Otomatisasi Penerjemahan Kode Python ke Notasi Algoritmik ITB

Menggunakan Pendekatan Hibrida Berbasis Divide and Conquer dan Regex

Faqih Muhammad Syuhada - 13523057
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: faqihmsy@gmail.com , 13523057@std.stei.itb.ac.id

Abstract*—Dalam dunia akademik, seringkali dibutuhkan konversi dari kode sumber fungsional seperti Python ke dalam format notasi algoritmik standar untuk keperluan dokumentasi, analisis, dan pembelajaran. Proses konversi manual yang ada saat ini sangat memakan waktu dan berulang (redundant). Makalah ini menyajikan perancangan sebuah penerjemah otomatis untuk mengatasi masalah tersebut. Penerjemah ini mengimplementasikan pendekatan hibrida yang mengkombinasikan strategi Divide and Conquer untuk membedah struktur program menjadi blok-blok fungsi dan global, dengan Pencocokan String berbasis Regex untuk menerjemahkan sintaks pada setiap baris kode. Solusi ini diwujudkan dalam sebuah aplikasi web interaktif berbasis Python Flask, yang memungkinkan pengguna memasukkan kode Python dan mendapatkan hasil terjemahan secara langsung. Hasil utamanya adalah sebuah translator fungsional yang mampu menangani berbagai struktur kontrol, termasuk repeat-until, serta melakukan inferensi tipe data sederhana untuk menghasilkan KAMUS LOKAL yang informatif. Kontribusi dari makalah ini tidak hanya terletak pada perangkat lunak yang dihasilkan, tetapi juga pada analisis mendalam mengenai batasan-batasan fundamental dari pendekatan berbasis Regex dalam tugas-tugas parsing bahasa.

Keywords—regular expression; divide; conquer; python; notasi algoritma;

I. Pendahuluan

A. Latar Belakang

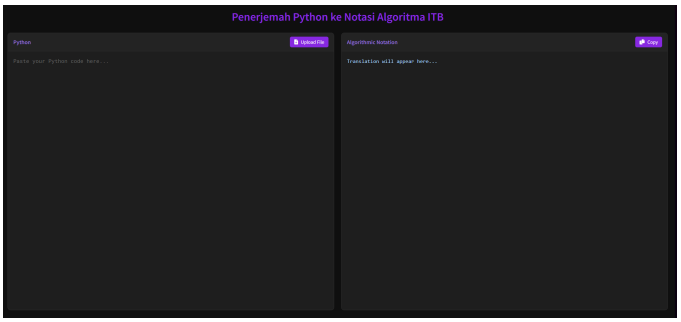
Dalam proses pembelajaran di Program Studi Teknik Informatika, Notasi Algoritmik memegang peranan penting sebagai salah satu standar untuk mendokumentasikan rancangan solusi. Pada beberapa mata kuliah, mahasiswa disarankan untuk merepresentasikan solusi pemrogramannya ke dalam format notasi algoritmik untuk keperluan laporan, analisis, dan penyajian ide yang lebih formal dan terstruktur. Namun, proses konversi dari kode sumber fungsional seperti Python ke notasi algoritmik secara manual merupakan kegiatan yang berulang dan memakan waktu, sehingga berpotensi menghambat efisiensi belajar.

Selain untuk kebutuhan akademik, sebuah penerjemah otomatis juga memiliki manfaat yang lebih luas. Bagi programmer pemula, alat ini dapat berfungsi sebagai media pembelajaran interaktif untuk memahami bagaimana sebuah logika pemrograman direpresentasikan secara formal. Bagi sebuah tim pengembang perangkat lunak, ia dapat membantu dalam standarisasi pembuatan dokumentasi teknis secara cepat dan konsisten. Oleh karena itu, pengembangan sebuah penerjemah otomatis menjadi sebuah solusi yang relevan untuk meningkatkan efisiensi belajar dan bekerja.

