UC Projeto
3º ano Licenciatura em Ciências da Computação
Construção de um ferramenta genérica de verificação SAT para propriedades de segurança e animação de sistemas de transição de 1º ordem (FOTS)

Alef Keuffer (A91683)

Alexandre Baldé (A70373)

Bruno Machado (A91680)

Pedro Pereira (A88062)

Supervisor: Professor José Manuel Esgalhado Valença

21 de junho de 2022

Resumo

Neste relatório apresenta-se o trabalho realizado para a UC de Projeto, que consistiu no desenvolvimento, em Python[1], de uma ferramenta genérica de verificação de propriedades de "*First-Order Transition Systems*" através de uma biblioteca-interface para SMT-LIB.

Para a verificação de correção de propriedades de segurança e animação de FOTS, foram analisados os seguintes métodos: "*Bounded Model-Checking*", "*k-induction*", "*Property Directed Reachability*", "*Interpolant-Based Model-Checking*".

[1] `https://www.python.org/`

Conteúdo

Lista de Figuras

List of Listings

Capítulo 1

Introdução

Este relatório contém a descrição do projeto realizado pelos autores para a UC de Projeto da Licenciatura em Ciências da Computação, para o ano letivo de 2021/2022.

1.1 Estrutura do Relatório

A estrutura do relatório é a seguinte:

- No capítulo **??** faz-se uma análise do trabalho já existente na área, e das referência usadas para o projeto.
- No capítulo **??** explicam-se alguns aspetos mais técnicos e concretos da implementação, assim como decisões tomadas e alternativas consideradas.
- No capítulo **??** apresenta-se um case de estudo com um FOTS que servirá para apresentação das funcionalidades desenvolvidas.
- No capítulo **??** termina-se o relatório com as conclusões e o trabalho futuro.
- Nos anexos **??** e **??** , encontra-se informação relativa ao código Python desenvolvido, assim como o repositório GitHub que o contém, e como utilizar o código.

1.2 Problema em análise

Parte da verificação formal de "*software*" prende-se com, se possível, extrair garantias de segurança ou animação de programas, e se impossível, obter contra-exemplos que demonstrem a insegurança do sistema.

Neste projeto, o objeto de estudo serão "*First-Order Transition Systems*", que podem ser codificados através de lógica de primeira ordem, sendo então representáveis e manipuláveis através de "*solvers*" — este tema já foi abordado na UC de Lógica Computacionalem [**?**], pelo que para evitar repetição, utilizar-se-ão referências ao material da UC.

Existe um vasto corpo de conhecimento relativo à verificação de propriedades de "*SMT solvers*", que inclui vários métodos diferentes para efetuar este tipo de provas — ver **??** para uma breve exploração das referências atuais no campo.

O objetivo deste projeto foi implementar uma ferramenta que permitisse a verificação de propriedades de "*First-Order Transition Systems*" através de alguns desses métodos.

Para implementar a ferramenta de verificação, utilizou-se a linguagem de programação Python, e a uma biblioteca de "*SMT solvers*" disponível **PySMT**[1].

1.3 Resolução e Estratégias adotadas

Para resolver o problema proposto, e implementar a ferramenta, consideraram-se as seguintes estratégias:

1.3.1 "k*-induction*" e "*Bounded Model-Checking*"

- As versões básicas destas duas técnicas foram estudadas e implementadas pelas autores após frequência da UC de Lógica Computacionalno ano letivo de 2021/2022.
- Logo, na ferramenta deste projeto utiliza-se a implementação desenvolvida pelos autores nessa UC, com auxílio dos docentes.

