جامعة دمشق
كلية الهندسة المعلوماتية
قسم هندسة البرمجيات ونظم المعلومات

Compiler project
المترجمات

نادر رشيد الرشيد محمود كمال برخش

لجين حسام الجوهري سوزان عدنان ساميز

عمار عاطف العلي

بإشراف :
م.وسيم البزرة / م. ساريا الزعبي

مقدمة :

يهدف هذا التقرير إلى توثيق وشرح مكونات مشروع بناء محلل لغوي (Compiler / Parser) مخصص لتطبيقات Flask التي تعتمد على لغة Python وقوالب Jinja2 بالإضافة إلى HTML و.CSS

يركّز التقرير على شرح البنية العامة للمشروع، وآلية تحليل الشيفرة المصدرية، وبناء شجرة القواعد المجردة (AST) ، وجدول الرموز (Symbol Table) ، مع توضيح دور كل جزء.

تم إعداد هذا التقرير ليكون مرجعًا يوضح الفكرة العامة للحل البرمجي ويساعد على فهم طريقة عمل المشروع

مقدمة عن ANTLR :

ANTLR (Another Tool for Language Recognition)

هو مولد تحليل يُستخدم لإنشاء مترجمات Compilers ومحللات لغات البرمجة وتنسيقات الملفات المختلفة.
يُتيح للمطورين تحديد قواعد لغوية Grammars باستخدام قواعد تحليلية مبسطة، ثم يُنشر مولد تحليل Parserومحلل لغويLexer يستند إلى هذه القواعد.

المهام الأساسية لـANTLR :

(Lexer) محلل الأقسام: يقوم بتحويل سلسلة من الرموز من المدخلات إلى وحدات أساسية Tokens مثل الكلمات المفتاحية، المعاملات، والرموز القياسية.
(Parser) محلل الجمل: يقوم بفحص البنية اللغوية للرموز المحللة من قبل المحلل اللغوي، ويبني بنية بيانات تمثل التركيب البرمجي والعلاقات بين مكونات البرنامج.

يقوم ANTLRبتوليد كود في لغة البرمجة المستهدفة مثل (جافا) للمحلل والمولد على أساس القواعد التي يتم تحديدها. القواعد تعبر عن قاعدة اللغة المراد تحليلها.

مميزات ANTLR تشمل:

1.سهولة الاستخدام: تمتاز قواعد ANTLRبأنها قريبة من الصياغة اللغوية المألوفة، مما يجعل من السهل على المطورين تحديد قواعد لغة جديدة.

2.دعم لغات متعددة : يمكن استخدام ANTLRلتحليل لغات برمجة متنوعة أو تنسيقات ملفات أخرى.

3.تكامل مع أنظمة متعددة :يمكن توليد المحللات باستخدام ANTLRلتناسب مجموعة متنوعة من الأنظمة والمشاريع.

4.مجتمع نشط :تتميز ANTLRبوجود مجتمع نشط من المستخدمين والمطورين، مع تحديثات دورية لتحسين أدائها

I. Grammer

في مشروعي هذا، الذي يهدف إلى بناء محلل نحوي (parser) ومحلل معجمي (lexer) لتطبيقات Flask مع دعم لقوالب Jinja2 وأكواد HTML وCSS، قمت بتخصيص مجلد يُدعى "grammer" ليحتوي على ملفات القواعد النحوية المكتوبة بلغة .ANTLR4 هذا المجلد يمثل النواة الرئيسية لعملية التحليل، حيث يحدد القواعد التي تسمح للنظام بفهم وتحليل الكود المصدري بدقة . يحتوي المجلد على أربعة ملفات رئيسية، كل منها يركز على جانب معين من اللغات المستهدفة، وهي مصممة للعمل معًا لتوفير تحليل شامل .

- ملف FlaskLexer.g4

هذا الملف يعرف القواعد المعجمية (lexical rules) للغة Python كما تستخدم في إطار Flask . يشمل تعريفات للكلمات المفتاحية مثل "def" ، "if"، "return" ، بالإضافة إلى الرموز مثل الأرقام، السلاسل النصية، والعمليات الحسابية .كما يدعم الفراغات، التعليقات، والرموز الخاصة بـ Python 3، مع التركيز على العناصر التي تظهر في تطبيقات Flask مثل تعريف الدوال والمتغيرات .هذا الملف يولد التوكنز الأساسية التي يستخدمها المحلل النحوي لاحقًا.

