Correcção em Why3 de algoritmos directos, CPS e desfuncionalizados de transformação de fórmulas para Forma Normal Conjuntiva

Pedro Barroso

Faculdade de Ciências e Tecnologia da Universidade NOVA de Lisboa

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

8 de Maio de 2019

Estilo direto

Estilo direto

CPS

- Possibilidade de executar o algoritmo passo-a-passo.
- Evitar o overflow da stack.

Tranformação em CPS

1.

Desfuncionalização

Obrigado

Apêndice

Código Why3

Módulo Fórmula

Código Why3

```
module Formula

  use list.List

  type ident

  type formula =
    | FVar   ident
    | FConst bool
    | FAnd   formula formula
    | FOr    formula formula
    | FImpl  formula formula
    | FNeg   formula

  type formula_wi =
    | FVar_wi ident
    | FConst_wi bool
    | FAnd_wi formula_wi formula_wi
    | FOr_wi formula_wi formula_wi
    | FNeg_wi formula_wi

end
```

Módulo Predicates

Código Why3

```
module Predicates

  use Formula

  predicate wf_negations_of_literals (f: formula_wi)
  = match f with
    | FNeg_wi f → (
      match f with
      | FOr_wi _ _ | FAnd_wi _ _ |  FNeg_wi _ → false
      | _ → wf_negations_of_literals f
      end)
    | FOr_wi f1 f2 | FAnd_wi f1 f2 → wf_negations_of_literals f1 ∧ wf_negations_of_literals f2
    | FVar_wi _ → true
    | FConst_wi _ → true
  end
```

Correcção em Why3 de algoritmos directos, CPS e desfuncionalizados de transformação de fórmulas para Forma Normal Conjuntiva - FCT-NOVA

Módulo Predicates


```
predicate wf_disjunctions (f: formula_wi)
= match f with
  | FAnd_wi _ _ → false
  | FOr_wi f1 f2 → wf_disjunctions f1 ∧ wf_disjunctions f2
  | FConst_wi _ → true
  | FVar_wi _ → true
  | FNeg_wi f1 → wf_disjunctions f1
end

predicate wf_conjunctions_of_disjunctions (f: formula_wi)
= match f with
  | FAnd_wi f1 f2 → wf_conjunctions_of_disjunctions f1 ∧ wf_conjunctions_of_disjunctions f2
  | FOr_wi f1 f2 → wf_disjunctions f1 ∧ wf_disjunctions f2
  | FConst_wi _ → true
  | FVar_wi _ → true
  | FNeg_wi f1 → wf_conjunctions_of_disjunctions f1
end
```

```
  constant f1 : formula_wi =
    FAnd_wi (FOr_wi (FConst_wi true) (FConst_wi false))
                (FOr_wi (FAnd_wi (FConst_wi true)(FConst_wi false)) (FConst_wi true))

  constant f2 : formula_wi =
    FAnd_wi (FOr_wi (FConst_wi true) (FConst_wi false))
                (FOr_wi (FAnd_wi (FConst_wi true)(FConst_wi false)) (FConst_wi true))

  constant f3 : formula_wi =
    FOr_wi f2 f1

  goal G : wf_conjunctions_of_disjunctions f3

end
```

Módulo Valuation

Código Why3

```
module Valuation

  use Formula, list.List

  type valuation = ident → bool

  let rec function eval (v: valuation) (f: formula) : bool
    variant { f }
  = match f with
      | FVar x      → v x
      | FConst b    → b
      | FAnd f1 f2  → eval v f1 && eval v f2
      | FOr f1 f2   → eval v f1 || eval v f2
      | FImpl f1 f2 → not (eval v f1) || eval v f2
      | FNeg f      → not (eval v f)
    end

  let rec function eval_wi (v: valuation) (f: formula_wi) : bool
    variant { f }
  = match f with
      | FVar_wi x      → v x
      | FConst_wi b    → b
      | FAnd_wi f1 f2  → eval_wi v f1 && eval_wi v f2
      | FOr_wi f1 f2   → eval_wi v f1 || eval_wi v f2
      | FNeg_wi f      → not (eval_wi v f)
    end
end
```

