Why3: Lógica

Why3 can be used in many different ways. Here we look at it from a purely logic perspective.

The Why3 logic language extends first-order logic with

- user-defined types, including algebraic (inductive types), and polymorphic types
- defined predicates and functions, including inductive and polymorphic definitions
- *inductive predicates*, defined in a closed way by a set of inference rules
- a rich logic library of theories, including (but not only) types that are particularly suited in the context of computer programming

The result is a language that is particularly suitable to encode many typical useful notions in computer science, for instance in programming language semantics.

Why3 is *not* a proof tool. Instead it interacts with external proof tools, and this is in fact one of its strongest points: it is able to interact with virtually every interesting SMT solver, automated prover, and proof assistant (including Coq and Isabelle).

Why3 does include a collection of internal proof transformations that can be used on goals before invoking proof tools (including transformations for inductive proofs), but this does not imply that it is meant as a proof tool to be used on its own.

Finally, the Why3 IDE, although not required, is an extremely useful tool for managing proof developments, including storing and replaying proof sessions.

hello_proof.why

```
theory HelloProof
  goal G1 : true
```

```
  goal G2 : (true -> false) /\ (true \/ false)

  use int.Int
  goal G3: forall x:int. x*x >= 0
end

```

Esta primeira teoria contém objectivos de prova triviais e ilustra também a utilização da teoria de inteiros do Why3.

1. Tente provar todos os objectivos com um qualquer solver.
   A tarefa de prova corrente pode ser visualizada a qualquer momento no tab "Task"

2. É evidente que um deles não será nunca provado porque não é válido. No entanto, trata-se da conjunção de duas fórmulas, uma das quais é válida. Isole e prove a parte válida do objectivo utilizando a transformação "split"

First.why

```
theory First
  type t
  constant c : t
  predicate a
  predicate p t
  predicate q t
  function f (t) : t

  goal P1 : (forall x:t. p(x)) -> (exists x:t. p(x))
  goal P5 : (forall x:t. p(x) -> p(f(x))) -> forall x:t.p(x) -
> p(f(f(x)))
end

```

Esta teoria ilustra a utilização de predicados, funções não interpretadas, e quantificadores.
Tente provar ambos os objectivos com um SMT solver

genealogy.why

```
1
2  theory Genealogy
3    type person
4    type gender = Male | Female
5    function gender person : gender
6    function father person : person
7    function mother person : person
8
9    axiom Father_gender : forall p : person. gender (father p) =
   Male
10   axiom Mother_gender : forall p : person. gender (mother p) =
   Female
11
12   predicate parent (p : person) (c : person) = p = father c \/
   p = mother c
13   predicate son (s : person) (p : person) = gender s = Male /\
   parent p s
14   predicate daughter (d : person)(p : person) = gender d = Fem
   ale /\ parent p d
15   predicate child (c : person) (p : person) = parent p c
16
17   goal Child_is_son_or_daughter:
18     forall c p : person. child c p <-> son c p \/ daughter c p
19
20   predicate sibling (p1 : person) (p2 : person) =
```

```
    p1 <> p2 /\ (father p1 = father p2 \/ mother p1 = mother p2)
  goal Sibling_sym : forall p1 p2 : person. sibling p1 p2 -> sibling p2 p1

  predicate brother (b : person) (p : person) = sibling b p /\ gender b = Male
  predicate sister  (s : person)(p : person) = sibling s p /\ gender s = Female

  goal Sibling_is_brother_or_sister:
    forall p1 p2 : person. sibling p1 p2 <-> brother p1 p2 \/ sister p1 p2

end

```

Esta teoria ilustra a utilização de predicados definidos e axiomas envolvendo funções; usa também o predicado de igualdade.

