Capitolo 1

Implementazione di procedure di decisione per frammenti Binding in Vampire

L'algoritmo di decisione, la classificazione, Il preprocessing

1.1 Preprocessing

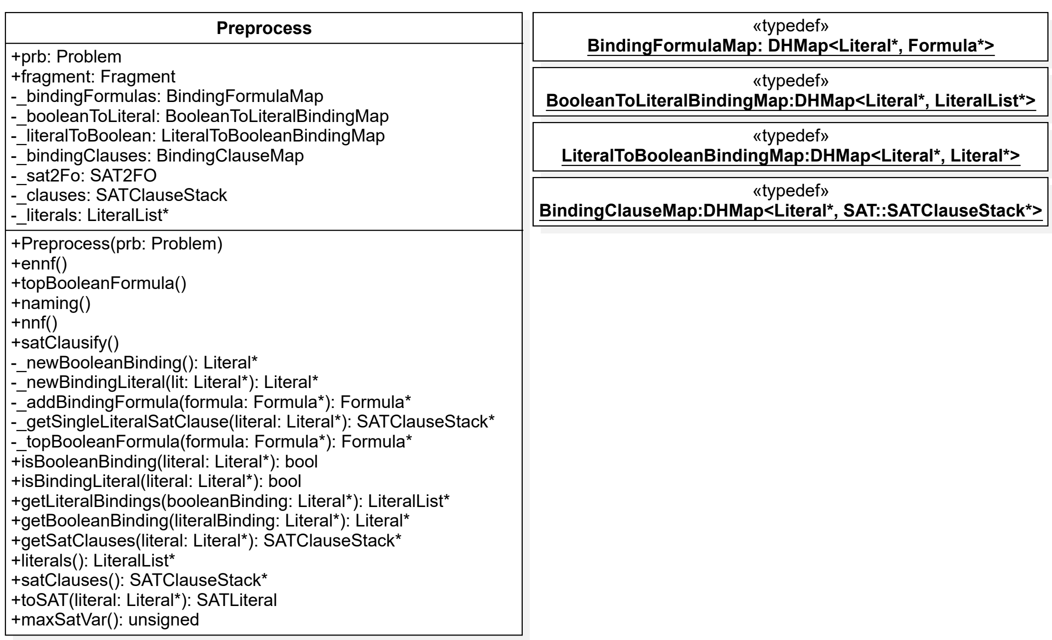

Figura 1.1: Struttura del Preprocessing

1.1.1 Boolean Top Formula

1.1.2 Forall-And

1.1.3 SAT-Clausification

1.2 Procedura di Decisione

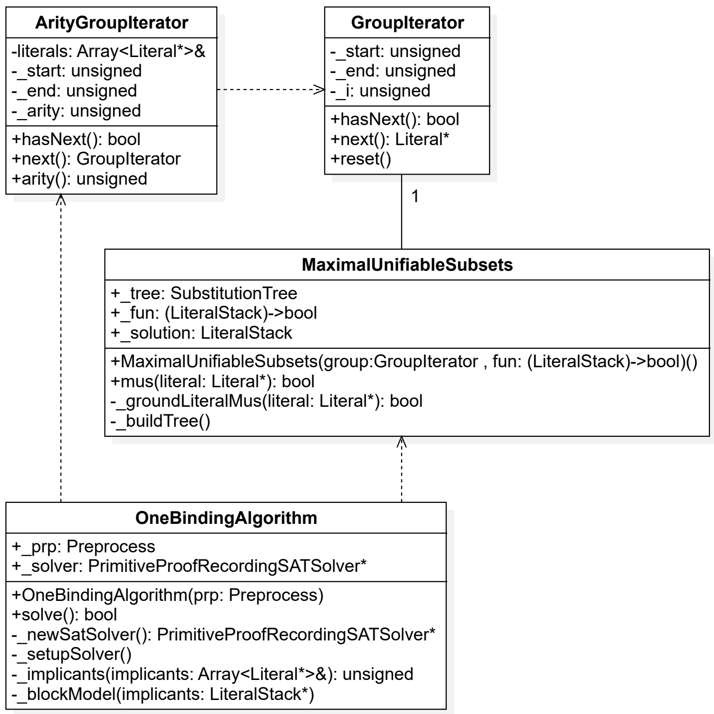

Figura 1.2: Struttura dell'algoritmo di decisione

1.2.1 Implicants Sorting

1.2.2 Maximal Unifiable Subsets

Algorithm 1: 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 ⊤;
end
if literal is ground then
│ return groundLiteralMus(literal);
end
S[literal] = ⊤;
res := mus(literal, ∅);
S[literal] = ⊥;
return res;
```

Algorithm 2: 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 3: 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 4: 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 5: 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 6: 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)

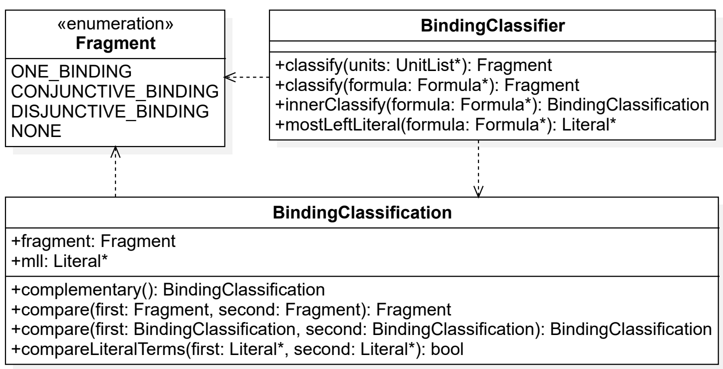

Figura 1.3: Classificatore

Algorithm 7: 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[∧, ∨]B do
        return compare(classify(A), classify(B));
    end
    case ¬A do
        return classify(A).complementary();
    end
    case [∀, ∃]A do
        sub := φ;
        connective := connective of φ;
        repeat
            sub := subformula of sub;
            connective := connective of sub;
        until connective ∉ {∀, ∃};
        (fragment, _) := innerClassify(sub);
        return fragment;
    end
    case A ⇔ B do
        return compare(classify(A ⇒ B), classify(B ⇒ A));
    end
    case A ⊕ B do
        return classify(A ⇔ B).complementary();
    end
    case A ⇒ B do
        return compare(classify(¬A), classify(B));
    end
end
```

Algorithm 8: 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[∧, ∨]B do
        return innerCompare(innerClassify(A), innerClassify(B), connective of φ);
    end
    case ¬A do
        return innerClassify(A).complementary();
    end
    case A[⇒, ⇔, ⊕]B do
        return innerCompare(innerClassify(A), innerClassify(B), connective of φ);
    end
    else
        return (None, null);
    end
end
```

Algorithm 9: 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 ∉ {A, B} then
    return None;
end
return max(A, B);
```

Algorithm 10: 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);
```