- ملفFlaskParser.g4

يبني على الملف السابق ليعرف القواعد النحوية (syntactic rules) لتطبيقات Flask. يرث من قاعدة Python 3 الأساسية، ويشمل قواعد للعبارات مثل تعريف الدوال، الشروط، الحلقات، والاستثناءات .كما يدعم الزخارف (decorators) مثل"@app.route" ، والتي هي أساسية في Flask.هذا الملف يسمح ببناء شجرة تحليل تجريدية (AST) للكود البرمجي، مما يسهل استخراج الهيكل الرئيسي للتطبيق.

- ملف HTMLCSSJinjaLexer.g4

هذا الملف مخصص للقواعد المعجمية المتعلقة بـHTML، CSS ، و Jinja2. يعرف التوكنز مثل العلامات الافتتاحية والإغلاقية"<tag>") ،("</tag>" ، السمات(attributes) ، النصوص، والتعليقات .كما يدعم تعبيرات Jinja2 مثل "{{ variable }}" و"{% block %}"، بالإضافة إلى قواعد CSS مثل الخصائص، القيم، والتعليقات .يتعامل مع الأنماط المختلفة مثل الـ modes للتبديل بينHTML، CSS ، و Jinjaلضمان تحليل دقيق

- ملف HTMLCSSJinjaParser.g4

يعرف القواعد النحوية لدمج HTML مع CSS و Jinja2. يشمل قواعد لبناء العناصر مثل العلامات العادية والذاتية الإغلاق، المحتويات النصية، والتعليقات .كما يدعم هياكل Jinja2 مثل الشروط ("{% if %}")، الحلقات("{% while %}") ، والكتل("{% block %}") ، بالإضافة إلى قواعد CSSمثل الـselectors ، الـdeclarations ، والـ media queries. هذا الملف يبني AST للقوالب، مما يتيح التحقق من صحة الهيكل واستخراج المعلومات الديناميكية.

الأساسيات التي اعتمدناها في كتابة هذه القواعد:

اعتمدنا على عدة أساسيات لضمان الكفاءة والدقة .

- ✓ استخدمنا أداة ANTLR4 كإطار عمل رئيسي، لأنها تسمح بفصل الـ lexer عن الـparser ، مما يجعل الصيانة أسهل ويقلل من التعقيد .قسمت القواعد إلى lexer لاستخراج التوكنز الأساسية، و parserلبناء الهيكل النحوي، مع الاستفادة من الميراث من قواعد Python 3 القياسية لتجنب إعادة كتابة الأجزاء المشتركة.

- ✓ ركزنا على الدعم المتكامل للغات المستهدفة Flask :كإطارPython ي، Jinja2 للقوالب الديناميكية، و HTML/CSSللعرض .اعتمدنا على نمط الـ modes في ANTLR للتبديل بين سياقات مختلفة، مثل الدخول إلى وضع CSS عند فتح علامة"<style>" ، أو وضع Jinja عند ."{{" هذا يمنع الالتباس بين الرموز المشابهة في اللغات المختلفة.

- ✓ أقمنا بوضع اولوية الدقة في التعامل مع الفراغات، التعليقات، والأخطاء المحتملة، باستخدام قنوات مخفية (hidden channels) لتجاهل العناصر غير الضرورية أثناء التحليل .كما اعتمدت على قواعد recursive للتعامل مع الهياكل المتداخلة مثل الكتل في Jinja أو الـ selectors فيCSS ، لضمان تغطية جميع الحالات الممكنة.

- ✓ كتبنا القواعد بطريقة تجعلها قابلة للتوسع، مع التركيز على الامتثال لمعايير اللغات) مثل Python 3.12و(HTML5، واختبرناها من خلال أمثلة حقيقية من تطبيقات Flask للتأكد من فعاليتها في بناء مترجم أو محلل كامل .هذا النهج جعل المشروع أكثر قوة وسهولة في التكيف مع التغييرات المستقبلية.

