Mestrado em Engenharia Informática,
Universidade do Minho

Falsification of Hybrid Programs

PROJETO EM MÉTODOS FORMAIS
DE PROGRAMAÇÃO

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

Recap do Problema e Objetivos do Milestone

O problema:

A complexidade dos sistemas híbridos (combinam eventos discretos com a dinâmica contínua do tempo) torna a verificação exaustiva impraticável em muitos cenários "black-box".

A solução:

Em vez de provar que o sistema está sempre certo, procuramos ativamente por um contra-exemplo que prove que ele falha, ou seja, esta solução é a falsificação.

Objetivos para o M2:

Dotar o simulador Lince de uma linguagem de propriedades (LTL) e de um motor de verificação (Random Sampler), de forma a automatizar esta busca por contra-exemplos.

Metodologia de Otimização Black-Box

O que significa “Black-Box”?
O SUT é tratado como uma função cujos detalhes internos são desconhecidos, ou seja, onde apenas observamos entradas e saídas.

Função de Custo e Robustez:
Em vez de um resultado binário, utilizámos uma métrica de robustez. A função de custo “mede” o quão sistema este perto de violar a propriedade de segurança.

Ciclo de Falsificação Adversária:
Amostragem: o algoritmo recolhe uma entrada x;
Simulação: executa o programa com essa entrada;
Avaliação: o verificador calcula a robustez da trajetória gerada;
Otimização: o algoritmo utiliza o valor da robustez para “guiar” a próxima procura para zonas mais perigosas.

Objetivos Alcançados: Implementação

Gramática LTL:
Implementação da gramática da linguagem de propriedades.

Operadores:
Atómicos - permite usar comparações já existentes no simulador (ex: pl>pf)
Temporais - permite verificar a temporabilidade dos programas
Lógicos - permite usar operadores lógicos

Vantagem:
Estrutura recursiva que permite aninhar propriedades complexas

```
enum LTL:
  case Atomic(c: Cond)
  case Not(phi: LTL)
  case And(phi1: LTL, phi2: LTL)
  case Or(phi1: LTL, phi2: LTL)
  case Implies(phi1: LTL, phi2: LTL)
  case Equiv(phi1: LTL, phi2: LTL)
  case Globally(phi: LTL)
  case Finally(phi: LTL)
  case Next(phi: LTL)
  case Until(phi1: LTL, phi2: LTL)
  case Release(phi1: LTL, phi2: LTL)
```

Figura 1 - Gramática implementada

Objetivos Alcançados: Implementação

Parser:
Transforma strings de texto em objetos da gramática definida.

Camadas:
Atomic - trata parênteses e condições base;
Unary - processa operadores de prefixo (G, F, X, !);
Temporal - lida com operadores binários (until e release);
Logic - lida com conectores lógicos (&, |, =>).

```
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 2 - Parte do Parser Implementado

Novas Funcionalidades do Lince

Range:

- Define o intervalo de valores iniciais para uma variável
- Permite explorar variações nas condições iniciais de execução
- Gera múltiplas execuções do sistema (uma por cada valor)
- **Propósito**: limitar o espaço de verificação e tornar a análise mais eficiente

Property:

- Representa a fórmula LTL introduzida pelo utilizador
- Define o comportamento esperado do sistema
- Permite expressar propriedades de:
  - Safety → algo de errado nunca acontece
  - Liveness → algo desejável eventualmente acontece
- **Propósito**: especificar formalmente o critério de verificação

```
// Adaptive Cruise Control (ACC)
p:=0; v:=0;  // follower
pl:=50; vl:=10; // leader
a:=0;
while true {
  // decide to speed up (acc=2) or brake
  if (v-8)^2 + 4*(p-pl+v-9) < 0
  then p'=v, v'= 2, pl'=vl, vl'=a for 1;
  else p'=v, v'=-2, pl'=vl, vl'=a for 1;
}
----
until 100
vars p.*, v.*
range pl 10
property G(pl > p)
```

Figura 3 - Exemplo de Range e de Property

Verificação de propriedades sobre o trace

Input:

- **Property:** fórmula LTL definida pelo utilizador
- **Trace:** sequência temporal de estados

$$[(t0, v0),(t1,v1),...,(tn,vn)]$$

Passo 1 - Avaliação Local(evaluate):

Verifica se a propriedade é satisfeita num estado específico:

- Atomic → avalia condições sobre variáveis
- And/Or/Not → operadores lógicos
- Implies/Equiv → relações lógicas