B. Solusi yang Diusulkan

Untuk mengatasi masalah tersebut, makalah ini menyajikan perancangan dan implementasi sebuah penerjemah otomatis dari kode Python ke Notasi Algoritmik ITB. Mengingat sifat tugas ini yang menuntut eksplorasi strategi algoritma, solusi yang diusulkan tidak menggunakan satu pendekatan tunggal. Sebaliknya, translator ini dibangun di atas arsitektur hibrida yang mengkombinasikan dua strategi utama yang tercantum dalam kurikulum IF2211 .

Strategi Pencocokan String berbasis Regex dipilih sebagai alat utama untuk melakukan analisis sintaksis pada level baris per baris. Regex sangat cocok untuk mengidentifikasi dan mengekstrak komponen dari sintaks Python yang terstruktur. Namun, menyadari keterbatasan fundamental Regex dalam memahami struktur bersarang dan konteks, maka arsitektur ini dipadukan dengan strategi Divide and Conquer. Pendekatan Divide and Conquer diimplementasikan untuk membedah keseluruhan kode sumber menjadi sub-masalah yang lebih kecil dan dapat dikelola, yaitu "blok-blok fungsi" dan "statement-statement global", yang kemudian diconquere secara terpisah sebelum hasilnya digabungkan.



Gambar 1. Tampilan Antarmuka Utama Aplikasi Web Penerjemah Python ke Notasi Algoritmik

C. Ruang Lingkup Makalah

Makalah ini akan fokus pada perancangan arsitektur translator, implementasi dari strategi-strategi algoritma yang dipilih, serta analisis fungsional dari program yang dihasilkan. Analisis akan mencakup keberhasilan dan kegagalan translator dalam menangani berbagai test case yang dibuat sesuai dengan materi acuan.

Pembahasan akan dibatasi pada logika dan strategi algoritma yang digunakan dalam backend penerjemahan. Makalah ini tidak akan membahas secara mendalam mengenai detail implementasi teknologi front-end atau kerangka kerja web Flask yang digunakan untuk menampilkan antarmuka. Kode sumber lengkap untuk proyek ini tersedia pada tautan repositori yang dilampirkan.

Penting untuk ditekankan bahwa istilah "Notasi Algoritmik" yang digunakan sebagai bahasa target dalam makalah ini merujuk secara spesifik pada standar dan kaidah penulisan yang digunakan dalam materi perkuliahan IF1210 Dasar Pemrograman. Acuan utama diambil dari berbagai contoh pseudocode yang mendefinisikan struktur seperti if-then-else, repeat-until, function, dan procedure. Dengan membatasi ruang lingkup pada standar ini, tujuan dan evaluasi dari translator yang dibangun menjadi jelas dan terukur.

II. Dasar Teori

A. Strategi Brute Force

Strategi Brute Force adalah pendekatan yang paling mendasar dan lugas dalam perancangan algoritma. Ia bekerja dengan cara mencoba semua kemungkinan solusi atau jalur secara sistematis untuk menemukan solusi yang diinginkan. Menurut Munir, algoritma Brute Force umumnya lebih mudah diimplementasikan namun seringkali bukanlah yang paling efisien dalam hal kompleksitas waktu, terutama untuk masalah dengan ruang pencarian yang besar [1].

Contoh klasik dari algoritma Brute Force adalah Sequential Search, di mana untuk mencari sebuah elemen dalam sebuah array, setiap elemen dari awal hingga akhir diperiksa satu per satu tanpa adanya heuristik atau lompatan cerdas.

