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 ТЕОРЕТИЧЕСКАЯ ИНФОРМАТИКА И КОМПЬЮТЕРНЫЕ ТЕХНОЛОГИИ

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К КУРСОВОМУ ПРОЕКТУ
ПО ДИСЦИПЛИНЕ
«Конструирование компиляторов»
НА ТЕМУ:
Интерпретатор формул TeX (статическая проверка типов)

Руководитель курсового проекта _____ Коновалов А.В.
(Подпись, дата) (Фамилия И.О.)

Исполнитель курсового проекта ИУ9-72Б _____ Васянович Д. С.
(Группа) (Подпись, дата) (Фамилия И.О.)

Москва
2022

Содержание

Введение

TeX является системой компьютерной верстки, созданной в целях создания компьютерной типографии.

По сравнению с базовыми текстовыми процессорами, TeX позволяет пользователю задавать лишь текст и его структуру, форматируя самостоятельно на основе выбранного пользователем шаблона документ. TeX файлы набираются на собственном языке разметки в виде $ASCII$-файлов, содержащих информацию о форматировании текста или выводе изображений.

Ядро TeX представляет собой язык низкоуровневой разметки, содержащий команды отступа и смены шрифта, имеющий множество готовых наборов макросов и расширений. Наиболее распространённым расширением TeX является $LaTeX$.

Данный курсовой проект является продолжением работ Натальи Стрельниковой и Нурсултана Эсенбаева. В данных работах был разработан препроцессор формул TeX. Препроцессор является средством автоматического вычисления расчетов в документах TeX. В понятие "расчет"входят:

1. Арифметические выражения
2. Векторы и матрицы, а также арифметические операции с ними (с учетом проверки их размерностей: количества строк и столбцов)
3. Базовые математические функции
4. Циклы
5. Условные операторы
6. Двумерные графики функций
7. Пользовательские функции, в том числе высшего порядка
8. Операторы суммы и произведения
9. Оператор модуля

10. Операторы округления в большую/меньшую сторону

11. Физические (размерные) величины, а также операции с ними

Препроцессор использует файл с определениями вспомогательных команд ($preproc.tex$) для исполнения расчетов в целевом $.tex$ файле. Для этого необходимо включить в данный файл вспомогательный файл препроцессора с помощью

```
\input {preproc.tex}
```

Для того, чтобы препроцессор обработал расчеты, их требуется включить в окружение $preproc$ в файле $preproc.tex$ с помощью

```
\newenvironment{preproc}
{\begin{equation*}\begin{array}{l}}
{\end{array}\end{equation*}}
```

Препроцессор принимает на вход файл с расширением $.tex$ в языковой разметке TeX и преобразует формулы

```
x = \placeholder[]{}
```

вычисляя значения и замещая их в новом порождающем файле. Таким образом, препроцессор позволяет одновременного производить вычисления в расчетах и документирование отчетов.

1 Постановка задачи

Целью данной курсовой работы является реализация статической проверки типов данных (скаляр, функция, матрица) и размерностей физических величин, ранее реализованных в проекте, с помощью введения стадии семантического анализа.

В текущей реализации обработка файлов производится следующим образом:

1. Включение в исходный файл файла с определениями для препроцессора и выделение нужных областей текста в блоки окружения для их дальнейшей обработки препроцессором.
2. Выделение в тексте исходного файла участков текста, включенных в блоки окружения, для дальнейшей обработки препроцессором.
3. Стадия лексического анализа входного текста: разбиение на лексемы для следующей стадии
4. Стадия синтаксического анализа лексем: построение AST
5. Стадия непосредственного вычисления значений в $placeholder$-ах.
6. Сохранение модифицированного текста с произведенными расчетами в исходный или новый файл (в зависимости от аргументов, данных программе на вход).

Требуется добавить стадию семантического анализа между стадией синтаксического анализа и стадией непосредственного вычисления.

