Capitolo 1

Implementazione di procedure di decisione per frammenti Binding in Vampire

L'algoritmo di decisione, la classificazione, Il preprocessing

1.1 Preprocessing

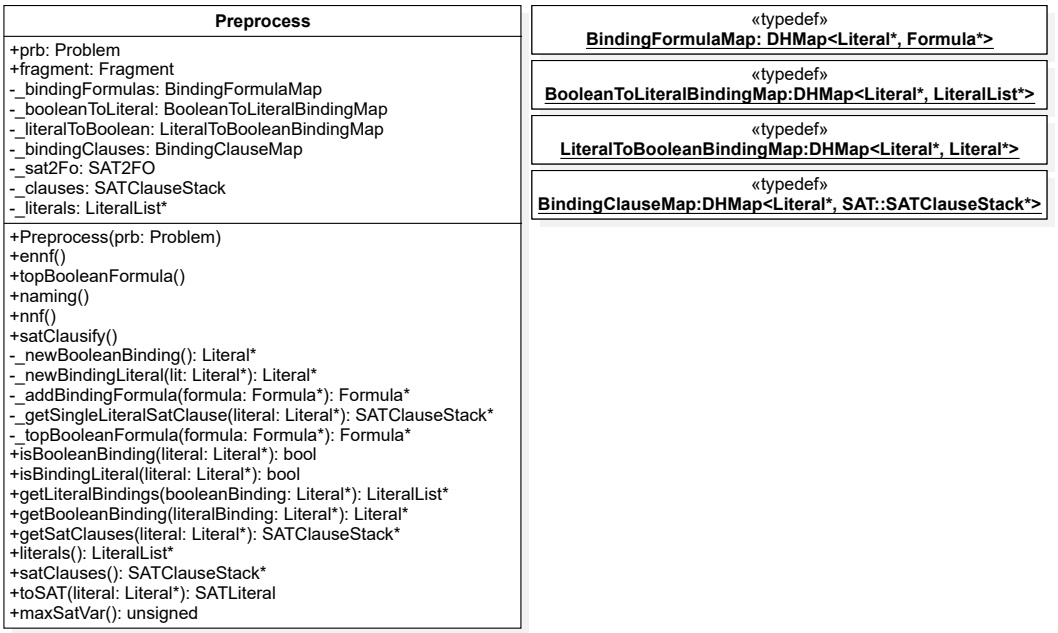

Figura 1.1: Struttura del Preprocessing

Algorithm 1: Top Boolean Formula

Firma: topBooleanFormula(φ)
Input: φ una formula rettificata
Output: Una formula ground
switch φ **do**
    case *Literal* l **do**
        return new AtomicFormula(l);
    end
    case $A[\wedge, \vee]B$ **do**
        return new JunctionFormula(topBooleanFormula(A), connective of φ,
         topBooleanFormula(B));
    end
    case $\neg A$ **do**
        return new NegatedFormula(topBooleanFormula(A));
    end
    case $[\forall, \exists]A$ **do**
        $b = newBooleanBinding()$;
        $bindingFormulas[b] := \varphi$;
        return new AtomicFormula(b);
    end
    case $A[\Leftrightarrow, \Rightarrow, \oplus]B$ **do**
        return new BinaryFormula(A, connective of φ, B);
    end
end

Dopo la nnf ci sono alcuni boolean Binding che sono stati negati. va aggiunta nella mappa la formula negata

foreach $l \in literals(\varphi)$ **do**
    if $\neg l.polarity()$ **then**
        continue
    end
    $positiveFormula := bindingFormulas[positiveLiteral(l)]$
    $bindingFormulas[l] := newNegatedFormula(positiveFormula)$
end

Mentre per ogni letterale ground di φ che non è un booleanBinding si aggiunge alla mappa bindingClauses la satClausola composta solamente dal satLetterale del letterale.

foreach $l \in literals(\varphi)$ **do**
    if l *is not a booleanBinding* **then**
        $bindingClauses[l] := newSatClause\{toSat(l)\}$
    end
end