1. Estude atentamente a teoria e prove os objectivos contidos no ficheiro.
2. Observe que a teoria define funções `father` e `mother`, e um predicado `parent` (pai ou mãe). Acrescente agora à teoria a definição de 3 novos predicados `grandfather`, `grandmother`, e `grandparent` , utilizando as funções e predicado referidos.
3. Prove depois o seguinte:
   a. ser `grandparent` de alguém é equivalente a ser seu avô ou avó
   b. qualquer avô (resp. avó) é do sexo masculino (resposta. feminino)
   c. ninguém tem mais do que dois avôs (`grandfather`)

Guest Placement

Recorde o problema da distribuição de 3 pessoas por 3 cadeiras, tal que:

- Anne does not want to sit near Peter.
- Anne does not want to sit in the left chair.
- Susan does not want to sit to the right of Peter.

A que acrescem as restrições

- Todas as pessoas devem ficar sentadas
- Não mais do que uma pessoa se poderá sentar em cada cadeira

O Why3 não foi concebido para resolver problemas de satisfazibilidade, mas podemos utilizar a ferramenta para esse fim. Para testar se um conjunto de axiomas é satisfazível, basta incluir o objectivo de prova `false`: se for provado, isto significa que o modelo é insatisfazível.

Complete então o seguinte ficheiro, escrevendo os axiomas necessários para o predicado `sits` (pode utilizar para isto quantificadores).

[guestPlacement.why]

```
theory GuestPlacement
  type person = Anne | Susan | Peter
  type chair = Left | Middle | Right
  predicate sits person chair

  (* alternativa: function sits person : chair *)

  (* COMPLETAR *)

  goal PlacementNotPossible: false

end
```

1. Interprete os resultados obtidos para `PlacementNotPossible`

2. Remova agora os axiomas relativos às preferências de Susan e repita a prova. Como interpreta o que observa?

3. Por forma a visualizar um modelo é possível recorrer à funcionalidade de geração de contra-exemplos, suportada pelo Why3 em interacção com os solvers que devolvem modelos (é o caso do Z3 e do CVC4). Para isto é necessário incluir objectivos de prova adicionais para este efeito, por exemplo:

```
goal sitsSusan: forall c :chair. not sits Susan c
```

   Este objectivo de prova não é válido se existir uma solução para o problema, e um contra-exemplo indicará a cadeira onde se sentará Susan nessa solução. Escreva um objectivo de prova que lhe permita visualizar uma solução completa para o problema, e utilize a opção `getCounterexamples` (tecla G) para a obter.

4. Repita a axiomatização e processo de prova, agora utilizando uma função `sits` em vez de um predicado.

Do it Yourself : *Scottish Private Club*

Um clube privado é definido pelas seguintes regras:

- every non-Scottish member wears red socks
- every member wears a kilt or doesn't wear socks
- the married members don't go out on sunday
- a member goes out on sunday if and only if he is Scottish
- every member who wears a kilt is Scottish and married
- every Scottish member wears a kilt

Pretende-se formalizar este *puzzle* em Why3. Para isso:

1. Escreva uma formalização *proposicional* e mostre que o clube não pode ter quaisquer membros.
2. Elimine agora uma das regras, que torne possível que o clube admita membros. Utilize a funcionalidade de obtenção de modelos para caracterizar um possível membro do clube.
3. Escreva agora uma formalização de primeira ordem, definindo um tipo para pessoas e predicados sobre esse tipo (*scottish*, *married*, etc). Defina também um predicado para exprimir que uma pessoa pertence ao clube.
4. Elimine novamente a regra que eliminou no ponto 2. Observe que agora não é

possível visualizar um modelo correspondente a uma solução do problema.

5. É possível "povoar" o tipo das pessoas. Inclua no modelo uma pessoa chamada Alan, e:
   a. tente de novo obter o modelo pedido no ponto 4
   b. inclua um objectivo de prova correspondente à pertença de Alan ao clube
   c. como se pode interpretar o facto de Alan ser um modelo em a., não sendo no entanto provado o objectivo de prova b.?