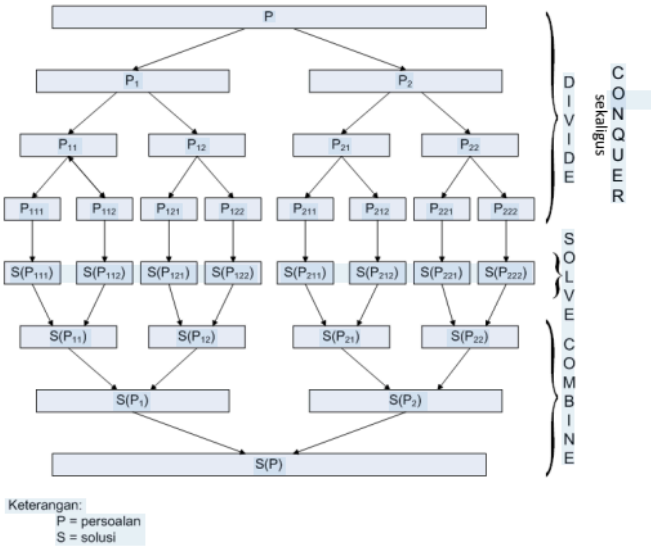
Dalam konteks translator yang dibangun dalam makalah ini, prinsip Brute Force terlihat pada cara kerja fungsi terjemahkan_satu_baris. Ketika menerima satu baris kode, fungsi tersebut tidak memiliki cara untuk "mengetahui" jenis sintaks apa yang terkandung di dalamnya. Oleh karena itu, ia melakukan pencarian secara brute force dengan mencoba mencocokkan baris tersebut dengan setiap pola Regex yang tersimpan di dalam dictionary patterns, satu demi satu secara berurutan, hingga menemukan pola pertama yang cocok.

B. Strategi Divide and Conquer (D&C)

Divide and Conquer adalah paradigma perancangan algoritma yang kuat untuk menyelesaikan masalah-masalah kompleks dengan cara memecahnya menjadi bagian-bagian yang lebih kecil. Filosofi ini, yang awalnya merupakan strategi militer divide ut imperes, menjadi fundamental dalam ilmu komputer. Proses D&C secara formal terdiri dari tiga langkah utama yang dijalankan secara rekursif

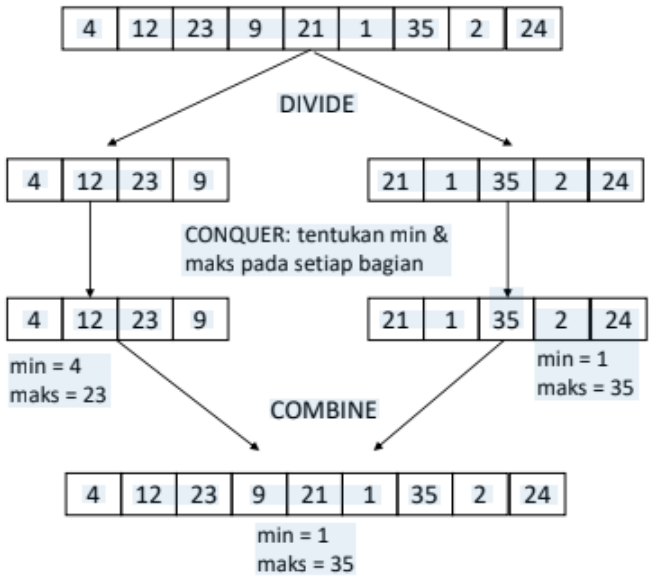
1. Divide Pada tahap ini, masalah utama dibagi menjadi beberapa sub-masalah P1, P2, ..., Pr. Setiap sub-masalah memiliki karakteristik yang sama persis dengan masalah semula, tetapi dengan ukuran yang lebih kecil (n1, n2, ..., nr). Dalam translator ini, tahap divide terjadi saat kode sumber dipisahkan menjadi blok definisi fungsi dan blok statement global.
2. Conquer Setiap sub-masalah kemudian diselesaikan. Jika ukuran sub-masalah sudah cukup kecil atau kasus basis, ia diselesaikan secara langsung. Jika masih besar, ia diselesaikan secara rekursif dengan menerapkan kembali strategi Divide and Conquer. Pada translator ini, tahap conquer diimplementasikan dengan memanggil fungsi helper yang berbeda untuk memproses blok fungsi dan statement global.
3. Combine Setelah semua sub-masalah selesai, solusi dari masing-masing sub-masalah tersebut digabungkan untuk membentuk solusi dari masalah awal. Dalam translator ini, tahap combine adalah saat hasil terjemahan dari KAMUS global, ALGORITMA global, dan REALISASI FUNGSI/PROSEDUR disusun menjadi satu dokumen notasi algoritmik yang utuh.