2 Программная реализация

Стадия семантического анализа необходима для выявления ошибок в операциях с различными типами данных (скаляр, функция, матрица) и величинами, имеющие физические размерностями, на более ранней стадии анализа исходного текста, что позволяет раньше выявить несоотвествия, если данные имеются, и завершить обработку исходного текста с выбросом исключения. В текущей реализации данная проверка выполняется во время выполнения стадии непосредственного вычисления значений формул, что может замедлить выполнение программы в случае наличия ошибок. Введение статической проверки размерностей величин позволит избежать лишних потерь во времени исполнения программы.

2.1 Изменения в структуре стадий анализа

Последовательностей стадей вызова анализа дополнилась семантическим анализом

```
...
Parser B;
...
Lexer l;
try {
        std::vector<Token> p =
          l.program_to_tokens(Position::ps);
        ...
        B.init(p);
        res = new Node();
        res->fields = B.block(NONE);
        res->set\_tag(ROOT);
        res->print("");

    // Semantic analysis stage for checking
       correctness of operations with physical
```

```
        dimensions
          res->semantic_analysis();
          res->exec();
          std::string replacement =
            make_replacement(Position::ps.program,
            Node::reps);
          fh.print_to_out(replacement);
          Node::reps.clear();
}
...
```

2.2 Стадия семантического анализа

Все проверки типов и размерностей физических величин, которые раньше выполнялись в runtime стадии, теперь реализованы в вызове функции

```
void Node::semantic_analysis();
```

Функция изначально вызывается для вершины синтаксического дерева, построенного в предыдущей стадии, и, в зависимости от типа вершины, для которой был вызван данный метод (он является рекурсивным), происходит семантический анализ соотвествующих поддеревьев.

1. Для корневой вершины ($\_tag == ROOT$) и вершины начала блока ($\_tag == BEGINB$) идет рекурсивный вызов по всем $field$ из $fields$

2. В случае оператора произведения ($\_tag == PRODUCT$), или оператора суммирования ($\_tag == SUM$), или оператора возведения в степень ($\_tag == POW$) происходит вызов функции

   ```
   Coordinate* Node::have_dimensions(Coordinate*
     coordinate);
   ```

   Она проверяет, нет ли среди аргументов оператора произведения размерных физических величин. Для оператора суммирования идет проверка только по значениям нижней и верхней границ суммирования. В случае,

если таковые имеются, выводится сообщение об ошибке с координатой первого такого аргумента:

   (a) $Element\ of\ product\ is\ not\ allowed\ to\ be\ dimensional$ в случае оператора произведения.

   (b) $Lower\ or\ higher\ bound\ of\ sum\ is\ not\ allowed\ to\ be\ dimensional$ в случае оператора суммирования.

   (c) $Power\ of\ dimensional\ number\ is\ not\ defined$ в случае оператора возведения в степень.

3. В случае всевозможных операторов сравнения, а также операторы сложения и вычитания:

   (a) Больше ($\_tag == GT$)

   (b) Меньше ($\_tag == LT$)

   (c) Больше или равно ($\_tag == LEQ$)

   (d) Меньше или равно ($\_tag == GEQ$)

   (e) Равно ($\_tag == EQ$)

   (f) Не равно ($\_tag == NEQ$)

   (g) Сложение ($\_tag == ADD$)

   (h) Вычитание ($\_tag == SUB$)

   Происходит вызов функции

```
int* Node::calculate_dimensions(int* dims, bool
    is_mul);
```

   Которая считает размерность левой/правой части (вызов происходит для обеих частей выражения, соотвественно). После этого реализовано сравнение размерностей посредством функции

```
bool check_dimensions(const int* first, const int*
    second);
```

   В случае несоотвествия размерностей будет выброшено исключение с сообщением:

(a) $Comparison\ of\ different\ dimensions$ для операторов сравнения

(b) $Addition\ of\ different\ dimensions$ для оператора сложения

(c) $Subtraction\ of\ different\ dimensions$ для оператора вычитания

Также, для операторов сложения ($\_tag == ADD$) и вычитания ($\_tag == SUB$) реализована проверка типов с помощью вызова функции

```
std::string& Node::get_type(std::string& tag);
```

Которая проверяет отсутствие попыток сложения или вычитания функций со скалярами, матрицами/векторами или размерными величинами.