1.3.2 "*Interpolant-Based Model-Checking*"

Para implementar esta técnica, consideraram-se duas formas principais:

- Uma permite provar propriedades sobre "*First-Order Transition Systems*" arbitrários, e requer o uso do teorema do interpolante de Craig[2]
- Outra possibilidade que se considerou foi converter "*First-Order Transition Systems*" para um programa na linguagem de fluxos abordada na UC de Lógica Computacional— apenas quando fosse possivel, porque nem sempre o será —, e depois utilizar as noções de WPC/SPC como interpolante de fórmulas
- O item acima não foi completado, mas considerou-se também, caso tivesse sido, implementar uma versão do "*Interpolant-Based Model-Checking*" que utilizasse ambas técnicas em simultâneo de forma dinâmica, consoante as características do "*First-Order Transition System*", com o propósito de melhorar o desempenho do método

Linguagem de representação para "*First-Order Transition Systems*"

Uma das referências que se considerou para implementar "*Interpolant-Based Model-Checking*" foi [**?**] — ver **??**. Aí, utiliza-se a ferramenta CPAChecker, que possui uma linguagem própria para a definição de FOTS [3].

Para este projeto, considerou-se a implementação de uma linguagem própria semelhante à usada pelo CPAChecker para representar FOTS, recorrendo à análise semântica do FOTS para verificar se é possível convertê-lo para linguagem de fluxos.

Esta ideia foi discutida e fragmentos de um protótipo estão identificados no projeto; em última instância não se completou a implementação devido a restrições temporais.

[1] Documentação disponível em `https://pysmt.readthedocs.io/en/latest/`

[2] `https://en.wikipedia.org/wiki/Craig_interpolation`

[3] veja-se um exemplo em `https://gitlab.com/sosy-lab/software/cpachecker/-/blob/trunk/config/specification/TerminatingStatements.spc`

1.3.3 “*Property Directed Reachability*”

A técnica do PDR foi abordada brevemente na UC de LC.
Em suma, consideraram-se duas abordagens.

- A primeira consistiu em seguir [**?**], e reimplementar a noção de “*induction-guided abstraction-refinement*”, com a implementação em Java deste método servindo de referência. Optou-se por não terminar esta a favor da seguinte, dada a complexidade da implementação-guia, e o tempo disponível.
- A segunda foi baseada em [**?**], que é mais simples por ser uma extensão da k-indução.

1.4 Agradecimentos

Capítulo 2

Estado de arte

"k-*induction*" e "*Bounded Model-Checking*"

Explorar: [?] [?].

"*Interpolant-Based Model-Checking*"

Explorar: [?] [?] [?].

"*Property Directed Reachability*"

Explorar: [?] [?] [?].

Capítulo 3

Análise do trabalho

3.1 “*k-induction*” e “*Bounded Model-Checking*”

3.2 “*Interpolant-Based Model-Checking*”

Como foi descrito acima, considerou-se a implementação da interpolação presente em [?], pp. 38. Em seguida está o pseudocódigo presente na citação acima:

Input: Transition system (S,T), property P
Output: SAFE or UNSAFE and a counterexample
Data: k: bound, $R(i)$: overapproximation of states at distance at most i from S, $I(i)$: interpolant

begin
    if $S \wedge \neg P$ *is satisfiable* **then** return UNSAFE, counterexample
    $k \leftarrow 1, i \leftarrow 0$
    $R(i) \leftarrow S$
    while true **do**
        $A \leftarrow R(i) \wedge T^0$
        $B \leftarrow \bigwedge_{j=1}^{k-1} T^j \wedge \bigvee_{l=0}^{k} \neg P^l$
        if $A \wedge B$ *is satisfiable* **then**
            if $R(i) = S$ **then** return UNSAFE, counterexample
            else
                $k \leftarrow k+1, i \leftarrow 0$
                $R(i) \leftarrow S$
        else
            $I(i) \leftarrow Itp(A,B)$
            if $I(i) \models R(i)$ **then** return SAFE
            else
                $R(i+1) \leftarrow R(i) \vee I(i)$
                $i \leftarrow i+1$
    end
end

Figura 3.1: Pseudocódigo para IMC retirado de [?]