Skema kerja rekursif dari Divide and Conquer ini seringkali diilustrasikan dengan diagram pohon, seperti yang ditunjukkan pada Gambar 2, di mana sebuah masalah P dipecah terus-menerus hingga mencapai kasus basis yang solusinya dapat ditemukan, lalu solusi-solusi tersebut digabungkan kembali ke atas.



Gambar 2. Skema Umum Algoritma Divide and Conquer [1]

Sebagai contoh simulasi, mari kita tinjau persoalan MinMaks, yaitu mencari nilai minimum dan maksimum dalam sebuah larik secara bersamaan. Dengan pendekatan D&C, larik tersebut dibagi dua secara terus-menerus hingga hanya tersisa satu atau dua elemen pada tahap Divide. Nilai min dan maks dari sub-larik kecil ini kemudian ditemukan secara langsung pada tahap Conquer. Selanjutnya, hasil dari setiap bagian digabungkan kembali dengan membandingkan nilai min dari bagian kiri dengan min dari bagian kanan, dan maks dari bagian kiri dengan maks dari bagian kanan pada tahap Combine. Proses ini diilustrasikan pada Gambar 3.



Gambar 3. Simulasi Penyelesaian Masalah MinMaks dengan D&C [1]

Dalam translator ini, strategi D&C menjadi arsitektur utama. Keseluruhan kode Python dibagi menjadi statement global dan blok fungsi atau sub-problems. Setiap sub-problem ini lalu diconquere oleh fungsi helper yang berbeda, sebelum hasilnya digabungkan kembali dalam urutan yang benar yaitu (KAMUS, ALGORITMA, REALISASI.

C. Strategi Greedy

Algoritma Greedy adalah strategi yang membangun solusi langkah per langkah dengan membuat pilihan yang tampak paling optimal pada saat itu juga. Menurut Munir, pada setiap langkahnya, algoritma Greedy memilih pilihan yang "paling rakus" dengan harapan bahwa rangkaian pilihan optimal lokal ini akan menghasilkan solusi optimal global [1]. Namun, tidak semua masalah dapat diselesaikan dengan optimal oleh strategi ini.

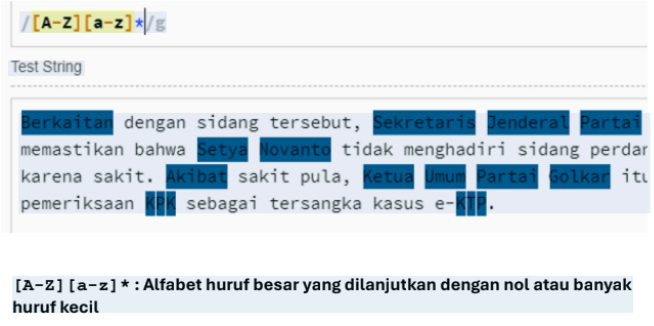
Salah satu contoh klasik adalah Fractional Knapsack Problem, di mana seorang pencuri harus memilih barang untuk dimasukkan ke dalam karung dengan kapasitas terbatas. Solusi greedy-nya adalah dengan selalu memilih barang yang memiliki rasio value/weight tertinggi terlebih dahulu hingga karung penuh.

Dalam translator yang dibangun, pendekatan heuristik untuk membedakan function dari procedure menerapkan prinsip yang mirip dengan Greedy. Pada "Pass 1", saat skrip memindai kode, begitu ia menemukan sebuah pernyataan return yang mengembalikan nilai, ia secara "rakus" dan langsung membuat keputusan bahwa def yang sedang aktif saat itu adalah sebuah function. Ia tidak menunggu hingga akhir analisis blok atau mempertimbangkan semua jalur eksekusi, melainkan mengambil keputusan berdasarkan informasi lokal terbaik yang tersedia pada saat itu.

