

# Capitolo 1

## Implementazione di procedure di decisione per frammenti Binding in Vampire

(descrizione dell'approccio, minime modifiche al kernel, utilizzo di funzionalità esistenti,)

### 1.1 Algoritmo di Classificazione

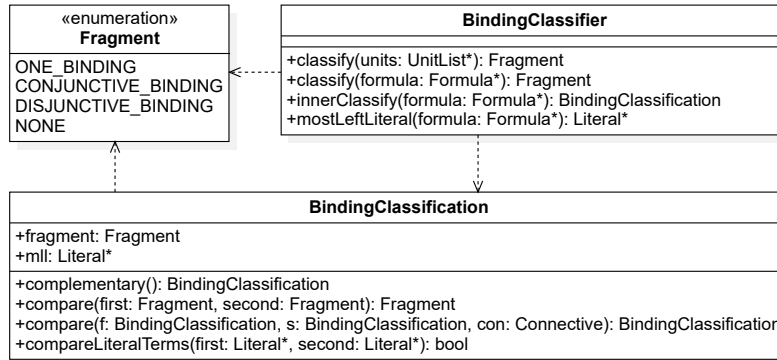


Figura 1.1: Classificatore

La preconditione più importante per la correttezza dell'algoritmo di decisione è che la formula faccia parte del frammento  $\forall 1B$ . Per questo motivo è stato creato un classificatore che prende in input una formula rettificata e restituisce l'elemento del frammento a cui appartiene. Una formula è rettificata se non contiene  $\top$  o  $\perp$ . L'algoritmo non fa altro che verificare la forma sintattica della formula e capisce a quale grammatica della sezione ?? appartiene. Per questo scopo sono state create due funzioni delle Classificatore Esterno (Algoritmo 1) e Classificatore Interno (Algoritmo 3). La prima verifica la parte della formula senza quantificatori mentre la seconda verifica la parte interna ai quantificatori e confronta i termini dei letterali. Entrambi gli algoritmi hanno una struttura di visita dell'albero sintattico in postOrder e hanno una complessità lineare rispetto alla dimensione della formula.

---

**Algorithm 1:** Classificatore esterno

---

**Firma:**  $\text{classify}(\varphi)$  **Input:**  $\varphi$  Una formula rettificata

**Output:** Un elemento dell'enumerazione Fragment

```
switch  $\varphi$  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  $\varphi$ ;
    | 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
end
```

---

---

**Algorithm 2:** Compare esterno

---

**Firma:**  $\text{compare}(A, B)$  **Input:**  $A, B$  due elementi dell'enumerazione Fragment

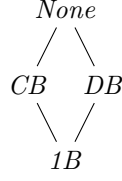
**Output:** Un elemento dell'enumerazione Fragment

```
if  $A = B$  then
  | return  $A$ ;
end
if  $One\_Binding \notin \{A, B\}$  then
  | return None;
end
return  $\max(A, B)$ ;
```

---

Il classificatore esterno si appoggia ad una funzione ausiliaria chiamata *compare* che prende in input due elementi dell'enumerazione Fragment e restituisce il frammento risultante dalla combinazione booleana ( $\wedge, \vee$ ) dei due frammenti. La combinazione di due *1B* è sempre un *1B* mentre la combinazione di un *1B* con un *CB* o *DB* è sempre un *CB* o *DB*. Infine la combinazione di un *CB* con un *DB* fa parte del frammento Boolean Binding che però in questa sezione verrà chiamato *None*. Per la comparazione è

stato creato un ordinamento dei frammenti che segue una struttura a rombo:



Dove  $1B$  è il minimo e  $None$  è il massimo. Il risultato è un reticolo e la funzione `compare` restituisce l'estremo superiore dei due frammenti. La funzione `complementary` restituisce il frammento della negazione di una formula di un determinato frammento. In particolare il complementare di un  $1B$  è  $1B$  mentre il complementare di un  $CB$  è  $DB$  e viceversa.