Considere-se agora a implementação quase direta em Python, através do PySMT:

```python
def IMC(TS: TransitionSystem,
        P: FNode,
        S=None,
        customInterpolator=False):

    if not S:
        S = TS.init

    if m := get_model(S & Not(P)):
        print(m)
        return Status.UNSAFE1

    k = 1

    i = 0
    R_i = S.substitute(TS.get_subs(i))
```

```python
while True:
    A = R_i & TS.get_unrolling(1)
    B = And(And(TS.get_unrolling(1, k)), Or(get_unrolling(Not(P), k)))
    if m := is_sat(A & B):
        if is_valid(R_i.EqualsOrIff(S)):
            print(m)
            return Status.UNSAFE2
        else:
            k += 1
            i = 0
            R_i = S.substitute(TS.get_subs(i))
    else:
        if customInterpolator:
            I_i = bin_itp(A, B)
        else:
            I_i = binary_interpolant(A, B)

        if is_valid(I_i.Implies(R_i)):
            print(f"Proved at step {i + 1}")
            return Status.SAFE
        else:
            R_i = R_i | I_i
            i += 1
```

Excerto de Código 3.1: Implementação em PySMT do algoritmo de IMC retirado de [?]

A versão com comentário encontra-se no anexo em **??**, ou no GitHub com o resto do código fonte, também no anexo **??**.

Interpolante de Craig, escolha de lógica SMT

Para implementar este algoritmo, foi necessário utilizar o teorema de interpolação de Craig[1], que já estava implementado no PySMT.

É preciso notar que a implementação do PySMT não suporta todas as lógicas SMT-LIB. Por exemplo, no caso do estudo em **??**, a lógica

[1] https://en.wikipedia.org/wiki/Craig_interpolation

3.3 “*Property Directed Reachability*”

Capítulo 4

Caso de Estudo

Para testar a ferramenta desenvolvida, escolheu-se um exemplo simples mas não trivial de um "*First-Order Transition System*" visto num dos trabalhos práticos da UC de Lógica Computacional.
Considere-se o seguinte problema:

Trabalho 4

> *Todos os problemas deste devem ser resolvidos usando **pySMT** e SMT's que suportem BitVec*

Considere o seguinte programa, em Python anotado, para multiplicação de dois inteiros de precisão limitada a 16 bits.

```
     assume m >= 0 and n >= 0 and r == 0 and x == m and y == n
  0: while y > 0:
  1:    if y & 1 == 1:
              y , r  = y-1 , r+x
  2:    x , y = x<<1  ,  y>>1
  3: assert r == m * n
```

1. Prove por indução a terminação deste programa

2. Pretende-se verificar a *correção total* deste programa usando a metodologia dos invariantes e a metodologia do "single assignment unfolding". Para isso,
   a. Codifique usando a LPA (linguagem de programas anotadas) a forma recursiva deste programa.
   b. Proponha o invariante mais fraco que assegure a correção, codifique-o em SMT e prove a correção.
   c. Construa a definição iterativa do "single assignment unfolding" usando um parâmetro limite N e aumentando a pré-condição com a condição
   $$(n < N) \land (m < N)$$
   O número de iterações vai ser controlado por este parâmetro N

Figura 4.1: FOTS de trabalho 4 de Lógica Computacional, ano letivo 2021/2022

Abaixo está uma possível codificação do problema acima.

```
def trab4FinalSimplification(bit_count):
    from pysmt.typing import BVType

    # Variables
    x = Symbol("x", BVType(bit_count))
    m = Symbol("m", BVType(bit_count))
    n = Symbol("n", BVType(bit_count))
    y = Symbol("y", BVType(bit_count))
    r = Symbol("r", BVType(bit_count))
    pc = Symbol("pc", BVType(bit_count))

    npc = next_var(pc)
    ny = next_var(y)
    nx = next_var(x)
    nr = next_var(r)

    from util.transition import TransitionPredicate

    T = TransitionPredicate()

    # pc = 0  y > 0 -> pc = 1                  enters WHILE
    T.add(pc.Equals(0) & (y > 0), npc.Equals(1))
    # pc = 0  y  0 -> pc = 3                  doesn't enters WHILE
    T.add(pc.Equals(0) & (y <= 0), npc.Equals(3))
    # pc = 1  y & 1 = 1 -> y = y  1  r = r + x  pc = 2      IF condition is true
    T.add(pc.Equals(1) & (y & 1).Equals(1), npc.Equals(2) & ny.Equals(y - 1) & nr.Equals(r + x))
    # pc = 1  y & 1  1 -> pc = 2                          IF condition is false
    T.add(pc.Equals(1) & (y & 1).NotEquals(1), npc.Equals(2))
    # pc = 2 -> y = y >> 1  x = x << 1  pc = 0      loop back after attributions
    T.add(pc.Equals(2), npc.Equals(0) & nx.Equals(x << 1) & ny.Equals(y >> 1))
    # pc = 3          end of program
    T.add(pc.Equals(3))

    pre = ((m >= 0) &  # m  0
           (n >= 0) &  # n  0
           r.Equals(0) &  # r = 0
           x.Equals(m) &  # x = m
           y.Equals(n)  # y = n
           )

    init = pc.Equals(0) & pre
    return TransitionSystem(init, T.get())
)
```