D. Pencocokan String dengan Regular Expression (Regex)

Pencocokan String adalah persoalan fundamental untuk menemukan keberadaan sebuah pola dalam sebuah teks. Meskipun ada banyak algoritma untuk pencocokan string literal (seperti Boyer-Moore dan KMP), pendekatan yang digunakan dalam translator ini adalah Pencocokan Pola dengan Regular Expression (Regex).

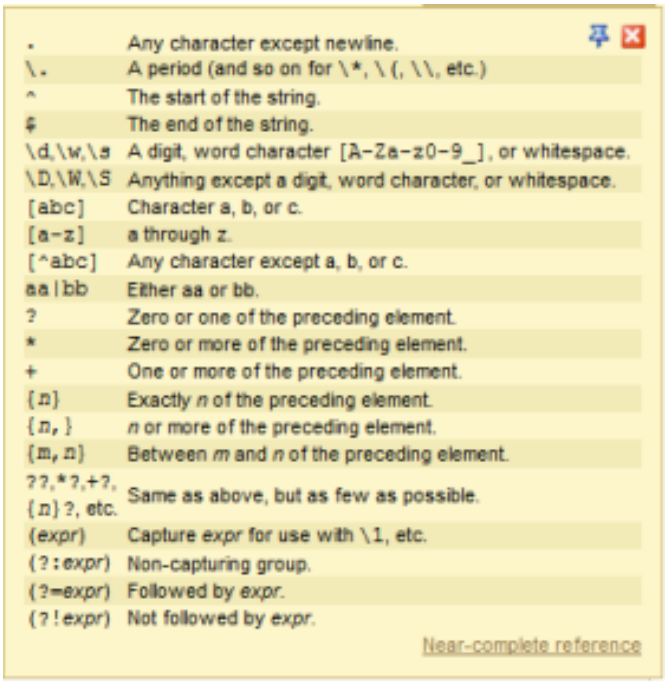
Regex adalah sebuah bahasa sekuensial yang sangat kuat untuk mendefinisikan pola teks secara fleksibel, melampaui pencarian kata kunci biasa. Sebagai contoh, pola /[A-Z][a-z]*/ dapat menemukan semua kata yang diawali dengan huruf kapital, sebuah tugas yang mustahil dilakukan oleh pencocokan string literal biasa. Fleksibilitas ini membuat Regex menjadi alat yang ideal untuk tugas analisis sintaksis.



Gambar 4. Simulasi Penggunaan Regex untuk Menemukan Pola Kata [1]

Pada translator ini, Regex adalah "pekerja keras" utama. Setiap elemen sintaksis Python mulai dari def, if, for, while, hingga assignment didefinisikan sebagai pola Regex yang unik di dalam dictionary patterns. Penggunaan capturing groups (()) dalam pola-pola ini sangatlah krusial. Ia memungkinkan skrip untuk tidak hanya mengidentifikasi sebuah baris sebagai if statement, tetapi juga untuk mengekstrak dan mengisolasi bagian kondisinya, yang kemudian dapat diproses dan diterjemahkan ke dalam format notasi algoritmik yang benar.

Pola teks yang fleksibel ini memiliki notasi dasar yang biasa di sebut notasi dasar Regex seperti yang ditunjukkan seperti table 1.



.	Any character except newline.
\.	A period (and so on for *, \(, \\, etc.)
^	The start of the string.
$	The end of the string.
\d,\w,\s	A digit, word character [A-Za-z0-9_], or whitespace.
\D,\W,\S	Anything except a digit, word character, or whitespace.
[abc]	Character a, b, or c.
[a-z]	a through z.
[^abc]	Any character except a, b, or c.
aa|bb	Either aa or bb.
?	Zero or one of the preceding element.
*	Zero or more of the preceding element.
+	One or more of the preceding element.
{n}	Exactly *n* of the preceding element.
{n,}	*n* or more of the preceding element.
{m,n}	Between *m* and *n* of the preceding element.
??,*?,+?, {n}?, etc.	Same as above, but as few as possible.
(*expr*)	Capture *expr* for use with \1, etc.
(?:*expr*)	Non-capturing group.
(?=*expr*)	Followed by *expr*.
(?!*expr*)	Not followed by *expr*.

