhs, rhs))
  until | release | P.pure(lhs)
}
```

Figura 3 - Parte do Parser Implementado

Monte Carlo (Random Sampling)

- Gera casos de teste independentes e distribuídos uniformemente pelo espaço de entrada
- Baseline imparcial, sofre da limitação de ser uma pesquisa não guiada, que depende inteiramente da sorte para atingir as fronteiras críticas onde o sistema falha
- Obter uma visão inicial da robustez do sistema, estabelecendo a base de comparação para os algoritmos de otimização mais avançados que implementámos em seguida

Hill Climbing

Passo 1 - Inicialização	Passo 2 - Exploração	Passo 3 - Avaliação & Update	Passo 4 - Terminação
Seleciona um estado inicial aleatório $x_0 \in \chi$ e calcula a robustez $c(x_0)$	Gera estados vizinhos Xneighbors aplicando pequenas perturbações a x_0	Se c(x') < c(x): atualiza posição. O algoritmo segue o "gradiente de perigo"	Para quando c(x') < 0 (falsificação!), mínimo local, ou 15 iterações

Gradiente de Perigo: o algoritmo aprende que parâmetros (posição, velocidade inicial) aproximam os veículos, transforma pesquisa cega em exploração dirigida.

Manipulação da AST do Lince 2.0

Como o algoritmo injeta fisicamente os parâmetros no simulador

O Hill Climbing não altera um ficheiro de texto, injeta os valores diretamente na **Árvore de Sintaxe Abstrata (AST)** do Lince 2.0 (através da função **overrideAssignments**)

A função percorre recursivamente a estrutura do programa:

- Assign(v,_) → Substitui o valor se v ∈ newValues
- Seq(p1, p2) → Propaga recursivamente em ambas
- ITE(b, pt, pf) → Avalia e propaga no then e no else
- While(b, p) → Propaga diretamente no corpo do ciclo
- Other → Deixa inalterado

```
def overrideAssignments(prog: Program, newValues: Map[String, Double]): Program = prog match {
  case Program.Assign(v, _) if newValues.contains(v) => Program.Assign(v, lince.syntax.Lince.Expr.Num(newValues(v)))
  case Program.Seq(p1, p2) => Program.Seq(overrideAssignments(p1, newValues), overrideAssignments(p2, newValues))
  case Program.ITE(b, pt, pf) => Program.ITE(b, overrideAssignments(pt, newValues), overrideAssignments(pf, newValues))
  case Program.While(b, p) => Program.While(b, overrideAssignments(p, newValues))
  case other => other
}
```

Figura 4 - Implementação da função overrideAssignments

O Cenário de Teste: ACC

Seguidor (Follower)

- p ∈ [10,40] → posição inicial variável
- v = 0 → começa parado
- Aceleração **+ 2** se seguro, **-2** se não

Veículo Líder (Leader)

- pl ∈ [10,20] → começa próximo do seguidor
- v = 10 → velocidade constante
- Zona de colisão apertada → díficil por azar

20 ut	0.2 s	30	G(pl > p)
Horizonte	Step de avaliação	Runs	Propriedade LTL

Desafio: O líder nasce próximo do seguidor mas ainda numa distância tecnicamente segura. Um Monte Carlo aleatório raramente acerta nesta zona crítica, precisa de guia.

Resultados Comparativos

Metric	Monte Carlo (Random Sampling)	Hill Climbing (Local Search)
Total Execution Time	329 ms	6326 ms
Total Simulations Run	30	58
Falsification Rate	0% (0/30)	90% (27/30)

Figura 5 - Benchmark de desempenho entre os algoritmos Monte Carlo e Hill Climbing

0%
Monte Carlo - nenhuma falha detetada

90%
Hill Climbing - 27 em 30 runs falsificados

Análise do “Plateau Effect”

Porque razão o Monte Carlo falhou e o Hill Climbing precisou de 58 simulações?

Monte Carlo - Pesquisa Cega

O espaço de parâmetros (p *x* pl) é imenso. A zona de falha da 4ª Definição é uma fatia estreita e matematicamente precisa.

A pesquisa aleatória fica “cega” neste plateau: Dentro dos limites seguros, todos os pontos aleatórios têm robustez positiva, o algoritmo não tem sinal que o guie para a fronteira.

→ Resultado: 0/30 colisões detetadas

Hill Climbing - Exploração Dirigida

Com 58 simulações (≈ 6 s), o algoritmo:

- 1. Testa a vizinhança para medir a inclinação da robustez
- 2. Navega iterativamente para a fronteira da 4ª Definição
- 3. Encontra exatamente os parâmetros que causam colisão

O custo computacional extra é o preço da precisão:
→ 90% de taxa de falsificação (27/30)

Demonstração Lince 2.0

Conclusões

- **Robustez ≠ Imunidade**: A 4ª Definição Matemática é a mais robusta, mas foi falsificada. Nenhum controlador está imune a condições de fronteira quando testado sistematicamente.

- **Pesquisa Guiada vs Pesquisa Cega:** Hill Climbing (90%) é dramaticamente superior ao Monte Carlo (0%) em espaço de parâmetros com zonas de falhas estreitas. O gradiente de robustez é a chave.

- **Integração no Simulador:** A função **overrideAssignments** e o parser LTL estendem o Lince 2.0 com falsificação avançada sem comprometer a integridade estrutural do simulador.

A falsificação em simulação é uma alternativa escalável e de alta confiança à verificação formal exaustiva em CPS críticos.

Trabalho Futuro

Duas direções promissoras para expandir o motor de falsificação

01 - Agentes de Reinforcement Learning

Substituir o Hill Climbing por agentes de RL (DQN/DDPG) que aprendem com o histórico de contraexemplos anteriores.

Benefícios Esperados:

- Redução do orçamento de simulações necessárias
- Generalização para novos cenários sem reparametrização
- Exploração adaptativa do espaço de falhas

02 - Sistemas CPS em Rede com Falhas

Aplica o motor de falsificação a sistemas ciber-físicos em rede, onde **delays de comunicação e packet loss** adicionaram novas dimensões de incerteza.

Desafios a explorar:

- Modelação de latências variáveis na AST do Lince
- Falsificação com perturbações da rede como parâmetros
- Validação em sistemas de vários veículos (platoon)

Questões?

4
Definições de “safe” testadas

2
Algoritmos implementados

90%
Taxa de falsificação HC

0%
Taxa MC (baseline)