```
* Generated by SAE Prolog - www.opal-project.de
* DO NOT CHANGE MANUALLY - THE CLASS WILL COMPLETELY BE REGENERATED
package predicates;
import saere.*;
import saere.predicate.ArithNotEqual2;
import saere.predicate.Is2;
import saere.predicate.Unify2;
import saere.term.*;
import static saere.term.Terms.*;
import static saere.IntValue.*;
import static saere.StringAtom.*;
public final class not_attack3 implements Solutions {
    private int clauseToExecute = 1;
    // private Solutions clauseSolutions;
    private GoalStack goalStack = GoalStack.emptyStack();
    private int goalToExecute = 1;
    private boolean cutEvaluation = false;
    final private Term arg0;
    final private Term arg1;
    final private Term arg2;
    public not_attack3(final Term arg0, final Term arg1, final Term arg2) {
      this.arg0 = arg0;
      this.arg1 = arg1;
      this.arg2 = arg2;
    }
    public boolean next() {
      switch (this.clauseToExecute) {
      case 1: {
          if (this.clause1()) {
            return true;
          } else {
            if (this.cutEvaluation) {
                return false;
            } else {
                this.clauseToExecute = 2;
                this.goalToExecute = 1;
            }
      }
      case 2: {
          return this.clause2();
      }
      default:
          // // should never be reached
```

```
throw new Error("internal compiler error");
   }
  }
 public void abort() {
   // this.clauseSolutions.abort();
   // this.clauseSolutions = null;
   goalStack.abortPendingGoals();
  public boolean choiceCommitted() {
   return false;
  }
//How to translate the cut (let it be goal X)?
//case X: this.cutEvaluation = true; this.goalToExec
                                                       = X+2; continue;
//case X+1: this.goalStack = goalStack.abortPendingGoals(); return false;
 private boolean clause1() {
   do {
        switch (this.goalToExecute) {
       case 0: // the clause failed (there are no more solutions)
          return false;
       case 1: {
         Term $H1 = this.arg0;
          // // arg1 is not used
         // // arg2 is not used
         this.goalStack = goalStack.put(new Unify2($H1, EMPTY_LIST));
        }
       case 2: {
          // boolean succeeded = this.clauseSolutions.next();
          final Solutions solutions = this.goalStack.peek();
          final boolean succeeded = solutions.next();
          if (!succeeded) {
              this.goalStack = this.goalStack.drop();
              goalToExecute = 0; // -= 2
              continue;
          }
        }
       case 3: {
         this.cutEvaluation = true;
          this.goalToExecute = 5;
          continue;
        }
       case 4: {
          this.goalStack = this.goalStack.abortPendingGoals();
          return false;
       case 5: // the clause succeeded
          goalToExecute = 4; // -1 (redo the previous clause...)
          return true;
        default:
          // should never be reached
          throw new Error("internal compiler error");
```

```
} while (true);
    private Variable Y = variable();
    private Variable Ys = variable();
    private Term N1 = variable();
    private State $H1State;
    // not_attack([Y|<u>Ys</u>],X,N) :-
    // X = \ Y+N
    // X = Y - N
    // N1 is N+1,
    // not_attack(Ys,X,N1).
    private boolean clause2() {
      do {
          switch (this.goalToExecute) {
          case 0:
            return false;
//
          case 1: {
            Term $H1 = this.arg0;
            this.goalStack = goalStack.put(new Unify2(\$H1,compoundTerm(LIST, Y, \underline{Y}s)));
//
//
          }
//
          case 2: {
//
            final Solutions solutions = this.goalStack.peek();
            final boolean succeeded = solutions.next();
//
//
            if (!succeeded) {
//
                this.goalStack = this.goalStack.drop();
                this.goalToExecute = 0; // -= 2 // previous goal
//
//
                continue;
            }
//
//
          case 1: {
            Term $H1 = this.arg0;
            $H1State = $H1.manifestState();
            if(Term.unify($H1, compoundTerm(LIST, Y, Ys))) {
                goalToExecute = 3;
                continue;
            }
          }
          case 2: {
            if ($H1State != null)
                $H1State.reset();
            goalToExecute = 0;
            continue;
          }
          case 3:{
            Term X = this.arg1;
            Term N = this.arg2;
            this.goalStack = goalStack.put(new ArithNotEqual2(X, compoundTerm(PLUS, Y, N)));
          }
```

```
final Solutions solutions = this.goalStack.peek();
            final boolean succeeded = solutions.next();
            if (!succeeded) {
                this.goalStack = this.goalStack.drop();
                this.goalToExecute = 2; // -= 2 // previous goal
                continue;
            }
          }
          case 5:{
            Term X = this.arg1;
            Term N = this.arg2;
            this.goalStack = goalStack.put(new ArithNotEqual2(X, compoundTerm(MINUS, Y, N)));
            this.goalToExecute = 6;
          }
          case 6:{
            final Solutions solutions = this.goalStack.peek();
            final boolean succeeded = solutions.next();
            if (!succeeded) {
                this.goalStack = this.goalStack.drop();
                this goalToExecute = 4; // -= 2 // previous goal
                continue;
            }
          }
          case 7:{
//
//
            Term N = this.arg2;
            this.goalStack = goalStack.put(new Is2(N1, compoundTerm(PLUS, N, IntValue_1)));
//
          }
//
//
          case 8:{
            final Solutions solutions = this.goalStack.peek();
//
            final boolean succeeded = solutions.next();
//
//
            if (!succeeded) {
                this.goalStack = this.goalStack.drop();
//
//
                this.goalToExecute = 6; // -= 2 // previous goal
                continue;
            }
//
//
          }
          case 7:{
            Term N = this.arg2;
            N1 = IntValue.get(N.intEval() + 11);
            this.goalToExecute = 9;
            continue;
          }
          case 8:{
            this.goalToExecute = 6;
            continue;
          }
          case 9:{
            Term X = this.arg1;
            this.goalStack = goalStack.put(new not_attack3(Ys, X, N1));
          case 10:{
```

case 4:{

```
final Solutions solutions = this.goalStack.peek();
    final boolean succeeded = solutions.next();
    if (!succeeded) {
        this.goalStack = this.goalStack.drop();
        this.goalToExecute = 8; // -= 2 // previous goal
        continue;
    }
}
case 11: {
    goalToExecute = 10;
    return true;
}
default:
    // should never be reached
    throw new Error("internal compiler error");
}
while (true);
}
```