Near-complete reference

Tabel 1. Contoh Notasi dan Metacharacter pada Regular Expression [1]

E. Bahasa Pemrograman Python (Bahasa Sumber)

Python adalah bahasa pemrograman interpretatif tingkat tinggi yang dikenal karena sintaksnya yang bersih, mudah dibaca, dan dinamis[2]. Tiga karakteristik utama Python yang relevan dengan perancangan translator ini adalah,

1. Tipe Data Dinamis (Dynamically Typed)
   Berbeda dengan bahasa seperti C atau Pascal, Python tidak mengharuskan programmer untuk mendeklarasikan tipe data sebuah variabel secara eksplisit. Tipe sebuah variabel ditentukan secara otomatis saat program dijalankan (runtime) berdasarkan nilai yang disimpannya. Sebagai contoh, x = 10 akan membuat x menjadi integer, dan di baris selanjutnya bisa diubah menjadi x = "hello" yang mengubah tipenya menjadi string. Karakteristik ini memberikan fleksibilitas bagi programmer, namun menjadi tantangan utama bagi translator yang harus menyimpulkan tipe data untuk notasi algoritmik yang bersifat statis.
2. Struktur Blok Berbasis Indentasi
   Python tidak menggunakan kurung kurawal {} atau kata kunci seperti begin-end untuk menandai sebuah blok kode. Sebaliknya, Python menggunakan spasi atau tab di awal baris atau identasi untuk mendefinisikan blok-blok di dalam struktur seperti if, for, dan def. Aturan yang kaku ini menyederhanakan proses identifikasi struktur blok bagi translator.
3. Sintaks yang Ekspresif
   Python memiliki banyak sintaks tingkat tinggi seperti list comprehension dan lambda function yang memungkinkan penulisan kode yang ringkas. Untuk ruang lingkup makalah ini, translator akan berfokus pada sintaksis dasar yang memiliki padanan langsung dalam notasi algoritmik, seperti yang diajarkan dalam mata kuliah IF1210 Dasar Pemrograman.

F. Notasi Algoritmik ITB (Bahasa Target)

Notasi Algoritmik yang digunakan sebagai acuan dalam makalah ini adalah standar yang diajarkan dalam kurikulum Program Studi Teknik Informatika ITB, seperti yang terdokumentasi dalam berbagai materi kuliah. Notasi ini bersifat seperti pseudocode terstruktur yang dirancang untuk menjadi jembatan antara ide algoritmik dan implementasi kode, dengan mengabaikan detail sintaksis bahasa pemrograman tertentu.

Struktur sebuah program dalam notasi ini umumnya terdiri dari tiga bagian utama,

- Kepala Program (Header)
  Berisi judul program dan spesifikasi singkat mengenai apa yang dilakukan oleh algoritma.
- KAMUS
  Bagian ini digunakan untuk mendeklarasikan semua "peubah" yang digunakan, seperti variabel, konstanta, atau tipe data bentukan, lengkap dengan tipe datanya secara eksplisit
- ALGORITMA
  Bagian ini berisi langkah-langkah atau instruksi dari algoritma itu sendiri.

Notasi ini menggunakan kata kunci yang jelas untuk setiap struktur kontrol, seperti if...then...else, while...do, repeat...until, for...traversal, serta membedakan sub-program menjadi function dan procedure.

III. Implementasi dan Pengujian

A. Arsitektur Perangkat Lunak

Perangkat lunak ini dirancang sebagai aplikasi web interaktif untuk memberikan kemudahan akses bagi pengguna. Arsitektur perangkat lunak terdiri dari dua komponen utama: front-end sebagai antarmuka pengguna dan back-end sebagai mesin penerjemah.