Excerto de Código 4.1: Representação em PySMT de FOTS em ??

Note-se que é possível definir um FOTS correspondente a um sistema de várias formas. Neste caso escolheu-se introduzir a variável pc de forma a usar as técnicas estudadas para fazer provas que envolvessem terminação e animação.

Capítulo 5

Conclusão

Conclui-se desta forma a apresentação do trabalho desenvolvido pelos autores para a UC *Projeto* no ano letivo 2021/2022.

5.1 Comentários

5.2 Trabalho Futuro

Apêndice A

Excertos de Código Utilizado no Projeto

```python
def IMC(TS: TransitionSystem,
        P: FNode,
        S=None,
        customInterpolator=False):

    if not S:
        S = TS.init

    # first makes sure P is not violated by S
    if m := get_model(S & Not(P)):
        # halt return a counterexample
        print(m)
        return Status.UNSAFE1

    # bound
    k = 1

    # overapproximation of states at distance at most i from S
    i = 0
    R_i = S.substitute(TS.get_subs(i))

    # for a bound k and a current overapproximation R(i) of the states at distance at
    # most i from S, the algorithm checks if P is violated by the states reachable
    # from R(i) in at most k steps.
    while True:
        A = R_i & TS.get_unrolling(1)
        B = And(And(TS.get_unrolling(1, k)), Or(get_unrolling(Not(P), k)))
        if m := is_sat(A & B):
            # the error might be real or spurious, caused by an insufficient value of k
            if is_valid(R_i.EqualsOrIff(S)):
                # error is real so the system is unsafe
                print(m)
                return Status.UNSAFE2
            else:
```

```
        # error is spurious so k is increased to allow finer
        # overapproximations, and the algorithm restarts from S.
        k += 1
        i = 0
        R_i = S.substitute(TS.get_subs(i))
# R(i) _{j=0}^{k1} T^j _{l=0}^k ¬P^l is unsat
else:
    # an interpolant I(i) is computed, which represents an approximation of the
    # image of R(i) (i.e., of the states reachable from R(i) in one step).
    if customInterpolator:
        I_i = bin_itp(A, B)
    else:
        I_i = binary_interpolant(A, B)

    # a fixpoint check is carried out: if I(i) |= R(i), it means that all
    # states have been covered, and the system is safe; otherwise, R(i + 1) is
    # set to R(i)  I(i) and the procedure continues.
    if is_valid(I_i.Implies(R_i)):
        # the current R(i) corresponds to an inductive invariant P stronger
        # than P: on one side, S |= R(i), moreover R(i)  T |= I'(i) and I(i)
        # |= R(i) imply R(i)  T |= R'(i); on the other side, the fact that at
        # each iteration 0  h  i, R(h)  _{j=0}^{k1} T |= _{l=0}^k P^l,
        # together with R(i) being an inductive invariant, yield R(i) |= P.
        print(f"Proved at step {i + 1}")
        return Status.SAFE
    else:
        R_i = R_i | I_i
        i += 1
```

Excerto de Código A.1: Implementação comentada do algoritmo de IMC -[?]

Apêndice B

Repositório *GitHub* com código fonte e documentação

"*Source code*"

O código fonte da ferramenta desenvolvida é accessível através do link: `https://github.com/Alef-Keuffer/FOTS-Prover`.
Está sediado numa página de GitHub de um dos autores.

Documentação da ferramenta

A documentação da ferramente encontra-se disponível aqui.