|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Информатика и системы управления\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

КАФЕДРА \_\_\_\_\_\_\_\_Программное обеспечение ЭВМ и информационные технологии\_\_\_\_\_\_\_\_\_

**ОТЧЕТ**

**по Лабораторной работе №1**

**по курсу**

**«Проектирование компиляторов»**

**Тема**

**«Распознавание цепочек регулярного языка»**

**Вариант 4**

Студент \_\_\_ИУ7-21М\_\_\_\_ \_\_\_Карпухин А.С.\_\_\_

(Группа) (И.О.Фамилия)

Преподаватель \_\_\_\_Ступников\_\_\_\_\_\_

(И.О.Фамилия)

*2020 г.*

**Задание**

Необходимо реализовать программу, которая в качестве входа принимает произвольное регулярное выражение, и выполняет следующие преобразования:

1. По регулярному выражению строит недетерминированный конечный автомат (НКА);
2. По полученному НКА строит эквивалентный ему детерминированный конечный автомат (ДКА);
3. По ДКА строит эквивалентный ему конечный автомат, имеющий наименьшее возможное количество состояний;
4. Моделирует минимальный конечный автомат для входной цепочки из терминалов исходной грамматики.

Для минимизации использовать алгоритм Бржозовского. Данный алгоритм применим как для детерминированных, так и для недетерминированных КА. Он основан на теореме, утверждающей, что для некоторого КА A минимальным КА будет автомат d(r(d(r(A)))), где r(A) – КА, обратный к A, d(A) – детерминированный КА, полученный из A.

**Листинги кода**

Листинг 1. Удаление лямбда-переходов.

std::set<std::shared\_ptr<Arc>> FiniteStateMachine::\_bfs(

const std::shared\_ptr<State>& init,

std::set<std::shared\_ptr<State>>& finalStates)

{

std::queue<std::shared\_ptr<Arc>> queue;

std::set<std::shared\_ptr<State>> marked;

std::set<std::shared\_ptr<Arc>> arcs;

queue.push(std::shared\_ptr<Arc>(new Arc(init, init)));

while (!queue.empty())

{

auto arc = queue.front();

auto& state = arc->getFinalState();

queue.pop();

if (arc->getType() != ArcType::Lambda)

{

arcs.insert(std::shared\_ptr<Arc>(

new Arc(init, state, arc->getType(), arc->getMark())));

}

else

{

if (\_finalStates.find(state) != \_finalStates.end())

{

finalStates.insert(init);

}

if (marked.find(state) != marked.end())

{

continue;

}

marked.insert(state);

std::set<std::shared\_ptr<Arc>> successors;

std::copy\_if(\_arcs.begin(), \_arcs.end(),

std::inserter(successors, successors.end()), [&](auto v) {

return v->getInitialState() == state;

});

for (auto& succ : successors) { queue.push(succ); }

}

}

return arcs;

}

void FiniteStateMachine::\_removeLambda()

{

\_writeToFile();

std::set<std::shared\_ptr<Arc>> newArcs;

std::set<std::shared\_ptr<State>> newStates;

std::set<std::shared\_ptr<State>> newFinalStates;

std::copy\_if(\_states.begin(), \_states.end(),

std::inserter(newStates, newStates.end()), [&](auto state) {

return state == \_initState || std::any\_of(\_arcs.begin(), \_arcs.end(), [&](auto arc) {

return arc->getFinalState() == state && arc->getType() != ArcType::Lambda;

});

});

for (auto& state : newStates)

{

std::set<std::shared\_ptr<Arc>> arcs = \_bfs(state, newFinalStates);

newArcs.insert(arcs.begin(), arcs.end());

}

\_states = std::move(newStates);

\_arcs = std::move(newArcs);

\_finalStates = std::move(newFinalStates);

\_writeToFile();

}

Листинг 2. Детерминизация КА.

void FiniteStateMachine::\_determine()

{

FiniteStateMachine machine;

\_initState->addInnerState(\_initState);

machine.\_initState = \_initState;

\_determineRecur(machine, \_initState);

std::copy\_if(machine.\_states.begin(), machine.\_states.end(),

std::inserter(machine.\_finalStates, machine.\_finalStates.end()), [&](auto state) {

return std::any\_of(state->getInnerStates().begin(), state->getInnerStates().end(), [&](auto s) {

return \_finalStates.find(s) != \_finalStates.end();