- Front-end
  Dibangun menggunakan HTML, CSS, dan JavaScript standar, menyediakan area teks bagi pengguna untuk memasukkan kode Python dan sebuah tombol untuk memicu proses translasi. Komunikasi dengan back-end dilakukan secara asinkron melalui API.

- Back-end
  Dibangun menggunakan kerangka kerja Flask pada Python. Back-end menyediakan satu endpoint API (/translate) yang menerima kode Python dari front-end, memprosesnya menggunakan logika translator, dan mengembalikan hasilnya dalam format JSON.

Inti dari arsitektur translator ini adalah penerapan strategi Divide and Conquer yang dilakukan dalam beberapa langkah pemrosesan untuk memastikan akurasi struktural, terutama dalam memisahkan KAMUS dan ALGORITMA. Alur kerja utamanya adalah sebagai berikut,

1. Analisis Awal
   Seluruh kode sumber dipindai terlebih dahulu untuk melakukan analisis heuristik. Tujuannya adalah untuk mengidentifikasi semua definisi fungsi dan menentukan apakah fungsi tersebut berperan sebagai function atau sebagai procedure. Informasi ini disimpan untuk digunakan pada tahap selanjutnya.
2. Divide
   Program kemudian memindai kode sumber untuk kedua kalinya. Pada tahap ini, ia secara cerdas memisahkan keseluruhan kode menjadi dua kategori sub-masalah yang berbeda yaitu Kumpulan statement yang berada di lingkup global, dan Kumpulan blok-blok kode yang masing-masing merupakan satu definisi fungsi yang utuh.
3. Conquer
   Setiap jenis sub-masalah diselesaikan oleh logika yang berbeda.
   - Statement global diterjemahkan secara langsung untuk membentuk KAMUS dan ALGORITMA utama.
   - Setiap blok fungsi diproses secara terpisah dengan metode dua-pindaian yaitu pindaian pertama untuk mengekstrak semua deklarasi variabel menjadi KAMUS LOKAL, dan pindaian kedua untuk menerjemahkan semua baris aksi menjadi isi dari ALGORITMA.
4. Combine
   Terakhir, semua hasil terjemahan dari setiap bagian digabungkan kembali dalam urutan yang benar sesuai standar notasi yaitu PROGRAM Utama, KAMUS global, ALGORITMA utama, dan diakhiri dengan blok { REALISASI FUNGSI/PROSEDUR } yang berisi hasil terjemahan dari setiap blok fungsi.

B. Implementasi Divide and Conquer

Strategi Divide and Conquer adalah tulang punggung dari arsitektur translator. Implementasinya terlihat pada fungsi utama terjemahkan_python_ke_notasi_final, di mana sebuah loop memindai seluruh baris kode untuk memisahkannya menjadi dua list yang berbeda statement_global dan definisi_fungsi.

```python
while i < len(lines):
    line = lines[i]
    indent = len(line) - len(line.lstrip(' '))
    stripped = line.strip()

    if patterns['def'].match(stripped):
        if current_func_lines:
            fungsi_blocks[def_match.group(1)] = current_func_lines

        def_match = patterns['def'].match(stripped)
        current_func_name = def_match.group(1)
        base_indent = indent
        current_func_lines = [line]

        j = i + 1
        while j < len(lines):
            if lines[j].strip() == "" or (len(lines[j])
                - len(lines[j].lstrip(' '))) > base_indent:
                current_func_lines.append(lines[j])
                j += 1
            else:
                break
        fungsi_blocks[current_func_name] = current_func_lines
        i = j -1
    else:
        program_utama_lines.append(line)
    i += 1
```

Gambar 6. Kode Implementasi Tahap Pembagian (Divide)

Setelah pemisahan ini, setiap blok fungsi di dalam list definisi_fungsi diconquere secara rekursif oleh pemanggilan fungsi proses_blok_fungsi. Fungsi ini sendiri menerapkan lagi prinsip serupa dengan melakukan dua pass terpisah untuk KAMUS dan ALGORITMA.

