Цель работы

Целью работы является изучение правил использования отсечения в двух случаях: для подтверждения правильности выбранного решения и для прекращения процесса порождения и проверки возможных решений.

Задание 1

Написать для игры «Крестики-нолики» процедуру «Следующий ход», которая для заданного положения на доске находила бы наилучший ход, обеспечивающий либо предотвращение проигрыша, либо выигрыш, либо наилучший прогнозируемый результат.

Результат выполнения задания 1

Был разработан предикат makeStep, который имеет три аргумента: первый — поле игры, второй — строка клетки поля, на которую нужно сделать следующий ход, третий — столбец клетки поля, на которую нужно сделать следующий ход. Предикат для заданной конфигурации доски находит лучший ход, обеспечивающий либо предотвращение проигрыша, либо выигрыш, либо наилучший прогнозируемый результат. Предполагается, что игрок всегда ходит крестиком и компьютер всегда ходит ноликом, но первым может ходить как игрок, так и компьютер, в зависимости от выбора игрока. Код предиката makeStep и вспомогательных предикатов представлен в Листинге 1 в Приложении А.

Предикат состоит из нескольких фактов и правил. Факты описывают специфические стратегии, приводящих к выигрышу или ничье. Например, правило $makeStep([[x,a,a],[a,a,a],[a,a,a]],2,2)$. гласит, что если игрок поставил «x» в верхнем левом углу поля, то компьютер должен поставить «o» в центре поля, в позиции $(2,2)$, т.е. в клетке во втором столбце второй строки. В Листинге 1 перечислены аналогичные факты, которые гласят, что если игрок сделал ход в угол или край поля, компьютер должен сделать ход в центр поля.

Также имееются следующие факты: поставить «o» в верхний левый угол, если игрок ходит первым и поставил «x» в центр; поставить «o» в верхний левый угол, если компьютер ходит первым; поставить «o» в нижний правый угол, если компьютер ходит первым, компьютер уже поставил «o» в верхний левый угол, пользователь поставил «x» в центр уже после этого; если компьютер ходит первым, компьютер уже поставил «o» в верхний левый угол, пользователь поставил «x» где угодно, кроме центра, то компьютер должен поставить «o» в любой угол таким образом, чтобы между первым «o» и вторым «o» оставалось свободное место; если поле содержит диагональ вида «xox», поставить «o» в позицию $(1,2)$; поставить «o» в верхний правый угол, если есть строка или столбец вида «xox» посередине поля.

Правила описывают более общую беспроигрышную стратегию. Правила расположены таким образом, чтобы компьютер попытался сначала выиграть игру, завершив последовательность из ноликов (предикат tryToWin), затем попытался не проиграть игру, не дав игроку закончить последовательность из крестиков (предикат tryNotLoose), затем попытался продолжить начатую последовательность (например, когда поле содержит строку «oaa» и нет других кандидатов, кроме этой строки, он должен поставить «o» в центр или конец этой строки, предикат fillNext), и только потом, если все предыдущие попытки не увенчались успехом, попытаться поставить «o» в любую свободную клетку (предикат moveToEmpty). Предикат tryToWin устроен следующим образом: это последовательность правил, где мы сначала пытаемся найти «выигрышную» строку (т.е. строку, где два нолика и одна свободная клетка, предикат findWinningRow), потом пытаемся найти «выигрышный» столбец (предикат findWinningColumn), потом пытаемся найти «выигрышную» диагональ (предикат findWinningDiag). Аналогичным образом устроены предикаты tryNotLoose и fillNext.

Задание 2

Написать программу, реализующую выигрышную стратегию для игры «Крестики-нолики» на доске $3 \cdot 3$. Игровое поле и весь процесс игры должен отображаться на экране.

Результат выполнения задания 2

Вдобавок к предикату makeStep и его вспомогательным предикатам, разработанным в ходе выполнения задания 1, были разработаны предикаты, отвечающие за сам процесс игры (см. Листинг 2). Предикат