A questo punto le formule della mappa bindingFormulas vanno trasformate in NNF,Skolemizzate e SatClausificate. Ogni booleanBinding è associato ad una formula del frammento ConjunctiveBinding, per questo dopo la skolemizzazione il quantificatore universale viene distribuito sull'and per ottenere le sottoformule del frammento OneBinding. Per ogni sottoformula OneBinding viene creato un nuovo letterale, che verrà chiamato LiteralBinding, in rappresentaza della sottoformula. Il nuovo letterale avrà gli stessi termini del letterale più a sinistra della sottoformula (che sono gli stessi di tutti i letterali

della sottoformula). Successivamente la formula viene SatClausificata. Si aggiunge alla mappa satClauses la coppia composta dal nuovo LiteralBinding e le satClausole della sottoformula. Alla mappa literalToBooleanBindings viene aggiunta la coppia composta dal nuovo LiteralBinding e il booleanBinding associato mentre alla mappa booleanBindingToLiteral viene aggiunta la coppia composta dal booleanBinding e la lista dei LiteralBinding che rappresentano le sottoformule della formula originale.

while *bindingFormulas* $\neq \emptyset$ **do**
    $(booleanBinding, formula) := bindingFormulas.pop()$
    $formula := nnf(formula)$
    $formula := skolemize(formula)$
    $toDo := \emptyset$

    if *formula is ConjunctiveBinding* **then**
        $formula := distributeForAll(formula)$
        "Add each subformula to the todo list"
    end
    else
        toDo.add(formula)
    end
    $literalBindings := \emptyset$ **while** *todo* $\neq \emptyset$ **do**
        $subformula := todo.pop()$
        $literalBinding := newLiteralBinding(subformula.mostLeftLiteral())$
        $clauses := SatClausifyBindingFormula(subFormula)$

        $satClauses[literalBinding] := clauses$
        $literalToBooleanBindings[literalBinding] := booleanBinding$
        $literalBindings.add(literalBinding)$

    end
    $booleanBindingToLiteral[booleanBinding] := literalBindings$

end

La funzione SatClausifyBindingFormula è una funzione che prende in input una formula la clausifica e converte tutte le clausole in SatClausole in modo che ogni satLetterale ha lo stesso indice del funtore del predicato associato. Questo è differente da quello che viene fatto dalla classe Sat2Fo che associa ogni puntatore a letterale ad un nuovo SatLetterale con un nuovo indice arbitrario. A questo punto la formula esterna (quella generata da topBooleanFormula) viene SatClausificata con il metodo standard tramite la classe Sat2Fo.

1.2 Procedura di Decisione

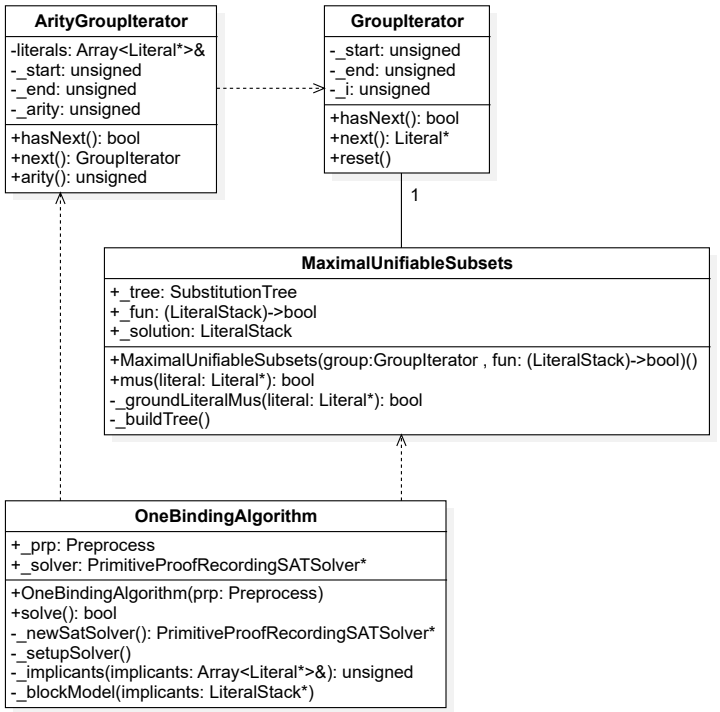

Figura 1.2: Struttura dell'algoritmo di decisione

1.2.1 Implicants Sorting

1.2.2 Maximal Unifiable Subsets

Algorithm 2: Maximal Unifiable Subsets

Firma: mus($literal$)
Input: $literal$ un puntatore ad un letterale
Output: $\top$ o $\bot$
GlobalData: S una mappa da letterali a bool
if $S[literal]$ **then**
  return $\top$;
end
if $literal$ *is ground* **then**
  return $groundLiteralMus(literal)$;
end
$S[literal] = \top$;
$res := mus(literal, \emptyset)$;
$S[literal] = \bot$;
return res;