```
case LTL.And(p1,p2) =>
  evaluate(p1,v) && evaluate(p2,v)
```

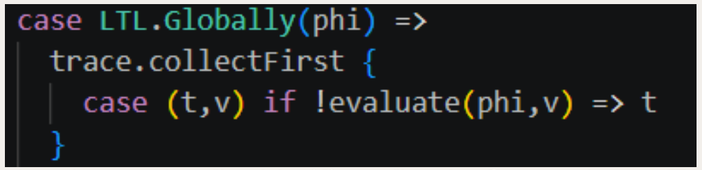
Figura 4 - Implementação: Avaliação de Fórmulas Lógicas

Verificação de propriedades sobre o trace

Passo 2 - Verificação Temporal (checkViolation):

Percorre o trace e verifica operadores temporais:

- G Ψ → procura o primeiro instante onde Ψ falha
- F Ψ → verifica se Ψ ocorre em algum instante
- Ψ1 U Ψ2 → garante Ψ1 até Ψ2 acontecer
- Ψ1 R Ψ2 → garante Ψ2 até Ψ1 acontecer (ou Ψ2 é sempre verdadeiro)



```
case LTL.Globally(phi) =>
  trace.collectFirst {
    case (t,v) if !evaluate(phi,v) => t
  }
```

Figura 5 - Implementação: Verificação de Operadores Temporais

Output:

- None → propriedade satisfeita
- Some(t) → violação detetada no instante t

Validação sobre intervalos de entrada

Objetivo: Verificar automaticamente se diferentes valores iniciais de uma variável podem levar à violação de uma propriedade.

Processo:

- Identificação da variável inicial x:= i (pl := 50)
- Geração do intervalo de teste:
  - $[i - c, ..., i, ..., i + c]$
- Execução da simulação para cada valor
- Geração do respetivo trace
- Verificação da propriedade LTL

Output:

- Valores que satisfazem a propriedade
- Valores que violam a propriedade
- Instante temporal da violação

```
property G(pl < p)
```

```
Análise da variável pl no intervalo [47, 53]
---------------
pl=47: Propriedade não satisfeita. Violação no instante t = 0.000s
pl=48: Propriedade não satisfeita. Violação no instante t = 0.000s
pl=49: Propriedade não satisfeita. Violação no instante t = 0.000s
pl=50: Propriedade não satisfeita. Violação no instante t = 0.000s
pl=51: Propriedade não satisfeita. Violação no instante t = 0.000s
pl=52: Propriedade não satisfeita. Violação no instante t = 0.000s
pl=53: Propriedade não satisfeita. Violação no instante t = 0.000s
```

Figura 6 - Violação Detetada (Some(t))

```
property G(pl > p)
```

```
Análise da variável pl no intervalo [47, 53]
---------------
pl=47: propriedade satisfeita
pl=48: propriedade satisfeita
pl=49: propriedade satisfeita
pl=50: propriedade satisfeita
pl=51: propriedade satisfeita
pl=52: propriedade satisfeita
pl=53: propriedade satisfeita
```

Figura 7 - Propriedade Satisfeita (None)

Calendarização dos Próximos Passos

1ª Semana	2ª Semana	3ª Semana
Começar a implementação do algoritmo de otimização **Hill Climbing.**	Finalizar a implementação do algoritmo de otimização **Hill Climbing** e fazer testes no caso de estudo do ACC.	Finalizar a análise comparativa entre **Random Sampler** e **Hill Climbing** e finalizar o relatório.