```
 * Generated by SAE Prolog - www.opal-project.de
package predicates;

import saere.*;

public final class rightTo3 implements Goal {
    // variables to control/manage the execution this predicate
    private int clauseToExecute/* = 0; */;
    private int goalToExecute/* = 0; */;
    private GoalStack goalStack = GoalStack.EMPTY_GOAL_STACK;
    // variables related to the predicate's state
    private Term arg0;
    private Term arg1;
    private Term arg2;
    final private State initialArg0state;
    final private State initialArg1state;
    final private State initialArg2state;
    // variables to store clause local information
    private Term clv0;

    public rightTo3(final Term arg0, final Term arg1, final Term arg2) {
        this.arg0 = arg0;
        this.arg1 = arg1;
        this.arg2 = arg2;
        this.initialArg0state = (arg0).manifestState();
        this.initialArg1state = (arg1).manifestState();
        this.initialArg2state = (arg2).manifestState();
    }

    public void abort() {
        if (this.initialArg0state != null)
            this.initialArg0state.reincarnate();
        if (this.initialArg1state != null)
            this.initialArg1state.reincarnate();
        if (this.initialArg2state != null)
            this.initialArg2state.reincarnate();
        this.goalStack = goalStack.abortPendingGoals();
    }

    public boolean choiceCommitted() {
        return false;
    }

    public boolean next() {
        eval_clauses: do {
            switch (this.clauseToExecute) {
            case 0: {
                if (this.clause0()) {
                    return true;
                }
                // this clause contains no "cut"
                // prepare the execution of the next clause
                this.goalToExecute = 0;
                this.clauseToExecute = 1;
            }
            case 1: {
                // tail recursive clause with last call optimization
                if (this.clause1()) {
                    this.clv0 = null;
                    this.goalToExecute = 0;
                    this.clauseToExecute = 0;
                    continue eval_clauses;
                }
                this.abort();
                return false;
            }
            }
        } while (true);
    }

    private boolean clause0() {
        switch (this.goalToExecute) {

        case 0: {
            this.goalStack = goalStack.put(UndoGoal.create
(arg0,arg1,arg2));
            if (arg2.unify(list(arg0, list(arg1, variable())))) {
                this.goalToExecute = 1;
                return true;
            }
        }
        case 1: {
            this.goalStack = goalStack.abortTopLevelGoal();
            return false;
        }

//      case 0: {
//          this.goalStack = goalStack.put(new Unify2(arg2, list(arg0,
list(arg1, variable())))));
//      }
//      case 1: {
//          boolean succeeded = this.goalStack.peek().next();
//          if (!succeeded) {
//              this.goalStack = goalStack.drop();
//              return false;
//          }
//          this.goalToExecute = 1;
//          return true;
//      }
        default:
            // should never be reached
            throw new Error("internal compiler error");
        }
    }

    private boolean clause1() {
        eval_goals: do {
            switch (this.goalToExecute) {
            case 0: {
                // [arg(2),clv(0)]
                this.clv0 = variable();
                this.goalStack = goalStack.put(new Unify2(arg2, list
(variable(), clv0)));
            }
            case 1: {
                boolean succeeded = this.goalStack.peek().next();
                if (!succeeded) {
                    this.goalStack = goalStack.drop();
                    return false;
                }
                // fall through ... 2
            }
            case 2: {
                // tail call with last call optimization
                this.arg0 = arg0;
                this.arg1 = arg1;
                this.arg2 = clv0;
                this.goalStack = GoalStack.EMPTY_GOAL_STACK;
                return true;
            }
            }
        } while (true);
    }
}
```