Algorithm 3: Maximal Unifiable Subsets

Firma: mus($literal$, $FtoFree$)
Input: $literal$ un puntatore ad un letterale, $FtoFree$ un puntatore ad una lista di letterali
Output: $\top$ o $\bot$
GlobalData: **S** una mappa da letterali a interi, **fun** una funzione da lista di letterali a bool, **tree** un SubstitutionTree

```
isMax := ⊤;
uIt = tree.getUnifications(query : literal, retrieveSubstitutions : true);
toFree := ∅;
while uIt.hasNext() do
    (u, σ) := uIt.next();
    if S[u] = 0 then
        S[u] = 1;
        l := literal^σ;
        if l = literal then
            u' := u^σ;
            if u' = u then
                FtoFree := FtoFree ∪ {u};
            end
            else
                toFree := toFree ∪ {u};
            end
        end
        else
            isMax = ⊥;
            if ¬mus(l, toFree) then
                return ⊥;
            end
            S[u] = −1;
            toFree := toFree ∪ {u};
        end
    end
end
if isMax then
    if ¬fun({x | S[x] = 1}) then
        return ⊥;
    end
end
while toFree ≠ ∅ do
    S[toFree.pop()] = 0;
end
return ⊤;
```

Algorithm 4: Maximal Unifiable Subsets Ground

Firma: groundMus($literal$)
Input: $literal$ un puntatore ad un letterale ground
Output: $\top$ o $\bot$
GlobalData: **S** una mappa da letterali a interi, **fun** una funzione da lista di letterali a bool, **tree** un SubstitutionTree

```
if S[literal] ≠ 0 then
    return ⊤;
end
uIt = tree.getUnifications(query : literal, retrieveSubstitutions : true);
solution := ∅;
while uIt.hasNext() do
    (u, σ) := uIt.next();
    if S[u] = 0 then
        if u is ground then
            S[u] = −1;
        end
        solution := solution ∪ {u};
    end
end
return fun(solution);
```

1.2.3 Algoritmo Finale

Algorithm 5: Algoritmo di decisione

Firma: solve(prp)
Input: *prp* il problema pre-processato
Output: $\top$ o $\bot$

```
satSolver := newSatSolver();
satSolver.addClauses(prp.clauses);
while satSolver.solve() = SATISFIABLE do
    res := ⊤;
    implicants := getImplicants(satSolver, prp);
    implicants := sortImplicants(implicants);
    if implicants contains only ground Literals then
        return ⊤;
    end
    agIt := ArityGroupIterator(implicants);
    while res And agIt.hasNext() do
        maximalUnifiableSubsets := SetupMus(group, internalSat);
        foreach lit ∈ group do
            if ¬maximalUnifiableSubsets.mus(lit) then
                res := ⊥;
                blockModel(maximalUnifiableSubsets.getSolution());
                Break;
            end
        end
        if res = ⊤ then
            return ⊤;
        end
    end
end
return ⊥;
```

Algorithm 6: Sat interna

Firma: internalSat(literals)
Input: *literals* una lista di letterali
Output: $\top$ o $\bot$

```
if literals.length = 1 And getSatClauses(literals.top()).length = 1 then
    return ⊤;
end
satSolver := newSatSolver();
foreach l ∈ literals do
    satSolver.addClause(getSatClauses(l));
end
return satSolver.solve() = SATISFIABLE;
```

Algorithm 7: getImplicants

Firma: getImplicants(solver, prp)
Input: *solver* un sat solver, *prp* il problema pre-processato
Output: Una lista letterali

```
implicants := ∅;
foreach l ∈ prp.literals() do
    satL := prp.toSat(l);
    if solver.trueInAssignment(satL) then
        if prp.isBooleanBinding(l) then
            implicants := implicants ∪ prp.getLiteralBindings(l);
        end
        else
            implicants := implicants ∪ {l};
        end
    end
end
return implicants;
```

1.3 Algoritmo di Classificazione

(Input formula rettificata senza true e false)