---

**Algorithm 3:** Classificatore interno

---

**Firma:** `innerClassify( $\varphi$ )` **Input:**  $\varphi$  Una formula rettificata

**Output:** Una coppia (Fragment, Literal)

```

switch  $\varphi$  do
  case Literal  $l$  do
    | return (ONE_BINDING,  $l$ );
  end
  case  $A[\wedge, \vee]B$  do
    | return innerCompare(innerClassify( $A$ ), innerClassify( $B$ ), 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( $B$ ), connective of  $\varphi$ );
  end
  else
    | return (None, null);
  end
end

```

---

La struttura del classificatore interno è molto simile a quella del classificatore esterno, mentre il comparatore interno è leggermente più complesso. Il caso base è quando la formula è un singolo letterale che è sempre un  $1B$ . La visita in `postOrder` restituisce una coppia (Fragment, Literal) che rappresenta il frammento a cui appartiene la formula e un letterale di rappresentanza della formula in questo caso il letterale più a sinistra. Il letterale serve a mantenere una reference alla lista di termini delle formule del frammento  $1B$ .

---

**Algorithm 4:** Compare interno

---

**Firma:**  $\text{innerCompare}(A, B, \text{con})$  **Input:**  $A, B$  due coppie (Fragment, Literal),  $\text{con}$  un connettivo

**Output:** Una coppia (Fragment, Literal)

```
switch  $A.\text{first}, B.\text{first}, \text{con}$  do
  case  $\text{One\_Binding}, \text{One\_Binding}, \_$  do
    if  $A.\text{second}$  has same terms of  $B.\text{second}$  then
      return  $A$ ;
    end
    else if  $\text{con} = \wedge$  then
      return ( $\text{Conjunctive\_Binding}, \text{null}$ );
    end
    else if  $\text{con} = \vee$  then
      return ( $\text{Disjunctive\_Binding}, \text{null}$ );
    end
  end
end
case [ $\text{One\_Binding}, \text{Conjunctive\_Binding} \mid \text{Conjunctive\_Binding}, \text{One\_Binding}$ ],  $\wedge$  do
  return ( $\text{Conjunctive\_Binding}, \text{null}$ );
end
case [ $\text{One\_Binding}, \text{Disjunctive\_Binding} \mid \text{Disjunctive\_Binding}, \text{One\_Binding}$ ],  $\vee$  do
  return ( $\text{Disjunctive\_Binding}, \text{null}$ );
end
case  $\text{Conjunctive\_Binding}, \text{Conjunctive\_Binding}, \wedge$  do
  return ( $\text{Conjunctive\_Binding}, \text{null}$ );
end
case  $\text{Disjunctive\_Binding}, \text{Disjunctive\_Binding}, \vee$  do
  return ( $\text{Disjunctive\_Binding}, \text{null}$ );
end
end
return ( $\text{None}, \text{null}$ );
```

---

La combinazione booleana di due frammenti  $1B$  (all'interno di un quantificatore) può portare a tre diversi risultati. Se i termini dei letterali di rappresentanza sono uguali allora la combinazione è ancora un  $1B$  altrimenti la combinazione è un  $CB$  se il connettivo è  $\wedge$  e un  $DB$  se il connettivo è  $\vee$ . Il termine  $\text{null}$  viene usato come sostituto del letterale di rappresentanza in formule del frammento  $CB$  e  $DB$  in quanto sono una combinazione di più  $1B$  e non hanno un letterale di rappresentanza. Due frammenti  $CB$  rimangono  $CB$  solo se il loro connettivo è  $\wedge$ . La combinazione di un  $1B$  con un  $CB$  è un  $CB$  se il connettivo è  $\wedge$ . Stesso discorso per i  $DB$  e il connettivo  $\vee$ . In tutti gli altri casi la combinazione è  $\text{None}$ . Nell'algoritmo 4 sono stati omessi i casi con connettivi  $\Rightarrow, \Leftrightarrow, \oplus$  in quanto non sono riconducibili a formule composte da  $\wedge$  e  $\vee$  come è stato fatto ad esempio nell'algoritmo 1. Con la funzione  $\text{complementary}$  applicata ad una coppia:  $(\text{Fragment}, \text{Literal}).\text{complementary}()$  si intende la coppia  $(\text{Fragment.complementary}(), \text{Literal})$ .