initField инициализирует поле игры. Предикат computerFirst начинает игру, предоставляя право сделать первый шаг компьютеру. Предикат computerFirstLoop является главным циклом игры в случае, если компьютер ходит первым. userFirst начинает игру, в которой игрок ходит первым. userFirstLoop является главным циклом игры в случае, когда игрок ходит первым. computerMove делает шаг, который компьютер выбрал с помощью makeStep и выводит ход компьютера на экран. userMove предоставляет возможность игроку сделать ход. chooseWinner выбирает победителя или объявляет о ничье в случае, если игрок или компьютер не может сделать следующий ход. hasWinORow определяет, содержит ли поле строку из ноликов. hasWinOCol определяет, содержит ли поле столбец из ноликов. hasWinODiag определяет, содержит ли поле диагональ из ноликов. hasWinXRow, hasWinXCol, hasWinXDiag работают так же, как и hasWinORow, hasWinOCol, hasWinODiag соответственно, с той лишь разницей, что они проверяют поле на наличие клеток с крестиками, а не ноликами. atLeastOneEmpty проверяет, содержит ли поле хотя бы одну пустую клетку. movePossible проверяет, можно ли сделать шаг компьютеру или игроку. checkRows проверяет, не содержит ли поле строку только из ноликов или только из крестиков. checkColumns и checkDiags работают аналогичным образом, с той лишь разницей, что они проверяют столбцы и диагонали соответственно, а не строки. fineshedRow проверяет, является ли строка состоящей только из ноликов или только из крестиков. writeByLine построчно выводит поле на экран. set ставит в определенной ячейке поля крестик или нолик.

Вывод

В ходе выполнения данной лабораторной работы я изучил правила использования отсечения в языке Пролог.

Приложение А

Листинг 1. Предикат makeStep и вспомогательные предикаты

```
makeStep([[x,a,a],[a,a,a],[a,a,a]], 2,2).
makeStep([[a,x,a],[a,a,a],[a,a,a]], 2,2).
makeStep([[a,a,x],[a,a,a],[a,a,a]], 2,2).
makeStep([[a,a,a],[x,a,a],[a,a,a]], 2,2).
makeStep([[a,a,a],[a,a,x],[a,a,a]], 2,2).
makeStep([[a,a,a],[a,a,a],[x,a,a]], 2,2).
makeStep([[a,a,a],[a,a,a],[a,x,a]], 2,2).
makeStep([[a,a,a],[a,a,a],[a,a,x]], 2,2).

makeStep([[a,a,a],[a,x,a],[a,a,a]], 1,1).

makeStep([[a,a,a],[a,a,a],[a,a,a]],1,1).

makeStep([[o,a,a],[a,x,a],[a,a,a]],3,3).

makeStep([[o,x,a],[a,a,a],[a,a,a]],3,1).
makeStep([[o,a,x],[a,a,a],[a,a,a]],3,1).
makeStep([[o,a,a],[x,a,a],[a,a,a]],1,3).
makeStep([[o,a,a],[a,a,x],[a,a,a]],1,3).
makeStep([[o,a,a],[a,a,a],[x,a,a]],1,3).
makeStep([[o,a,a],[a,a,a],[a,x,a]],1,3).
makeStep([[o,a,a],[a,a,a],[a,a,x]],1,3).

makeStep([[x,a,a],[a,o,a],[a,a,x]], 1,2).
makeStep([[a,a,x],[a,o,a],[x,a,a]], 1,2).

makeStep([[a,x,a],[a,o,a],[a,x,a]], 1,3).
makeStep([[a,a,a],[x,o,x],[a,a,a]], 1,3).

makeStep(Field, Row, Column) :-
    tryToWin(Field, Row, Column).
makeStep(Field, Row, Column) :-
    tryNotLoose(Field, Row, Column).
makeStep(Field, Row, Column) :-
    fillNext(Field, Row, Column).