4. В случае унарного оператора вычитания ($\_tag == USUB$) и операции умножения ($\_tag == MUL$) также реализована проверка типов с помощью вызова функции

   ```
   std::string& Node::get_type(std::string& tag);
   ```

   Которая проверяет отсутствие попыток унарного вычитания или умножения функций.

5. В случае оператора деления ($\_tag == DIV$) или ($\_tag == FRAC$) также реализована проверка типов с помощью вызова функции

   ```
   std::string& Node::get_type(std::string& tag);
   ```

   Которая проверяет отсутствие попыток деления функций или деления на матрицу.

Функция $have\_dimensions$, представленная выше, рекурсивно проверяет поддерево вершины, для которой была вызвана, на наличие лексем с $\_tag == DIMENSION$.

Функция $calculate\_dimensions$, представленная выше, рекурсивно проходит по поддереву вершины, для которой была вызвана. В случае обнаружения оператора умножения ($\_tag == MUL$) найденные размерности будут добавлены в числитель итоговой размерности, а в случае оператора деления ($\_tag == DIV$) найденные размерности будут добавлены в знаменатель итоговой размерности. В случае же нахождения листовой размерной величины

($\_tag == DIMENSION$) происходит непосредственное обновление выходной размерной величины (в зависимости от параметра is_mul, определяющего, в числитель или в знаменатель стоит заносить листовую размерность).

Функция $check\_dimensions$, представленная выше, сравнивает вектора размерностей физических величин для выявления факта равенства самих физических величин.

Функция get_type, представленная выше, рекурсивно проверяет поддерево вершины, для которой была вызвана, на наличие лексем типа функций ($\_tag == FUNC$) и типа матриц ($\_tag == IDENT$ и поле $fields$ не пусто)

3 Изменения в стадии непосредственного вычисления

Так как проверки размерностей физических величин и типов были перенесены в стадию семантического анализа, на стадии синтаксического анализа они более не актуальны, потому были произведены следующие изменения

1. Оператор сложения $plus$

   ```
   static Value plus(const Value &left, const Value
     &right, const Coordinate& pos);
   ```

   Более не производит проверку типов данных и размерностей величин перед их сложением.

2. Операторы вычитания sub и $usub$

   ```
   static Value sub(const Value &left, const Value
     &right, const Coordinate& pos);
   static Value usub(const Value &arg, const
     Coordinate& pos);
   ```

   Более не производит проверку типов данных размерностей величин перед их вычитанием.

3. Операторы сравнения eq

   ```
   static Value eq(const Value &left, const Value
     &right, const Coordinate& pos);
   ```

   Более не производит проверку размерностей величин перед их сравнением.

4. Операторы сравнения le

   ```
   static Value le(const Value &left, const Value
     &right, const Coordinate& pos);
   ```

   Более не производит проверку размерностей величин перед их сравнением.

5. Операторы сравнения ge

```
static Value ge(const Value &left, const Value
  &right, const Coordinate& pos);
```

Более не производит проверку размерностей величин перед их сравнением.

6. Операторы сравнения lt

```
static Value lt(const Value &left, const Value
  &right, const Coordinate& pos);
```

Более не производит проверку размерностей величин перед их сравнением.

7. Операторы сравнения gt

```
static Value gt(const Value &left, const Value
  &right, const Coordinate& pos);
```

Более не производит проверку размерностей величин перед их сравнением.

8. Операторы возведения в степень pow

```
static Value pow(const Value &left, const Value
  &right, const Coordinate& pos);
```

Более не производит проверку размерностей величин перед их сравнением.

9. Операторы умножения pow и деления div

```
static Value pow(const Value &left, const Value
  &right, const Coordinate& pos);
static Value mul(const Value &left, const Value
  &right, const Coordinate& pos);
```

Более не производит проверку типов данных перед их умножением/делением.