## 1.2 Preprocessing

Preprocess	«typedef» BindingFormulaMap: DHMap<Literal*, Formula*>
+prb: Problem +fragment: Fragment - _bindingFormulas: BindingFormulaMap - _booleanToLiteral: BooleanToLiteralBindingMap - _literalToBoolean: LiteralToBooleanBindingMap - _bindingClauses: BindingClauseMap - _sat2Fo: SAT2FO - _clauses: SATClauseStack - _literals: LiteralList*	«typedef» BooleanToLiteralBindingMap: DHMap<Literal*, LiteralList*>
	«typedef» LiteralToBooleanBindingMap: DHMap<Literal*, Literal*>
	«typedef» BindingClauseMap: DHMap<Literal*, SAT::SATClauseStack*>
+Preprocess(prb: Problem) +ennf() +topBooleanFormula() +naming() +nnf() +satClausify() - _newBooleanBinding(): Literal* - _newBindingLiteral(lit: Literal*): Literal* - _addBindingFormula(formula: Formula*): Formula* - _getSingleLiteralSatClause(literal: Literal*): SATClauseStack* - _topBooleanFormula(formula: Formula*): Formula* +isBooleanBinding(literal: Literal*): bool +isBindingLiteral(literal: Literal*): bool +getLiteralBindings(booleanBinding: Literal*): LiteralList* +getBooleanBinding(literalBinding: Literal*): Literal* +getSatClauses(literal: Literal*): SATClauseStack* +literals(): LiteralList* +satClauses(): SATClauseStack* +toSAT(literal: Literal*): SATLiteral +maxSatVar(): unsigned	

Figura 1.2: Struttura del Preprocessing

In questa sezione verrà descritto l'algoritmo di preprocessing utilizzato per trasformare una formula in input del frammento *1B* o *CB* in una struttura trattabile dall'algoritmo di decisione. Per utilizzare il SatSolver di Vampire per la ricerca degli implicanti è necessario clausificare la formula. Inoltre per evitare un'esplosione esponenziale di formule causate dalle forme NNF e CNF è necessario utilizzare tecniche di naming. Qui sorgono i primi problemi visto che né la clausificazione né il naming sono processi conservativi rispetto ai frammenti. Ad esempio la semplice formula del frammento *1B*  $\forall x_1(p_1(x_1)) \vee p_2$  clausificata diventa  $\{\{p_1(x_1), p_2\}\}$  che fa parte del frammento *DB*. L'approccio utilizzato è stato quello di creare una nuova formula ground che rappresenta la struttura booleana esterna della formula originale, applicare le funzioni standard di preprocessing e mantenere una serie di strutture per risalire ai componenti originali. Per questo scopo viene introdotto un nuovo insieme di simboli di predicato  $\Sigma_b = \{b_1, b_2, \dots\}$ . I predicati di  $\Sigma_b$  con arità 0 saranno chiamati *booleanBinding* e saranno associati ad una formula del frammento *1B* o *CB*. I predicati di  $\Sigma_b$  con arità  $n > 0$  saranno chiamati *literalBinding* e fungeranno da rappresentanti dei  $\tau$ -Binding delle formule *1B*. Il preprocessing seguirà pressoché questa struttura:

1. Rettificazione
2. Trasformazione in ENNF
3. Creazione della formula booleana esterna (FBE) e associazione dei booleanBinding
4. Naming della FBE
5. Trasformazione in NNF della FBE
6. Creazione dei literalBinding e Sat-Clausificazione delle formule mappate dai booleanBinding
7. Creazione delle Sat-Clausole della FBE

La rettificazione e la trasformazione in ENNF sono processi conservativi rispetto ai frammenti e quindi verranno applicate direttamente le funzioni standard di Vampire. La creazione della FBE e l'associazione dei booleanBinding avviene tramite l'algoritmo 5.