makeStep(Field, Row, Column) :-
    moveToEmpty(Field, Row, Column).

tryToWin(Field, Row, Column) :-
    findWinningRow(Field, Row, Column),!.
tryToWin(Field, Row, Column) :-
    findWinningColumn(Field, Row, Column),!.
tryToWin(Field, Row, Column) :-
    findWinningDiag(Field, Row, Column),!.

tryNotLoose(Field, Row, Column) :-
    findLoosingRow(Field, Row, Column),!.
tryNotLoose(Field, Row, Column) :-
    findLoosingColumn(Field, Row, Column),!.
tryNotLoose(Field, Row, Column) :-
    findLoosingDiag(Field, Row, Column),!.

fillNext(Field, Row, Column) :-
```

```
    fillRow(Field, Row, Column),!.
fillNext(Field, Row, Column) :-
    fillColumn(Field, Row, Column),!.
fillNext(Field, Row, Column) :-
    fillDiag(Field, Row, Column),!.

moveToEmpty([X,_,_], 1, Column) :-
    hasEmpty(X, Column),!.
moveToEmpty([_,X,_], 2, Column) :-
    hasEmpty(X, Column),!.
moveToEmpty([_,_,X], 3, Column) :-
    hasEmpty(X, Column),!.

hasEmpty([_,a,_], 2).
hasEmpty([a,_,_], 1).
hasEmpty([_,_,a], 3).

findWinningRow([X,_,_], 1, Column) :-
    isWinRow(X, Column),!.
findWinningRow([_,X,_], 2, Column) :-
    isWinRow(X, Column),!.
findWinningRow([_,_,X], 3, Column) :-
    isWinRow(X, Column),!.

isWinRow([a,o,o],1).
isWinRow([o,a,o],2).
isWinRow([o,o,a],3).

findWinningColumn([[X1,Y1,Z1],[X2,Y2,Z2],[X3,Y3,Z3]], Row, Column) :-
    findWinningRow([[X1,X2,X3],[Y1,Y2,Y3],[Z1,Z2,Z3]], Column, Row),!.

findWinningDiag([[X1,_,_],[_,Y2,_],[_,_,Z3]], RowAndCol, RowAndCol) :-
    isWinRow([X1,Y2,Z3], RowAndCol),!.
findWinningDiag([[_,_,Z1],[_,Y2,_],[X3,_,_]], Row, Column) :-
    isWinRow([Z1,Y2,X3], Row),
    Column is 4 - Row,!.

findLoosingRow([X,_,_], 1, Column) :-
    isLooseRow(X, Column),!.
findLoosingRow([_,X,_], 2, Column) :-
    isLooseRow(X, Column),!.
findLoosingRow([_,_,X], 3, Column) :-
    isLooseRow(X, Column),!.

isLooseRow([a,x,x],1).
isLooseRow([x,a,x],2).
isLooseRow([x,x,a],3).

findLoosingColumn([[X1,Y1,Z1],[X2,Y2,Z2],[X3,Y3,Z3]], Row, Column) :-
    findLoosingRow([[X1,X2,X3],[Y1,Y2,Y3],[Z1,Z2,Z3]], Column, Row),!.

findLoosingDiag([[X1,_,_],[_,Y2,_],[_,_,Z3]], RowAndCol, RowAndCol) :-
    isLooseRow([X1,Y2,Z3], RowAndCol),!.
findLoosingDiag([[_,_,Z1],[_,Y2,_],[X3,_,_]], Row, Column) :-
    isLooseRow([Z1,Y2,X3], Row),
    Column is 4 - Row,!.
```

```
fillRow([X,_,_], 1, Column) :-
    rowToFill(X, Column).
fillRow([_,X,_], 2, Column) :-
    rowToFill(X, Column).
fillRow([_,_,X], 3, Column) :-
    rowToFill(X, Column).

rowToFill([o,a,a],3).
rowToFill([a,o,a],1).
rowToFill([a,a,o],1).

fillColumn([[X1,Y1,Z1],[X2,Y2,Z2],[X3,Y3,Z3]], Row, Column) :-
    fillRow([[X1,X2,X3],[Y1,Y2,Y3],[Z1,Z2,Z3]], Column, Row),!.

fillDiag([[X1,_,_],[_,Y2,_],[_,_,Z3]], RowAndCol, RowAndCol) :-
    rowToFill([X1,Y2,Z3], RowAndCol),!.
fillDiag([[_,_,Z1],[_,Y2,_],[X3,_,_]], Row, Column) :-
    rowToFill([Z1,Y2,X3], Row),
    Column is 4 - Row,!.
```