});

});

machine.\_clearInnerStates();

\*this = std::move(machine);

}

void FiniteStateMachine::\_determineRecur(

FiniteStateMachine& machine, const std::shared\_ptr<State>& state)

{

if (!state)

{

throw std::invalid\_argument("Null state pointer.");

}

std::set<std::shared\_ptr<Arc>> arcs;

machine.\_states.insert(state);

// Находим все дуги, исходящие из данного множества состояний

for (auto& innerState : state->getInnerStates())

{

std::copy\_if(\_arcs.begin(), \_arcs.end(), std::inserter(arcs, arcs.end()), [&](auto v) {

return v->getInitialState() == innerState;

});

}

std::set<char> marks;

for (auto& arc : arcs)

{

if (marks.find(arc->getMark()) != marks.end()) continue;

marks.insert(arc->getMark());

std::set<std::shared\_ptr<Arc>> oneMarkArcs;

std::set<std::shared\_ptr<State>> states;

// Определяем множество дуг с одинаковыми метками

std::copy\_if(arcs.begin(), arcs.end(),

std::inserter(oneMarkArcs, oneMarkArcs.end()), [&](auto v) {

return v->getMark() == arc->getMark();

});

// Определяем множество состояний, в которые есть переход по данной метке

for (auto& arc : oneMarkArcs) { states.insert(arc->getFinalState()); }

// Ищем состояние, являющееся таким же множеством состояний в новом КА

auto it = std::find\_if(machine.\_states.begin(), machine.\_states.end(), [&](auto v) {

return v->getInnerStates() == states;

});

// Если оно уже есть, добавляем дугу из нового состояния в данное

if (it != machine.\_states.end())

{

machine.\_arcs.insert(std::shared\_ptr<Arc>(

new Arc(state, \*it, arc->getType(), arc->getMark())));

}

// Иначе добавляем состояние с данным множеством состояний и

// дугу в него, а также ищем все состояния, связанные с новым

else

{

auto newState = std::shared\_ptr<State>(new State);

newState->addInnerState(states);

machine.\_states.insert(newState);

machine.\_arcs.insert(std::shared\_ptr<Arc>(

new Arc(state, newState, arc->getType(), arc->getMark())));

\_determineRecur(machine, newState);

}

}

}

Листинг 3. Построение обратного КА и минимизация.

void FiniteStateMachine::\_execDebug(const std::function<void()>& action, const std::string& message)

{

action();

\_writeToFile();

if (!Interface::isDebugOn()) return;

std::cout << std::endl << message << std::endl;

system("pause");

}

void FiniteStateMachine::minimize()

{

\_execDebug([&]() {}, "Initial FSA printed to file");

\_execDebug([&]() { \_reverse(); }, "Reversed FSA printed to file");

\_execDebug([&]() { \_removeLambda(); }, "FSA without lambdas printed to file");

\_execDebug([&]() { \_determine(); }, "Determined FSA printed to file");

\_execDebug([&]() { \_reverse(); }, "Reversed FSA printed to file");

\_execDebug([&]() { \_removeLambda(); }, "FSA without lambdas printed to file");

\_execDebug([&]() { \_determine(); }, "Determined FSA printed to file");

}

void FiniteStateMachine::\_reverse()

{

std::set<std::shared\_ptr<Arc>> newArcs;

for (auto& arc : \_arcs)

{

newArcs.insert(std::shared\_ptr<Arc>(new Arc(

arc->getFinalState(),

arc->getInitialState(),

arc->getType(),

arc->getMark())));

}

\_arcs = std::move(newArcs);

if (\_finalStates.size() == 1)

{

auto state = \_initState;

\_initState = \*\_finalStates.begin();

\_finalStates = { state };

}

else

{

auto state = \_initState;

\_initState = std::shared\_ptr<State>(new State());

\_states.insert(\_initState);

for (auto& finalState : \_finalStates)

{

\_arcs.insert(std::shared\_ptr<Arc>(new Arc(\_initState, finalState)));

}

\_finalStates = { state };