II. AST

AST (Abstract Syntax Tree) هو تمثيل هيكلي للكود المصدر يستخدم في المترجمات والمحللات النحوية النظام الحالي يدعم أربع لغات متكاملة:

1. HTML لغة توصيف النصوص التشعبية
2. CSS لغة الأنماط المتتالية
3. Jinja لغة قوالب Python
4. Flask/Python إطار عمل ويب ولغة برمجة

- الهيكل المعماري

1. التصميم الهرمي:

ASTnode(الفئة الأساسية المجردة)

CSS Package(+35فئة)

HTML Package(+25فئة)

JINJA Package (+30فئة)

Flask Package(+10فئة)

2 . مبادئ التصميم

- الفصل بين المسؤوليات: كل حزمة مستقلة
- إعادة الاستخدام: فئات قابلة للتوسع
- التكامل: واجهات واضحة بين الحزم
- الأداء: تصميم خفيف الوزن

- حزمة HTML AST

الغرض: تمثيل هيكل صفحات الويب

جذر المستند → HtmlDocumentNode

العناصر العادية (<p>,) → NormalTagNode

السمات(class, id) → HtmlAttributeNode

المحتوى الداخلي → HtmlContentNode

الميزات:

- دعم HTML5 الكامل
- معالجة السمات بأنواعها المختلفة
- دعم العناصر الخاصة (CDATA, DTD)
- إدارة التسلسل الهرمي للعناصر

- حزمة CSS AST

الغرض: تمثيل قواعد وأنماط التصميم

المحددات (div, .class, #id) → SelectorNode

التصريحات (color: red) → DeclarationNode

قواعد كاملة (selector { declarations }) → RulesetNode

القواعد الخاصة(@media, @keyframes) →AtRuleNode

الميزات:

- دعم CSS3 المتقدم
- معالجة المحددات المعقدة
- دعم قواعد الوسائط والرسوم المتحركة
- معالجة الدوال والحسابات(calc(), var())

- حزمة Jinja AST

الغرض: تمثيل القوالب الديناميكية

ExpressionNode →) التعبيرات{{ variable }})

ControlNode →) هياكل التحكم{% if %}, {% for %})

CodeBlockNode →) كتل مختلطةHTML + Jinja)

AssignmentNode →) التعيينات{% set x = y %})

الميزات :

- دعم كامل لبناء جملة Jinja
- معالجة التعبيرات المنطقية والحسابية
- إدارة تدفق التحكم المعقد
- تكامل مع HTML في نفس الكتلة

- حزمة Flask AST

الغرض : تمثيل منطق تطبيقات الويب

ProgramNode → البرنامج الرئيسي

FunctionDefNode → تعريف الدوال

ControlNode →) هياكل التحكم(if, while, for)

ExpressionNode → التعبيرات البرمجية

الميزات :

- تحليل بنية تطبيقات Flask
- دتركيب دوال Python
- معالجة تدفق التحكم
- دعم الاستيراد والعبارات البرمجية

- التكامل بين الحزم

1. آليات التكامل

تحليل CSS المضمن : StyleTagNode (HTML) → StylesheetNode (CSS)

مزج القوالب مع html : CodeBlockNode (Jinja) → HtmlContentNode (HTML)

معالجة القوالب : Jinja Templates → Flask Application

2.مثال على التكامل

```
DOCTYPE html>!>
<html>
<head>
<style>
title { color: {{ theme.color }}; }.
<style/>
<head/>
<body>
<h1> class="title">{{ page_title }}</h1>
for item in items %} %}
<div>{{ item.name }}</div>
endfor %} %}
<body/>
<html/>
```

يتم تحليل هذا النموذج عبر:

1. HTML AST هيكل الصفحة والعناصر
2. CSS AST قاعدة .title مع قيمة ديناميكية
3. Jinja AST التعبيرات وحلقة for
4. Flask AST معالجة القالب وتوفير البيانات

○ التطبيقات الأكاديمية

8.1 مجالات البحث

- تحليل الكود الثابت : اكتشاف الأخطاء والثغرات
- تحويل الكود : التحويل بين التنسيقات والمستويات
- تحسين الأداء : تحليل تأثير CSS على وقت التصيير
- التوليد التلقائي : توليد أكواد من النماذج

8.2 أدوات قابلة للتطوير

1. المصحح النحوي : اكتشاف أخطاء بناء الجملة والدلالة
2. المحلل الأدائي : قياس تأثير المحددات والتعبيرات
3. المحول : تحويل بين إصدارات اللغات
4. المولد : توليد أكواد من AST