---

**Algorithm 5:** Top Boolean Formula

---

**Firma:** topBooleanFormula( $\varphi$ )

**Input:**  $\varphi$  una formula rettificata

**Output:** Una formula ground

**GlobalData:** bindingFormulas una mappa da booleanBinding a formula **switch**  $\varphi$  **do**

```

    case Literal  $l$  do
        | return new AtomicFormula( $l$ );
    end
    case  $A[\wedge, \vee]B$  do
        | return new JunctionFormula(topBooleanFormula( $A$ ), connective of  $\varphi$ ,
        |   topBooleanFormula( $B$ ));
    end
    case  $\neg A$  do
        | return new NegatedFormula(topBooleanFormula( $A$ ));
    end
    case  $[\forall, \exists]A$  do
        |  $b = \text{new BooleanBinding}()$ ;
        |  $\text{bindingFormulas}[b] := \varphi$ ;
        | return new AtomicFormula( $b$ );
    end
    case  $A[\Leftrightarrow, \Rightarrow, \oplus]B$  do
        | return new BinaryFormula( $A$ , connective of  $\varphi$ ,  $B$ );
    end
end

```

---

L'algoritmo prende in input una formula rettificata e restituisce una formula ground sostituendo le sottoformule quantificate con un nuovo booleanBinding aggiungendo la sottoformula originale alla mappa bindingFormulas. Da adesso in poi qualunque modifica fatta alla FBE preserverà l'appartenenza al frammento originale. Gli step successivi sono quindi applicare le funzioni standard di Vampire per il naming e la trasformazione in NNF. La trasformazione in NNF potrebbe portare alla negazione di qualche booleanBinding e va quindi aggiunta alla mappa bindingFormulas la formula negata associata.

**foreach**  $l \in \text{literals}(\varphi)$  **do**

```

    if  $\neg l.\text{polarity}()$  then
        | continue
    end

     $\text{positiveFormula} := \text{bindingFormulas}[\text{positiveLiteral}(l)]$ 
     $\text{bindingFormulas}[l] := \text{new NegatedFormula}(\text{positiveFormula})$ 
end

```

A questo punto inizia il processo di SatClausificazione delle formule interne (quelle associate ai booleanBinding). Ogni letterale ground che non è un booleanBinding viene trasformato in una SatClausola di lunghezza 1 composta dal solo satLetterale associato al letterale.

```

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

```

```

end

```

Per essere clausificate le formule della mappa *bindingFormulas* vanno trasformate in NNF, Skolemizzate. Anche in questo caso vengono utilizzate le funzioni standard di Vampire. 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 *LiteralBinding* in rappresentanza 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  $\text{bindingFormulas} \neq \emptyset$  do
  (booleanBinding, formula) :=  $\text{bindingFormulas.pop}()$ 
  formula :=  $\text{nnf}(\text{formula})$ 
  formula :=  $\text{skolemize}(\text{formula})$ 
  todo :=  $\emptyset$ 

  if formula is ConjunctiveBinding then
    | formula :=  $\text{distributeForAll}(\text{formula})$ 
    | "Add each subformula to the todo list"
  end

  else
    | todo.add(formula)
  end

  literalBindings :=  $\emptyset$  while todo  $\neq \emptyset$  do
    subformula :=  $\text{todo.pop}()$ 
    literalBinding :=  $\text{newLiteralBinding}(\text{subformula.mostLeftLiteral}())$ 
    clauses :=  $\text{SatClausifyBindingFormula}(\text{subFormula})$ 

    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. L'ultimo step è la SatClausificazione della FBE che avviene tramite le funzioni standard di Vampire della classe Sat2Fo. È importante ricordare che i satLetterali delle formule interne sono diversi dai satLetterali della FBE nonostante possano avere lo stesso indice.

Si prenda ad esempio la formula del frammento *CB* :

$$(\forall x_1, x_2((p_1(x_1) \vee p_2(x_1)) \wedge p_2(f_1(x_2))) \wedge \forall x_1(p_3(x_1) \Rightarrow p_1(x_1))) \vee (\forall x_1(p_2(x_1)) \Rightarrow p_4)$$

Il primo passo di preprocessing prevede la rettificazione e la trasformazione in ENNF. La formula è già rettificata mentre la trasformazione in ENNF porta all'eliminazione del  $\Rightarrow$ :

$$(\forall x_1, x_2((p_1(x_1) \vee p_2(x_1)) \wedge p_2(f_1(x_2))) \wedge \forall x_1(\neg p_3(x_1) \vee p_1(x_1))) \vee (\forall x_1(p_2(x_1)) \Leftrightarrow p_4)$$