Módulo T

Código Why3

```
module T

  use Formula, Valuation, Predicates

  let rec impl_free (phi: formula) : formula_wi
    variant { phi }
    ensures { forall v. eval v phi = eval_wi v result }
  = match phi with
    | FNeg phi1 → FNeg_wi (impl_free phi1)
    | FOr phi1 phi2 → FOr_wi (impl_free phi1) (impl_free phi2)
    | FAnd phi1 phi2 → FAnd_wi (impl_free phi1) (impl_free phi2)
    | FImpl phi1 phi2 → FOr_wi (FNeg_wi (impl_free phi1)) (impl_free phi2)
    | FConst phi → FConst_wi phi
    | FVar phi → FVar_wi phi
    end
```

Correcção em Why3 de algoritmos directos, CPS e desfuncionalizados de transformação de fórmulas para Forma Normal Conjuntiva - FCT-NOVA

```
use int.Int

function size (phi: formula_wi) : int = match phi with
  | FVar_wi _ | FConst_wi _ → 1
  | FNeg_wi phi → 1 + size phi
  | FAnd_wi phi1 phi2 | FOr_wi phi1 phi2 →
      1 + size phi1 + size phi2
  end

let rec lemma size_nonneg (phi: formula_wi)
  variant { phi }
  ensures { size phi ≥ 0 }
= match phi with
  | FVar_wi _ | FConst_wi _ → ()
  | FNeg_wi phi → size_nonneg phi
  | FAnd_wi phi1 phi2 | FOr_wi phi1 phi2 →
      size_nonneg phi1; size_nonneg phi2
  end
```

```
let rec nnfc (phi: formula_wi)
  variant { size phi }
  ensures { (forall v. eval_wi v phi = eval_wi v result) ∧ wf_negations_of_literals result}
= match phi with
  | FNeg_wi (FNeg_wi phi1) → nnfc phi1
  | FNeg_wi (FAnd_wi phi1 phi2) → FOr_wi (nnfc (FNeg_wi phi1)) (nnfc (FNeg_wi phi2))
  | FNeg_wi (FOr_wi phi1 phi2) → FAnd_wi (nnfc (FNeg_wi phi1)) (nnfc (FNeg_wi phi2))
  | FOr_wi phi1 phi2 → FOr_wi (nnfc phi1) (nnfc phi2)
  | FAnd_wi phi1 phi2 → FAnd_wi (nnfc phi1) (nnfc phi2)
  | phi → phi
end
```

```
lemma aux: forall x. wf_conjunctions_of_disjunctions x ∧
  wf_negations_of_literals x ∧ not (exists f1 f2. x = FAnd_wi f1 f2) →
  wf_disjunctions x

let rec distr (phi1 phi2: formula_wi)
  requires{ wf_negations_of_literals phi1 ∧ wf_negations_of_literals phi2}
  requires{ wf_conjunctions_of_disjunctions phi1 ∧ wf_conjunctions_of_disjunctions phi2}
  variant { size phi1 + size phi2 }
  ensures { (forall v. eval_wi v (FOr_wi phi1 phi2) = eval_wi v result) } (* EVAL *)
  ensures { wf_negations_of_literals result ∧ wf_conjunctions_of_disjunctions result }
= match phi1, phi2 with
  | FAnd_wi phi11 phi12, phi2 → FAnd_wi (distr phi11 phi2) (distr phi12 phi2)
  | phi1, FAnd_wi phi21 phi22 → FAnd_wi (distr phi1 phi21) (distr phi1 phi22)
  | phi1,phi2 → FOr_wi phi1 phi2
  end
```

Correcção em Why3 de algoritmos directos, CPS e desfuncionalizados de transformação de fórmulas para Forma Normal Conjuntiva - FCT-NOVA

```
let rec cnfc (phi: formula_wi)
  requires{ wf_negations_of_literals phi }
  variant { phi }
  ensures{ (forall v. eval_wi v phi = eval_wi v result) ∧ wf_negations_of_literals result}
  ensures{ wf_conjunctions_of_disjunctions result}
= match phi with
  | FOr_wi phi1 phi2 → distr (cnfc phi1) (cnfc phi2)
  | FAnd_wi phi1 phi2 → FAnd_wi (cnfc phi1) (cnfc phi2)
  | phi → phi
  end

let t (phi: formula) : formula_wi
  ensures { (forall v. eval v phi = eval_wi v result) ∧ wf_negations_of_literals result ∧
              wf_conjunctions_of_disjunctions result}
= cnfc (nnfc (impl_free phi))
```