4 Тестирование

Проверим, что после данных изменений не пострадала процедура проверки корректности вычислений с различными типами данных и размерными величинами.

4.1 Сложение/вычитание размерных величин

Рассмотрим несколько примеров для препроцессора на сложение/вычитание физических величин:

1. $7 \cdot kg - 5 \cdot kg + 3 \cdot kg = 5 \cdot kg$
2. $7 \cdot kg - 5 \cdot cd$ — получаем ошибку $Subtraction\ of\ different\ dimensions$, так как исходные величины были разных размерностей

4.2 Умножение/деление размерных величин

Рассмотрим несколько примеров для препроцессора на умножение/деление физических величин:

1. $7 \cdot kg \cdot 5 \cdot kg \cdot 3 \cdot kg = 105 \cdot kg^3$
2. $7 \cdot kg \cdot 5 \cdot cd \cdot 3 \cdot mol = 105 \cdot kg \cdot cd \cdot mol$
3. $20 \cdot kg \ / \ 5 \cdot cd \ / \ 8 \cdot mol = 0.5000 \cdot \frac{kg}{cd \cdot mol}$

4.3 Сравнение размерных величин

Рассмотрим несколько примеров для препроцессора на сравнение физических величин:

1. $7 \cdot kg \geq 3 \cdot kg = 1$

2. $7 \cdot kg \leq 3 \cdot kg = 0$

3. $7 \cdot kg^2 \ / \ cd^3 7 \cdot kg^2 \ / \ cd^3 = 1$

4. $7 \cdot kg^2 \ / \ cd^3 \neq 17 \cdot kg^2 \ / \ cd^3 = 1$

5. $7 \cdot kg^2 \neq 17 \cdot kg^{12}$ — получаем ошибку $Comparison\ of\ different\ dimensions$, так как исходные величины были разных размерностей

4.4 Функции с размерными величинами

В данной курсовой работе препроцессор не может обрабатывать размерные величины в функциях (ввиду трудности определения факта того, возвращает ли функция размерную величину или нет, а если возвращает, то какой именно размерности). Рассмотрим несколько примеров для препроцессора с функциями, содержащими размерные операции:

- Рассмотрим функцию, определяющую максимальную высоту подъема тела при вертикальном движении

```
\begin{preproc}
    height(v0) := \begin{block}
        v0 \cdot v0 / (2 \cdot 9.81 \cdot m / (s
          * s))
      \end{block} \\
\end{preproc}
```

  Получаем ошибку $Function\ cannot\ have\ dimensional\ arguments\ or\ dimensional\ returnable\ value$, так как ускорение свободного падения имеет размерность $\frac{m}{s^2}$.

- Рассмотрим другую функцию

```
\begin{preproc}
    switch(x) := \begin{caseblock}
        1 \cdot kg \when x > 0 \\
        1 \cdot m \when x < 0 \\
```

```
        1 \otherwise
    \end{caseblock} \\
\end{preproc}
```

Получаем ошибку *Function cannot have dimensional arguments or dimensional returnable value*, так как присутствуют операции с килограммами kg и метрами m.

- Рассмотрим функцию, имеющую в теле операции с размерностями, но возвращающую безразмерную величину.

```
\begin{preproc}
    nondimensional(x) := x \cdot (kg * kg / kg *
      kg / kg / kg) \\
\end{preproc}
```

Программа ошибки не выдаст, так как килограммы сокращаются с друг другом и возвращаемое значение является скаляром.

4.5 Проверка работоспособности ранее сделанных изменений

Проверим, не поломало ли текущее нововведение старый функционал.