La creazione della FBE porta alla generazione di un booleanBinding per ogni sottoformula quantificata:

$$(b_1 \wedge b_2) \vee (b_3 \Leftrightarrow p_4)$$

La mappa bindingFormulas contiene le seguenti coppie:

$$b_1 \rightarrow \forall x_1, x_2((p_1(x_1) \vee p_2(x_1)) \wedge p_2(f_1(x_2)))$$

$$b_2 \rightarrow \forall x_1(\neg p_3(x_1) \vee p_1(x_1))$$

$$b_3 \rightarrow \forall x_1(p_2(x_1))$$

La formula ottenuta è troppo piccola per poter applicare il namig quindi si procede direttamente con la trasformazione in NNF:

$$(b_1 \wedge b_2) \vee ((\neg b_3 \vee p_4) \wedge (b_3 \vee \neg p_4))$$

Durante il processo di NNF il booleanBinding  $b_3$  è stato negato e quindi va aggiunto alla mappa bindingFormulas:

$$\neg b_3 \rightarrow \exists x_1(\neg p_2(x_1))$$

A questo punto vengono trasformate in NNF e Skolemizzate le formule associate ai booleanBinding, vengono poi creati i literalBindings e le SatClausole delle formule interne. Il booleanBinding  $b_1$  è associato ad una formula *CB* quindi viene distribuito il quantificatore universale sull'and e creati due literalBindings. La skolemizzazione della formula associata a  $\neg b_3$  porta alla formula::

$$\neg b_3 \rightarrow \neg p_2(sk_1)$$

Vengono create così le mappe booleanBindingToLiteral e la sua inversa literalToBooleanBindings:

<b>booleanBindingToLiteral</b>	<b>literalToBooleanBindings</b>
$b_1 \rightarrow \{b_4(x_1), b_5(f_1(x_1))\}$	$b_4(x_1) \rightarrow b_1$
$b_2 \rightarrow \{b_6(x_1)\}$	$b_5(f_1(x_1)) \rightarrow b_1$
$b_3 \rightarrow \{b_7(x_1)\}$	$b_6(x_1) \rightarrow b_2$
$\neg b_3 \rightarrow \{b_8(sk_1)\}$	$b_7(x_1) \rightarrow b_3$
	$b_8(sk_1) \rightarrow \neg b_3$

Le formule associate ai literalBindings vengono clausificate:



- $\forall x_1, x_2((p_1(x_1) \vee p_2(x_1))) \rightarrow \{\{(p_1(x_1), p_2(x_1))\}\}$
- $\forall x_1, x_2(p_2(f_1(x_2))) \rightarrow \{\{p_2(f_1(x_2))\}\}$
- $\forall x_1(\neg p_3(x_1) \vee p_1(x_1)) \rightarrow \{\{\neg p_3(x_1), p_1(x_1)\}\}$
- $\forall x_1(p_2(x_1)) \rightarrow \{\{p_2(x_1)\}\}$
- $\neg p_2(sk_1) \rightarrow \{\{\neg p_2(sk_1)\}\}$

E successivamente SatClausificate e associate ai literalBindings:

- $b_4(x_1) \rightarrow \{\{s_1, s_2\}\}$
- $b_5(f_1(x_1)) \rightarrow \{\{s_2\}\}$
- $b_6(x_1) \rightarrow \{\{\neg s_3, s_1\}\}$
- $b_7(x_1) \rightarrow \{\{s_2\}\}$
- $b_8(sk_1) \rightarrow \{\{\neg s_2\}\}$

Gli ultimi due step sono la clausificazione della FBE:

$$\{\{b_1, \neg b_3, p_4\}, \{b_2, \neg b_3, p_4\}, \{b_1, b_3, \neg p_4\}, \{b_2, b_3, \neg p_4\}\}$$

E la SatClausificazione tramite sat2Fo:

$$\{\{s_1, \neg s_2, s_3\}, \{s_4, \neg s_2, s_3\}, \{s_1, s_2, \neg s_3\}, \{s_4, s_2, \neg s_3\}\}$$

Che crea internamente una bi-mappa che associa ogni satletterale ad un letterale:

- |                                  |                             |
|----------------------------------|-----------------------------|
| • $s_1 \leftrightarrow b_1$      | • $s_3 \leftrightarrow p_4$ |
| • $s_2 \leftrightarrow \neg b_3$ | • $s_4 \leftrightarrow b_2$ |

## 1.3 Procedura di Decisione

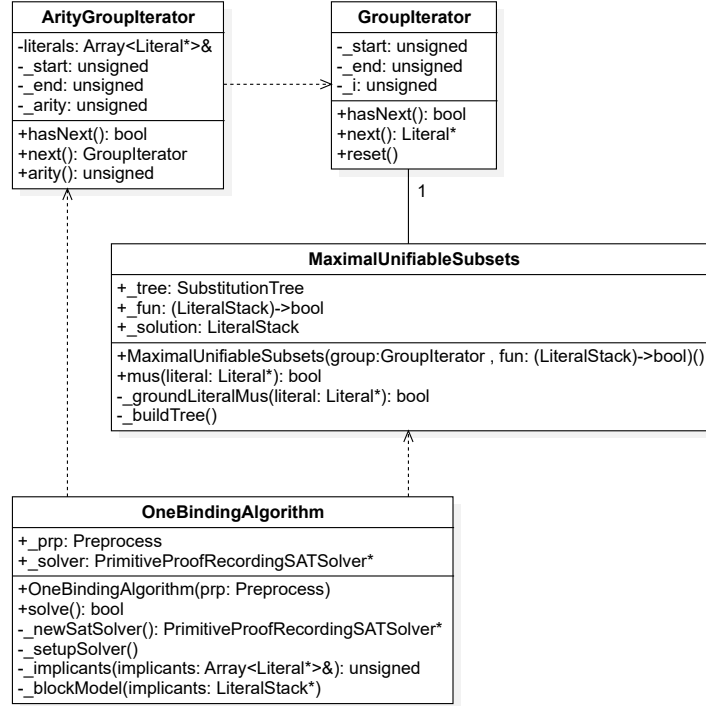


Figura 1.3: Struttura dell'algoritmo di decisione

### 1.3.1 Implicants Sorting

### 1.3.2 Maximal Unifiable Subsets

---

#### Algorithm 6: Maximal Unifiable Subsets

---

**Firma:**  $\text{mus}(\text{literal})$

**Input:**  $\text{literal}$  un puntatore ad un letterale

**Output:**  $\top$  o  $\perp$

**GlobalData:**  $S$  una mappa da letterali a bool

**if**  $S[\text{literal}]$  **then**

**return**  $\top$ ;

**end**

**if**  $\text{literal}$  is ground **then**

**return**  $\text{groundLiteralMus}(\text{literal})$ ;

**end**

$S[\text{literal}] = \top$ ;

$\text{res} := \text{mus}(\text{literal}, \emptyset)$ ;

$S[\text{literal}] = \perp$ ;

**return**  $\text{res}$ ;

---

---

**Algorithm 7:** Maximal Unifiable Subsets

---

**Firma:**  $\text{mus}(\text{literal}, \text{FtoFree})$

**Input:**  $\text{literal}$  un puntatore ad un letterale,  $\text{FtoFree}$  un puntatore ad una lista di letterali

**Output:**  $\top$  o  $\perp$

**GlobalData:**  $\mathbf{S}$  una mappa da letterali a interi, **fun** una funzione da lista di letterali a bool, **tree** un SubstitutionTree

$\text{isMax} := \top$ ;

$\text{uIt} = \text{tree.getUnifications}(\text{query} : \text{literal}, \text{retrieveSubstitutions} : \text{true})$ ;

$\text{toFree} := \emptyset$ ;

**while**  $\text{uIt.hasNext}()$  **do**

$(u, \sigma) := \text{uIt.next}()$ ;

**if**  $S[u] = 0$  **then**

$S[u] = 1$ ;

$l := \text{literal}^\sigma$ ;

**if**  $l = \text{literal}$  **then**

$u' := u^\sigma$ ;

**if**  $u' = u$  **then**

$\text{FtoFree} := \text{FtoFree} \cup \{u\}$ ;

**end**

**else**

$\text{toFree} := \text{toFree} \cup \{u\}$ ;

**end**

**end**

**else**

$\text{isMax} = \perp$ ;