}

}

**Тестирование**

|  |  |  |  |
| --- | --- | --- | --- |
| **Регулярное выражение** | **Входная цепочка** | **Ожидаемый результат** | **Полученный результат** |
| a | a | Match | Match |
| a | aa | Not match | Not match |
| a | Пустая строка | Not match | Not match |
| ab | ab | Match | Match |
| ab | a | Not match | Not match |
| ab | b | Not match | Not match |
| ab | abc | Not match | Not match |
| abc | abc | Match | Match |
| abc | ab | Not match | Not match |
| abc | a | Not match | Not match |
| abc | abcc | Not match | Not match |
| a|b | a | Match | Match |
| a|b | b | Match | Match |
| a|b | ab | Not match | Not match |
| ab|bc|cd|de | ab | Match | Match |
| ab|bc|cd|de | bc | Match | Match |
| ab|bc|cd|de | cd | Match | Match |
| ab|bc|cd|de | de | Match | Match |
| ab|bc|cd|de | abbc | Not match | Not match |
| a\* | Пустая строка | Match | Match |
| a\* | a | Match | Match |
| a\* | aa | Match | Match |
| a\* | b | Not match | Not match |
| a\* | ab | Not match | Not match |
| a+ | Пустая строка | Not match | Not match |
| a+ | a | Match | Match |
| a+ | aa | Match | Match |
| a+ | b | Not match | Not match |
| a+ | ab | Not match | Not match |
| . | Пустая строка | Not match | Not match |
| . | Любой одиночный символ | Match | Match |
| . | ab | Not match | Not match |
| a{1,3} | Пустая строка | Not match | Not match |
| a{1,3} | a | Match | Match |
| a{1,3} | aa | Match | Match |
| a{1,3} | aaa | Match | Match |
| a{1,3} | aaaa | Not match | Not match |
| a? | Пустая строка | Match | Match |
| a? | a | Match | Match |
| a? | aa | Not match | Not match |
| (a|b)\* | a | Match | Match |
| (a|b)\* | b | Match | Match |
| (a|b)\* | ab | Match | Match |
| (a|b)\* | ba | Match | Match |
| (a|b)\* | abab | Match | Match |
| (a|b)\* | aabab | Match | Match |
| (a|b)+ | Пустая строка | Not match | Not match |
| (a|b)+ | a | Match | Match |
| (a|b)+ | b | Match | Match |
| (a|b)+ | ab | Match | Match |
| (a|b)? | Пустая строка | Match | Match |
| (a|b)? | a | Match | Match |
| (a|b)? | b | Match | Match |
| (a|b)? | ab | Not match | Not match |
| (a|b){1,3} | Пустая строка | Not match | Not match |
| (a|b){1,3} | a | Match | Match |
| (a|b){1,3} | ab | Match | Match |
| (a|b){1,3} | aba | Match | Match |
| (a|b){1,3} | abab | Not match | Not match |
| (a|b.c|c\*d|d+f|k?){2,3} | a | Match | Match |
| (a|b.c|c\*d|d+f|k?){2,3} | abc | Not match | Not match |
| (a|b.c|c\*d|d+f|k?){2,3} | abfc | Match | Match |
| (a|b.c|c\*d|d+f|k?){2,3} | abfcd | Match | Match |
| (a|b.c|c\*d|d+f|k?){2,3} | abfcccccd | Match | Match |
| (a|b.c|c\*d|d+f|k?){2,3} | fk | Not match | Not match |
| (a|b.c|c\*d|d+f|k?){2,3} | dfk | Match | Match |
| (a|b.c|c\*d|d+f|k?){2,3} | addddfk | Match | Match |
| (a|b.c|c\*d|d+f|k?){2,3} | aaddddfk | Not match | Not match |

**Заключение**

В результате выполнения лабораторной работы была реализована программа, выполняющая построение недетерминированного конечного автомата по регулярному выражению, детерминизацию полученного НКА и его минимизацию, а затем полученный минимальный КА использовался для проверки произвольной входной цепочки на соответствие регулярному выражению, то есть на принадлежность регулярному языку, порождаемому построенным в результате моделирования конечным автоматом.

Программа допускает наличие следующих конструкций в регулярном выражении (и любых их комбинаций):

* Символьный литерал;
* Произвольный символьный литерал (символ «.»);
* Последовательность конструкций (конкатенация);
* Конструкция «или» (символ «|»);
* Произвольное число повторений (символ «\*»);
* Ненулевое число повторений (символ «+»);
* Нуль или единственное вхождение (символ «?»);
* Число вхождений, специфицированное диапазоном (конструкция вида {m, n}).

Для реализованного ПО были выполнены тесты, список которых был приведен в разделе «Тестирование».

**Список использованной литературы**