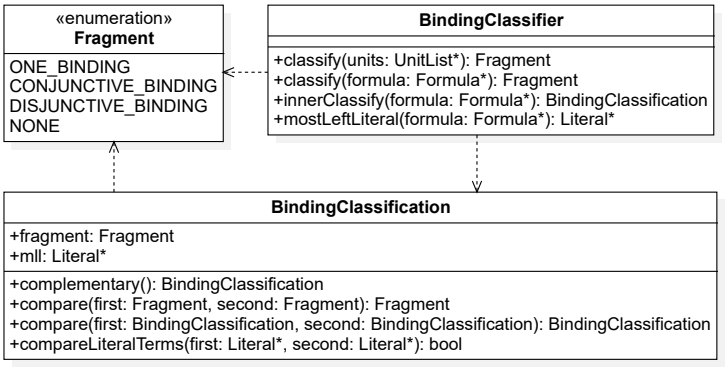

Figura 1.3: Classificatore

Algorithm 8: Classificatore esterno

Firma: classify(φ) **Input:** φ Una formula rettificata
Output: Un elemento dell'enumerazione Fragment
switch φ **do**
    case *Literal* **do**
        return ONE_BINDING;
    end
    case $A[\wedge, \vee]B$ **do**
        return $compare(classify(A), classify(B))$;
    end
    case $\neg A$ **do**
        return $classify(A).complementary()$;
    end
    case $[\forall, \exists]A$ **do**
        $sub := \varphi$;
        connective := connective of φ;
        repeat
            sub := subformula of sub;
            connective := connective of sub;
        until *connective* $\notin \{\forall, \exists\}$;
        $(fragment, \_) := innerClassify(sub)$;
        return $fragment$;
    end
    case $A \Leftrightarrow B$ **do**
        return $compare(classify(A \Rightarrow B), classify(B \Rightarrow A))$;
    end
    case $A \oplus B$ **do**
        return $classify(A \Leftrightarrow B).complementary()$;
    end
    case $A \Rightarrow B$ **do**
        return $compare(classify(\neg A), classify(B))$;
    end
end

Algorithm 9: Classificatore interno

Firma: innerClassify(φ) **Input:** φ Una formula rettificata
Output: Una coppia (Fragment, Literal)
switch φ **do**
  case *Literal* l **do**
    return $(ONE\_BINDING, l)$;
  end
  case $A[\wedge, \vee]B$ **do**
    return $innerCompare(innerClassify(A), innerClassify(B), \text{connective of } \varphi)$;
  end
  case $\neg A$ **do**
    return $innerClassify(A).complementary()$;
  end
  case $A[\Rightarrow, \Leftrightarrow, \oplus]B$ **do**
    return $innerCompare(innerClassify(A), innerClassify(B), \text{connective of } \varphi)$;
  end
  else
    return $(None, null)$;
  end
end

Algorithm 10: Compare esterno

Firma: compare(A, B) **Input:** A, B due elementi dell'enumerazione Fragment
Output: Un elemento dell'enumerazione Fragment
if $A = B$ **then**
  return A;
end
if $One\_Binding \notin \{A, B\}$ **then**
  return $None$;
end
return $max(A, B)$;

Algorithm 11: Compare interno

Firma: innerCompare(A, B, con) **Input:** A, B due coppie (Fragment, Literal), *con* un connettivo
Output: Una coppia (Fragment, Literal)

```
switch A.first, B.first, con do
    case One_Binding, One_Binding, __ do
        if A.second has same terms of B.second then
            return A;
        end
        else if conn = ∧ then
            return (Conjunctive_Binding, null);
        end
        else if conn = ∨ then
            return (Disjunctive_Binding, null);
        end
    end
    case [One_Binding, Conjunctive_Binding | Conjunctive_Binding, One_Binding], ∧ do
        return (Conjunctive_Binding, null);
    end
    case [One_Binding, Disjunctive_Binding | Disjunctive_Binding, One_Binding], ∨ do
        return (Disjunctive_Binding, null);
    end
    case Conjunctive_Binding, Conjunctive_Binding, ∧ do
        return (Conjunctive_Binding, null);
    end
    case Disjunctive_Binding, Disjunctive_Binding, ∨ do
        return (Disjunctive_Binding, null);
    end
end
return (None, null);
```