**if**  $\neg \text{mus}(l, \text{toFree})$  **then**

**return**  $\perp$ ;

**end**

$S[u] = -1$ ;

$\text{toFree} := \text{toFree} \cup \{u\}$ ;

**end**

**end**

**end**

**if**  $\text{isMax}$  **then**

**if**  $\neg \text{fun}(\{x \mid S[x] = 1\})$  **then**

**return**  $\perp$ ;

**end**

**end**

**while**  $\text{toFree} \neq \emptyset$  **do**

$S[\text{toFree.pop}()] = 0$ ;

**end**

**return**  $\top$ ;

---

---

**Algorithm 8:** Maximal Unifiable Subsets Ground

---

**Firma:**  $\text{groundMus}(\text{literal})$

**Input:** *literal* un puntatore ad un letterale ground

**Output:**  $\top$  o  $\perp$

**GlobalData:** **S** una mappa da letterali a interi, **fun** una funzione da lista di letterali a bool,  
tree un SubstitutionTree

**if**  $S[\text{literal}] \neq 0$  **then**

**return**  $\top$ ;

**end**

$uIt = \text{tree.getUnifications}(\text{query} : \text{literal}, \text{retrieveSubstitutions} : \text{true});$

$\text{solution} := \emptyset;$

**while**  $uIt.hasNext()$  **do**

$(u, \sigma) := uIt.next();$

**if**  $S[u] = 0$  **then**

**if**  $u$  is ground **then**

$S[u] = -1;$

**end**

$\text{solution} := \text{solution} \cup \{u\};$

**end**

**end**

**return**  $\text{fun}(\text{solution});$ 

---

### 1.3.3 Algoritmo Finale

---

**Algorithm 9:** Algoritmo di decisione

---

**Firma:** solve(*prp*)

**Input:** *prp* il problema pre-processato

**Output:**  $\top$  o  $\perp$

*satSolver* := newSatSolver();

*satSolver.addClauses*(*prp.clauses*);

**while** *satSolver.solve*() = SATISFIABLE **do**

*res* :=  $\top$ ;

*implicants* := *getImplicants*(*satSolver*, *prp*);

*implicants* := *sortImplicants*(*implicants*);

**if** *implicants* contains only ground Literals **then**

**return**  $\top$ ;

**end**

*agIt* := *ArityGroupIterator*(*implicants*);

**while** *res* And *agIt.hasNext*() **do**

*maximalUnifiableSubsets* := *SetupMus*(*group*, *internalSat*);

**foreach** *lit*  $\in$  *group* **do**

**if**  $\neg$ *maximalUnifiableSubsets.mus*(*lit*) **then**

*res* :=  $\perp$ ;

*blockModel*(*maximalUnifiableSubsets.getSolution*());

**Break**;

**end**

**end**

**if** *res* =  $\top$  **then**

**return**  $\top$ ;

**end**

**end**

**end**

**return**  $\perp$ ;

---

---

**Algorithm 10:** Sat interna

---

**Firma:** internalSat(*literals*)

**Input:** *literals* una lista di letterali

**Output:**  $\top$  o  $\perp$

**if** *literals.length* = 1 And *getSatClauses*(*literals.top*()) *length* = 1 **then**

**return**  $\top$ ;

**end**

*satSolver* := newSatSolver();

**foreach** *l*  $\in$  *literals* **do**

*satSolver.addClause*(*getSatClauses*(*l*));

**end**

**return** *satSolver.solve*() = SATISFIABLE;

---

---

**Algorithm 11:** getImplicants

---

**Firma:** getImplicants(solver, prp)

**Input:** *solver* un sat solver, *prp* il problema pre-processato

**Output:** Una lista letterali

*implicants* :=  $\emptyset$ ;

**foreach**  $l \in prp.literals()$  **do**

*satL* := *prp.toSat*(*l*);

**if** *solver.trueInAssignment*(*satL*) **then**

**if** *prp.isBooleanBinding*(*l*) **then**

*implicants* := *implicants*  $\cup$  *prp.getLiteralBindings*(*l*);

**end**

**else**

*implicants* := *implicants*  $\cup$  {*l*};

**end**

**end**

**end**

**return** *implicants*;

---