- Дан документ с пользовательскими физическими величинами.

```
\begin{preproc}
    N := kg \cdot m \cdot s^{-2}
    J := N \cdot m
    W := J / s
    hour := 3600 \cdot s
    kWh := 1000 \cdot W \cdot hour
    cal := 4.1868 \cdot J
    kcal := 1000 \cdot cal
    kWh = \placeholder[kcal]{}
\end{preproc}
```

Проверим, корректен ли выходной результат.

```
\begin{preproc}
    N := kg \cdot m \cdot s^{-2}
    J := N \cdot m
    W := J / s
    hour := 3600 \cdot s
    kWh := 1000 \cdot W \cdot hour
    cal := 4.1868 \cdot J
    kcal := 1000 \cdot cal
    kWh = \placeholder[kcal]{859.84523}
\end{preproc}
```

- Дан документ с пользовательскими функциями

```
\begin{preproc}
ceil(x) := \begin{block}
  x := x \cdot \pi - \pi / 2 \\
  frac := \arctan(\tan(x)) \\
  (x - frac) / \pi
\end{block} \\

ceil(0.4) = \placeholder{} \\
ceil(0.5) = \placeholder{} \\
ceil(0.6) = \placeholder{} \\
ceil(1.0) = \placeholder{} \\
ceil(2.1) = \placeholder{} \\

\graphic{ceil}{\range[0.1]{0}{4.9}}
  {(0.000000,0.000000)
}\end{preproc}
```

Проверим, корректен ли выходной результат

```
\begin{preproc}

ceil(x) := \begin{block}
  x := x \cdot \pi - \pi / 2 \\
  frac := \arctan(\tan(x)) \\
```

```
  (x - frac) / \pi
\end{block} \\

ceil(0.4) = \placeholder{0} \\
ceil(0.5) = \placeholder{0} \\
ceil(0.6) = \placeholder{0} \\
ceil(1.0) = \placeholder{0} \\
ceil(2.1) = \placeholder{2} \\

\graphic{ceil}{\range[0.1]{0}{4.9}}
  {(0.000000,0.000000)
(0.100000,0.000000)
(0.200000,0.000000)
(0.300000,0.000000)
(0.400000,0.000000)
(0.500000,0.000000)
(0.600000,0.000000)
(0.700000,0.000000)
(0.800000,0.000000)
(0.900000,0.000000)
(1.000000,0.000000)
(1.100000,1.000000)
(1.200000,1.000000)
(1.300000,1.000000)
(1.400000,1.000000)
(1.500000,1.000000)
(1.600000,1.000000)
(1.700000,1.000000)
(1.800000,1.000000)
(1.900000,1.000000)
(2.000000,2.000000)
(2.100000,2.000000)
(2.200000,2.000000)
(2.300000,2.000000)
```

```
(2.400000,2.000000)
(2.500000,2.000000)
(2.600000,2.000000)
(2.700000,2.000000)
(2.800000,2.000000)
(2.900000,2.000000)
(3.000000,3.000000)
(3.100000,3.000000)
(3.200000,3.000000)
(3.300000,3.000000)
(3.400000,3.000000)
(3.500000,3.000000)
(3.600000,3.000000)
(3.700000,3.000000)
(3.800000,3.000000)
(3.900000,3.000000)
(4.000000,4.000000)
(4.100000,4.000000)
(4.200000,4.000000)
(4.300000,4.000000)
(4.400000,4.000000)
(4.500000,4.000000)
(4.600000,4.000000)
(4.700000,4.000000)
(4.800000,4.000000)
(4.900000,4.000000)
}\end{preproc}
```

4.6 Проверка производительности

Проверим, действительно ли удалось получить выигрыщ в производительности при использовании статической проверки типов. Будут сравниваться версии программ до и после введения стадии семантического анализа. Для корректного сравнения для каждого тестового случая и для обеих версий программ будут посчитаны нижний и верхний квартили (0.25 и 0.75 квантили соотвественно), образующие интерквантильный размах и 50% доверительный интервал. Тогда медиана (0.5 квантиль) определяет типичное значение. Были проведены 9 тестовых прогонов для кода «до» и для кода «после», вычислены нижний и верхний квартили, подсчитаны медианное и среднее значения. Все значения в таблицах отсортированы по возрастанию для удобства восприятия.