8.3 دراسات حالة

1. تحليل كفاءة : CSSدراسة تأثير تعقيد المحددات
2. تحسين قوالب : Jinjaتحليل تأثير التعابير المعقدة
3. أمان تطبيقات : Flaskاكتشاف ثغرات الحقن
4. تحسين قابلية الوصول : تحليل بنية HTML للوصول الشامل

○ التحديات والتقنيات

9.1 التحديات

- تعقيد التكامل : تنسيق أربع لغات مختلفة
- الأداء : معالجة أشجار كبيرة بكفاءة
- التوسع : إضافة لغات جديدة دون كسر التوافق
- الدقة : معالجة الحالات الخاصة والحدودية

9.2 الحلول التقنية

- نمط : Visitorلاجتياز الأشجار بكفاءة
- التسلسل إلى : JSONللتخزين والنقل
- التحليل المتوازي : لتحسين الأداء
- التصميم المعياري : للتوسع السهل

○ الاستنتاجات والتوصيات

10.1 النتائج الرئيسية

1. نجاح تصميم نظام AST متعدد اللغات
2. تحقيق تكامل فعال بين أربع لغات مختلفة
3. توفير إطار عمل للبحث والتطوير
4. بناء أساس لأدوات متقدمة لتحليل الكود

10.2 التوصيات المستقبلية

1. إضافة دعم : JavaScriptلتحليل سلوك العميل
2. تحسين الأداء : تطبيق خوارزميات تحليل متوازية
3. التعلم الآلي : استخدام ML لتحسين التحليل
4. التوسع اللغوي : إضافة دعم للغات جديدة ((TypeScript, SCSS

10.3 القيمة الأكاديمية

يوفر هذا النظام:

- نموذجاً بحثياً : لدراسة تكامل اللغات البرمجية
- إطاراً تجريبياً : لتطوير وتقييم التقنيات الجديدة
- أداة تعليمية : لفهم تحليل وتصميم اللغات
- أساساً صناعياً : لتطوير أدوات تطوير البرمجيات

يمثل نظام AST المقدم إنجازاً مهماً في مجال معالجة اللغات البرمجية، حيث يوفر حلاً متكاملاً لتحليل لغات تطوير الويب الحديثة. التصميم المعياري والتكامل الفعال يجعلان النظام مناسباً للأغراض الأكاديمية والصناعية على حد سواء.

III. Visitor Pattern

تم اعتماد نمط التصميم **Visitor** لفصل منطق بناء الشجرة عن القواعد النحوية نفسها.
كل Visitor مسؤول عن:

- زيارة عقد شجرة التحليل الناتجة منANTLR
- إنشاء عقد AST مناسبة
- ربط العقد ببعضها بطريقة هرمية

هذا الأسلوب يسهّل:

- توسعة المشروع لاحقًا
- إضافة مراحل تحليل جديدة
- فصل المسؤوليات داخل الكود

- CSSASTVisitor

يُستخدم CSSASTVisitor لمعالجة قواعد CSS المدمجة مع HTML و Jinja2.

آلية العمل:

- عند زيارة كل قاعدة CSS:
  - يتم إنشاء عقدة AST مناسبة (مثل Ruleset، Selector، Declaration)
  - يتم تخزين رقم السطر والمحتوى النصي
- يتم التمييز بين:
  - القواعد الصحيحة (Good)
  - القواعد الخاطئة أو غير المعروفة (Bad / Unknown)

أمثلة على ما يعالجه:

- @charset, @import, @media
- Selectors (ID, Type, Universal)
- Expressions مثل `calc()`
- Keyframes و Media Queries

- FlaskASTVisitor

يُستخدم FlaskASTVisitor لمعالجة شيفرة Python الخاصة بتطبيقات Flask.

آلية العمل:

- يبدأ من العقدة الجذرية Program
- يتم تحليل:
  - الإسناد (Assignment)
  - الدوال (Function Definitions)
  - الجمل الشرطية (if / else)

- الحلقات(for / while)
- العبارات المنطقية والتعبيرات

كل بنية برمجية يتم تمثيلها بعقدة AST مستقلة:

- IfNode
- WhileNode
- ForNode
- FunctionDefNode
- ReturnNode

الهدف:

تحويل الشيفرة Python إلى تمثيل شجري يعكس منطق التنفيذ وليس فقط ترتيب النص

- HtmlASTVisitor

Visitorالـ HTML لا يعتمد مباشرة علىANTLR Visitor ، بل يعمل على AST مبنية مسبقًا لعناصر.HTML
دوره الأساسي هو المرور على عقد HTML المختلفة مثل:

- العناصر(Tags)
- السمات(Attributes)
- المحتوى النصي
- التعليقات
- Scriptو Style tags

يقوم هذا الزائر بزيارة العناصر بشكل هرمي، مما يضمن الحفاظ على البنية الصحيحة للـDOM ، كما يسمح بدمج CSS و Jinja داخل HTML بشكل منظم.