Correcção em Why3 de algoritmos directos, CPS e desfuncionalizados de transformação de fórmulas para Forma Normal Conjuntiva - FCT-NOVA

OCaml - Código extraído

Implementação OCaml

Código OCaml Extraído

```
type ident = string

type formula =
  | FVar of ident
  | FConst of bool
  | FAnd of formula * formula
  | FOr of formula * formula
  | FImpl of formula * formula
  | FNeg of formula

type formula_wi =
  | FVar_wi of ident
  | FConst_wi of bool
  | FAnd_wi of formula_wi * formula_wi
  | FOr_wi of formula_wi * formula_wi
  | FNeg_wi of formula_wi
```

Implementação OCaml

Código OCaml Extraído

```
let rec impl_free (phi: formula) : formula_wi =
  begin match phi with
  | FNeg phi1 -> FNeg_wi (impl_free phi1)
  | FOr (phi1, phi2) -> FOr_wi ((impl_free phi1), (impl_free phi2))
  | FAnd (phi1, phi2) -> FAnd_wi ((impl_free phi1), (impl_free phi2))
  | FImpl (phi1, phi2) -> FOr_wi ((FNeg_wi (impl_free phi1)), (impl_free phi2))
  | FConst phi1 -> FConst_wi phi1
  | FVar phi1 -> FVar_wi phi1
end

let rec nnfc (phi: formula_wi) : formula_wi =
  begin match phi with
  | FNeg_wi FNeg_wi phi1 -> nnfc phi1
  | FNeg_wi FAnd_wi (phi1, phi2) -> FOr_wi ((nnfc (FNeg_wi phi1)), (nnfc (FNeg_wi phi2))
  | FNeg_wi FOr_wi (phi1, phi2) -> FAnd_wi ((nnfc (FNeg_wi phi1)), (nnfc (FNeg_wi phi2))
  | FOr_wi (phi1, phi2) -> FOr_wi ((nnfc phi1), (nnfc phi2))
  | FAnd_wi (phi1, phi2) -> FAnd_wi ((nnfc phi1), (nnfc phi2))
  | phi1 -> phi1
end
```

```
let rec distr (phi1: formula_wi) (phi2: formula_wi) : formula_wi =
  begin match (phi1, phi2) with
  | (FAnd_wi (phi11, phi12), phi21) ->
    let o = distr phi12 phi21 in let o1 = distr phi11 phi21 in FAnd_wi (o1, o)
  | (phi11, FAnd_wi (phi21, phi22)) ->
    let o = distr phi11 phi22 in let o1 = distr phi11 phi21 in FAnd_wi (o1, o)
  | (phi11, phi21) -> FOr_wi (phi11, phi21)
end

let rec cnfc (phi: formula_wi) : formula_wi =
  begin match phi with
  | FOr_wi (phi1, phi2) -> let o = cnfc phi2 in let o1 = cnfc phi1 in distr o1 o
  | FAnd_wi (phi1, phi2) ->
    let o = cnfc phi2 in let o1 = cnfc phi1 in FAnd_wi (o1, o)
  | phi1 -> phi1
end

let t (phi: formula) : formula_wi = cnfc (nnfc (impl_free phi))
```

Implementação OCaml

Código OCaml Extraído

```
type valuation = ident -> (bool)

let rec eval (v: ident -> (bool)) (f: formula) : bool =
  begin match f with
  | FVar x -> v x
  | FConst b -> b
  | FAnd (f1, f2) -> (eval v f1) && (eval v f2)
  | FOr (f1, f2) -> (eval v f1) || (eval v f2)
  | FImpl (f1, f2) -> (not (eval v f1)) || (eval v f2)
  | FNeg f1 -> not (eval v f1)
  end

let rec eval_wi (v: ident -> (bool)) (f: formula_wi) : bool =
  begin match f with
  | FVar_wi x -> v x
  | FConst_wi b -> b
  | FAnd_wi (f1, f2) -> (eval_wi v f1) && (eval_wi v f2)
  | FOr_wi (f1, f2) -> (eval_wi v f1) || (eval_wi v f2)
  | FNeg_wi f1 -> not (eval_wi v f1)
  end
```

Correcção em Why3 de algoritmos directos, CPS e desfuncionalizados de transformação de fórmulas para Forma Normal Conjuntiva - FCT-NOVA