1. Тестовый файл с операциями с различными типами (без физических размерностей)

```
\usepackage{amsmath}
\begin{preproc}
x := 1\\
x := 100\\
x0_\text{text} := 228\\
f(x, y) := x \cdot (x + y)\\
v := \begin{pmatrix}
1 \\ 2 \\ x0_\text{text}
\end{pmatrix}
v_0 := 666\\
v = \placeholder{}\\
\graphic{f}{\range{-1}{1}, 0.5}{}\\
\end{preproc}
```

Сводная таблица результатов тестирования программы		
	До	После
	Время выполнения, мс	Время выполнения, мс
I прогон	0.855307	0.855662
II прогон	0.874702	0.877541
III прогон	0.883116	0.882467
IV прогон	0.914353	0.909901
V прогон	0.952313	0.915867
VI прогон	0.980947	1.02759
VII прогон	1.06664	1.07737
VIII прогон	1.22965	1.08608
IX прогон	1.28665	1.13098
Нижний квартиль	0.874702	0.877541
В среднем	1.004853	0.983619
Медиана	0.952313	0.915867
Верхний квартиль	1.06664	1.07737
Среднее преимущество — 2.1131 %		
Медианное преимущество — 3.8271 %		

Заметим, что доверительные интервалы фактически совпадают, что говорит о невозможности сделать вывод и преимуществе в скорости программы и о том, что в данном тесте программы «до» и «после» показали идентичные результаты по производительности.

2. Тестовый файл с операциями с различными физическими размерностями

```
\begin{preproc}
    N := kg \cdot m \cdot s^{-2}
    J := N \cdot m
    W := J / s
    hour := 3600 \cdot s
    kWh := 1000 \cdot W \cdot hour
    cal := 4.1868 \cdot J
    kcal := 1000 \cdot cal
    kWh = \placeholder[kcal]{}
\end{preproc}
```

Сводная таблица результатов тестирования программы		
	До	После
	Время выполнения, мс	Время выполнения, мс
I прогон	0.677239	0.635662
II прогон	0.700478	0.653436
III прогон	0.705746	0.687877
IV прогон	0.730876	0.688249
V прогон	0.733832	0.739234
VI прогон	0.752174	0.756426
VII прогон	0.772144	0.782026
VIII прогон	0.815893	0.783248
IX прогон	0.838679	0.790826
Нижний квартиль	0.700478	0.653436
В среднем	0.747451	0.724109
Медиана	0.733832	0.739234
Верхний квартиль	0.772144	0.782026
Среднее преимущество — 3.1229 %		
Медианное отставание — 0.7308 %		

Заметим, что доверительные интервалы фактически совпадают, что говорит о невозможности сделать вывод и преимуществе в скорости программы и о том, что в данном тесте программы «до» и «после» показали идентичные результаты по производительности.

3. Тестовый файл со стресс тестированием без операций с физическими размерностями

```
\begin{preproc}
i := 0
step := 1
limit := 10000
x := \while{i < limit} \begin{block}
i := i + step
\end{block}
x = \placeholder{}
\end{preproc}
```

Сводная таблица результатов тестирования программы		
	До	После
	Время выполнения, мс	Время выполнения, мс
I прогон	10.7054	8.78707
II прогон	10.8698	8.94345
III прогон	10.8988	9.02633
IV прогон	11.0469	9.12695
V прогон	11.0471	9.32904
VI прогон	11.3146	9.396
VII прогон	11.4976	9.83309
VIII прогон	11.5551	10.0111
IX прогон	12.5632	11.951
Нижний квартиль	10.8698	8.94345
В среднем	11.2776	9.60045
Медиана	11.0471	9.32904
Верхний квартиль	11.4976	9.83309
Среднее преимущество — 14.8715 %		
Медианное преимущество — 15.5521 %		

4. Тестовый файл со стресс тестированием с операциями с физическими размерностями

```
\begin{preproc}
i := 0 \cdot s
step := 1 \cdot s
limit := 10000 \cdot s
x := \while{i < limit} \begin{block}
i := i + step
\end{block}
x = \placeholder{}
\end{preproc}
```