Листинг 2

```
initField([[a,a,a],[a,a,a],[a,a,a]]).

userFirst() :-
    initField(Field),
    userFirstLoop(Field),!.

computerFirst() :-
    initField(Field),
    computerFirstLoop(Field),!.

computerFirstLoop(Field) :-
    computerMove(Field, Field2),
    userFirstLoop(Field2),!.

userFirstLoop(Field) :-
    movePossible(Field),
    userMove(Field, Field2),
    computerMove(Field2, Field3),
    userFirstLoop(Field3),!.

userFirstLoop(Field) :-
    \+ movePossible(Field),
    chooseWinner(Field, Winner),
    format('~w~n', [Winner]),!.

computerMove(Field, Field) :-
    \+ movePossible(Field),!.

computerMove(InField, OutField) :-
    movePossible(InField),
    makeStep(InField, Row, Column),
    set(InField, Row, Column, o, OutField),
    format('Computer\'s move:~n'),
    writeByLine(OutField),!.
```

```
userMove(InField, OutField) :-
    format('Make your move:~n'),
    read([R,C]),
    set(InField, R, C, x, OutField),
    format('Your move:~n'),
    writeByLine(OutField),!.

chooseWinner(Field, 'Computer won.') :-
    (hasWinORow(Field);
     hasWinOCol(Field);
     hasWinODiag(Field)),!.
chooseWinner(Field, 'You won.') :-
    (hasWinXRow(Field);
     hasWinXCol(Field);
     hasWinXDiag(Field)),!.
chooseWinner(_, 'We have a tie!').

hasWinORow([X,Y,Z]) :-
    (winORow(X);
     winORow(Y);
     winORow(Z)),!.

hasWinOCol([[X1,Y1,Z1],[X2,Y2,Z2],[X3,Y3,Z3]]) :-
    (winORow([X1,X2,X3]);
     winORow([Y1,Y2,Y3]);
     winORow([Z1,Z2,Z3])),!.

hasWinODiag([[X1,_,Z1],[_,Y2,_],[X3,_,Z3]]) :-
    (winORow([X1,Y2,Z3]);
     winORow([Z1,Y2,X3])),!.

winORow([o,o,o]).

hasWinXRow([X,Y,Z]) :-
    (winXRow(X);
     winXRow(Y);
     winXRow(Z)),!.

hasWinXCol([[X1,Y1,Z1],[X2,Y2,Z2],[X3,Y3,Z3]]) :-
    (winXRow([X1,X2,X3]);
     winXRow([Y1,Y2,Y3]);
     winXRow([Z1,Z2,Z3])),!.

hasWinXDiag([[X1,_,Z1],[_,Y2,_],[X3,_,Z3]]) :-
    (winXRow([X1,Y2,Z3]);
     winXRow([Z1,Y2,X3])),!.

winXRow([x,x,x]).

atLeastOneEmpty([[a,_,_],[_,_,_],[_,_,_]]).
atLeastOneEmpty([[_,a,_],[_,_,_],[_,_,_]]).
atLeastOneEmpty([[_,_,a],[_,_,_],[_,_,_]]).
atLeastOneEmpty([[_,_,_],[a,_,_],[_,_,_]]).
atLeastOneEmpty([[_,_,_],[_,a,_],[_,_,_]]).
atLeastOneEmpty([[_,_,_],[_,_,a],[_,_,_]]).
```

```
atLeastOneEmpty([[_,_,_],[_,_,_],[a,_,_]]).
atLeastOneEmpty([[_,_,_],[_,_,_],[_,a,_]]).
atLeastOneEmpty([[_,_,_],[_,_,_],[_,_,a]]).

movePossible(Field) :-
    atLeastOneEmpty(Field),
    checkRows(Field),!,
    checkColumns(Field),!,
    checkDiags(Field),!.

checkRows([X,Y,Z]) :-
    checkRow(X),!,
    checkRow(Y),!,
    checkRow(Z),!.

checkDiags([[X1,_,Z1],[_,Y2,_],[X3,_,Z3]]) :-
    checkRow([X1,Y2,Z3]),!,
    checkRow([Z1,Y2,X3]),!.

checkColumns([[X1,Y1,Z1],[X2,Y2,Z2],[X3,Y3,Z3]]) :-
    checkRow([X1,X2,X3]),!,
    checkRow([Y1,Y2,Y3]),!,
    checkRow([Z1,Z2,Z3]),!.

checkRow(X) :-
    \+ finishedRow(X).

finishedRow([x,x,x]).
finishedRow([o,o,o]).

writeByLine([Row1,Row2,Row3]) :-
    format('~w~n~w~n~w~n', [Row1,Row2,Row3]).

set([[a ,Y1,Z1],[X2,Y2,Z2],[X3,Y3,Z3]],1,1,E,[[E ,Y1,Z1],[X2,Y2,Z2],[X3,Y3,Z3]]).
set([[X1,a ,Z1],[X2,Y2,Z2],[X3,Y3,Z3]],1,2,E,[[X1,E ,Z1],[X2,Y2,Z2],[X3,Y3,Z3]]).
set([[X1,Y1,a ],[X2,Y2,Z2],[X3,Y3,Z3]],1,3,E,[[X1,Y1,E ],[X2,Y2,Z2],[X3,Y3,Z3]]).
set([[X1,Y1,Z1],[a ,Y2,Z2],[X3,Y3,Z3]],2,1,E,[[X1,Y1,Z1],[E ,Y2,Z2],[X3,Y3,Z3]]).
set([[X1,Y1,Z1],[X2,a ,Z2],[X3,Y3,Z3]],2,2,E,[[X1,Y1,Z1],[X2,E ,Z2],[X3,Y3,Z3]]).
set([[X1,Y1,Z1],[X2,Y2,a ],[X3,Y3,Z3]],2,3,E,[[X1,Y1,Z1],[X2,Y2,E ],[X3,Y3,Z3]]).
set([[X1,Y1,Z1],[X2,Y2,Z2],[a ,Y3,Z3]],3,1,E,[[X1,Y1,Z1],[X2,Y2,Z2],[E ,Y3,Z3]]).
set([[X1,Y1,Z1],[X2,Y2,Z2],[X3,a ,Z3]],3,2,E,[[X1,Y1,Z1],[X2,Y2,Z2],[X3,E ,Z3]]).
set([[X1,Y1,Z1],[X2,Y2,Z2],[X3,Y3,a ]],3,3,E,[[X1,Y1,Z1],[X2,Y2,Z2],[X3,Y3,E ]]).
```