- Jinja2ASTVisitor

Visitorالخاص بـ Jinja2 هو الجزء الأهم من ناحية التحليل الدلالي.
يقوم هذا الزائر بما يلي:

- زيارة عُقد الإسناد والتعابير
- تتبع المتغيرات المستخدمة داخل القوالب
- التحقق من تعريف المتغيرات قبل استخدامها
- إدارة النطاقات (Scopes) مثل if وwhile

عند زيارة عقدة Assignment يتم إدخال المتغير إلى جدول الرموز،
وعند زيارة عقدة Variable يتم التحقق من وجودها في جدول الرموز، وفي حال عدم وجودها يتم الإبلاغ عن خطأ.

كما يتم إنشاء نطاق جديد عند الدخول إلى بنية تحكم، والخروج منه عند انتهاء البنية، مما يضمن صحة النطاقات.

العلاقة بين AST وVisitor

- الـ Parser ينشئ Parse Tree
- الـ Visitor يحوّل Parse Tree إلىAST
- AST تمثل البنية المجردة للبرنامج
- Visitorآخر) مثل (Jinja Visitor يستخدم AST لبناء Symbol Table والتحقق الدلالي

بهذا الشكل، يكون المشروع قد طبّق المراحل الأساسية لأي Compiler أو Template Engine حديث.

IV. Symbol Table

1. مقدمة

في هذا المشروع تم تنفيذ جدول رموز ((Symbol Tableللغة تعتمد على
Java. باستخدام لغة HTML + Jinja
الهدف من جدول الرموز هو تتبع المتغيرات، أنواعها، ونطاقاتها أثناء تحليل البرنامج.

2. بنية جدول الرموز

تم إنشاء بنية خاصة لإدارة جدول الرموز باستخدام الصنف:

SymbolTableManager

يقوم هذا الصنف بالمهام التالية:

- إنشاء Scope جديد
- الخروج من Scope
- إضافة رمز جديد
- طباعة جدول الرموز

كل Scope يحتوي على مجموعة من الرموز الخاصة به.

3. محتويات الرمز (Symbol)

كل رمز في جدول الرموز يحتوي على المعلومات التالية:

- اسم الرمز (Name)
- نوع الرمز (Kind)مثل variable
- نوع البيانات (Declared Type)مثل int
- رقم السطر الذي تم تعريف الرمز فيه

4. إدارة النطاقات (Scopes)

- عند بداية البرنامج يتم إنشاء نطاق عام.global
- عند الدخول إلى بنية تحكم مثل if يتم إنشاء نطاق جديد.
- عند الخروج من البنية يتم إغلاق النطاق الحالي والعودة إلى النطاق السابق.

هذا يضمن أن كل متغير يتم تعريفه داخل نطاقه الصحيح.

5. إضافة الرموز إلى الجدول

عند وجود عملية إسناد (Assignment)يتم إدخال المتغير إلى جدول الرموز.

يتم التحقق من النطاق الحالي وإضافة الرمز إليه.

في حال عدم تحديد نوع المتغير بشكل صريح يتم تعيين النوع unknown.

6. طباعة جدول الرموز

تم تنفيذ تابع خاص لطباعة جدول الرموز بشكل منظم ومقروء.

الطباعة تعرض:

- اسم النطاق
- قائمة المتغيرات الموجودة بداخله
- نوع كل متغير ورقم السطر

مثال على ناتج الطباعة:

```
SCOPE: global
---------------------------
x  variable  int  line 1
z  variable  int  line 3
```

7. الخلاصة

تم بناء جدول رموز متكامل يدعم تعدد النطاقاتScopes و يسمح بتتبع المتغيرات أثناء تنفيذ البرنامج و يشكل أساساً للتحقق الدلالي في أي مترجم أو Template Engine

رابط GitHub :
https://github.com/nader22rasheed/Compiler-