Сводная таблица результатов тестирования программы		
	До	После
	Время выполнения, мс	Время выполнения, мс
I прогон	10.8043	8.59315
II прогон	10.8558	8.7387
III прогон	10.9468	8.82267
IV прогон	10.9759	9.01302
V прогон	10.9876	9.14761
VI прогон	11.5175	9.18158
VII прогон	11.9814	9.34385
VIII прогон	12.2378	9.91211
IX прогон	12.6224	10.1084
Нижний квартиль	10.8558	8.7387
В среднем	11.4366	9.2068
Медиана	10.9876	9.14761
Верхний квартиль	11.9814	9.34385
Среднее преимущество — 19.4970 %		
Медианное преимущество — 16.7461 %		

Во всех случаях код категории «до» брался по коммиту с хэшем $9127d3e...$, а код категории «после» имел хэш $a38be43...$. Тестирование проводилось с использованием компилятора $g++$ версии $9.3.0$ с флагом $-O2$ на ПК со следующими характеристиками:

1. CPU - Intel Core i5-8265u (4/8 @ 1.60 GHz / 3.90 GHz)
2. RAM - 8 GB @ 2400 MHz
3. OS - Linux Mint 20.2 64-bit

5 Заключение

В данном курсовом проекте, в дополенение к наработкам двух предыдущих курсовых проектов, которые были взяты за основу, была реализована стадия семантического анализа для препроцессора TeX для статической проверки корректности операций с различными типами данных (матрицами и векторами, функциями, скалярами и размерными физическими величинами).

В результаты замеров производительности программы было выявлено усредненное ускорение 17.18 % по сравнению с предыдущей версией при большом количестве команд во входном файле. При малом количестве команд преимущества в производительности по сравнению с предыдущей версией не было замечено. Тестирование проводилось при различных сценариях работы и при использовании различных типов данных.

6 Список литературы

1. Документация расширения $LaTeX$

   https://www.latex-project.org/help/documentation/

2. Курсовые проекты Натальи Стрельниковой и Нурсултана Эсенбаева

   https://github.com/bmstu-iu9/tex-preprocessor

Приложение А

Файл *preproc.tex* с определениями для препроцессора *TeX*

```
\usepackage{ifthen}
\usepackage{amsmath}
\usepackage{color}
\usepackage{mathtools}
\usepackage{tikz}

\newcommand{\placeholder}[2][]{\ifthenelse{
   \equal{#1}{} }
                                        {\color{blue}#2}
                                        {\color{blue}#2
                                           \cdot #1}}
\newcommand{\ifexpr}[1]{{\bf if\enspace}#1\enspace}
\newcommand{\when}{{\enspace\bf when\enspace}}
\newcommand{\otherwise}{{\enspace\bf otherwise}}
\newcommand{\while}[1]{{\bf while\enspace}#1\enspace}
\newcommand{\true}{true}
\newcommand{\false}{false}
\newcommand{\abs}[1]{|#1|}
\DeclarePairedDelimiter{\ceil}{\lceil}{\rceil}
\DeclarePairedDelimiter{\floor}{\lfloor}{\rfloor}

\newenvironment{preproc}
{\color{red}\begin{equation*}\begin{array}{l}}
{\end{array}\end{equation*}}

\newenvironment{block}
{\begin{array}{|l}}
{\end{array}}

\newenvironment{caseblock}
{\begin{array}{|l}}
```

```
{\end{array}}

\newcommand{\range}[3][]{%
        \ifthenelse{\isempty{#1}}{[#2:#3]}{[#2:#3:#1]}%
}

\newcommand{\graphic}[3]{
\color{blue}
\begin{tikzpicture}
\begin{axis}[
        xmajorgrids,
        ymajorgrids,
        axis lines=middle,
        x label style={at={(axis description
          cs:0.5,-0.1)},anchor=north},
        ylabel style={above left},
        xlabel=\text{#1(#2)},
]
\addplot[thin, mark = none] coordinates{
        #3
};
\end{axis}
\end{tikzpicture}}
```