C. Implementasi Pencocokan String dengan Regex

Regex merupakan "alat kerja" utama yang digunakan pada level mikro untuk menganalisis dan menerjemahkan setiap baris kode. Sekumpulan pola Regex yang telah dikompilasi sebelumnya disimpan dalam sebuah dictionary bernama patterns. Setiap pola dirancang untuk mengidentifikasi satu jenis sintaks Python secara spesifik.

```python
patterns = {
            'comment': re.compile(r'^\s*#\s*(.*)'),
            'def': re.compile(r'^\s*def\s+(\w+)\s*\((.*?)\):'),
            'if': re.compile(r'^\s*if\s+(.*):'),
            'elif': re.compile(r'^\s*elif\s+(.*):'),
            'else': re.compile(r'^\s*else:'),
            'while': re.compile(r'^\s*while\s+(.*):'),
            'for_range': re.compile(r'^\s*for\s+(\w+)\s+in\s+range\((.*?)\):'),
            'return': re.compile(r'^\s*return(\s+.*)?'),
            'break': re.compile(r'^\s*break\s*'),
            'print': re.compile(r'^\s*print\((.*)\)'),
            'input': re.compile(r'^\s*(\w+)\s*=\s*.*?input\((.*)\)'),
            'assignment': re.compile(r'^\s*([a-zA-Z_][\w]*)\s*=\s*(.*)')
        }
```

Gambar 7. Kumpulan Pola Regex untuk Identifikasi Sintaks Python

Saat memproses sebuah baris, skrip menggunakan pendekatan Brute Force dengan mencoba setiap pola dalam patterns pada baris tersebut melalui serangkaian if/elif. Ketika sebuah pola cocok, kemampuan capturing group (()) pada Regex digunakan untuk mengekstrak informasi penting. Sebagai contoh, pada pola for_range yaitu r'for\s+(\w+)\s+in\s+range\((.*?)\):', grup pertama (\w+) akan

menangkap nama variabel iterator, dan grup kedua (.*?) akan menangkap argumen di dalam range(). Informasi yang diekstrak inilah yang kemudian digunakan untuk membangun string notasi algoritmik yang sesuai.

D. Implementasi Heuristik Greedy

Untuk membedakan antara function dan procedure sebuah informasi yang tidak eksplisit dalam sintaks def Python sebuah pendekatan heuristik berbasis strategi Greedy diterapkan. Pada for pertama, program akan memindai seluruh kode. Begitu menemukan sebuah pernyataan return yang mengembalikan nilai, ia akan langsung "dengan rakusnya" membuat keputusan bahwa blok def yang sedang aktif saat itu adalah sebuah function, tanpa perlu menunggu analisis keseluruhan blok selesai. Meskipun pendekatan ini tidak sempurna untuk semua kasus, ia memberikan solusi yang cepat dan "cukup baik" untuk sebagian besar kasus umum.

```
functions_info = {}
current_func = None
for line in lines:
    stripped_line = line.strip()
    if match := patterns['def'].match(stripped_line):
        current_func = match.group(1)
        functions_info[current_func] = {'has_return_value': False, 'return_type': '<tipe_hasil>'}
    elif match := patterns['return'].match(stripped_line):
        if current_func and match.group(1) and match.group(1).strip():
            functions_info[current_func]['has_return_value'] = True
            return_val = match.group(1).strip()
            functions_info[current_func]['return_type'] = infer_tipe_dari_nilai(return_val)
```

Gambar 8. Implementasi Heuristik Greedy untuk Deteksi Fungsi

E. Pengujian Struktur Percabangan (if-elif-else)

Pengujian ini bertujuan untuk memverifikasi kemampuan *translator* dalam menangani struktur percabangan multi-kasus yang umum.

Kode Python Input:

```
# Test Case untuk if-elif-else
skor = 88
if skor >= 85:
    print("A")
elif skor >= 75:
    print("B")
else:
    print("D")
```

Hasil Notasi Algoritmik yang Dihasilkan:

```
{ Test Case untuk if-elif-else }
skor ← 88
if skor ≥ 85 then
    output("A")
else if skor ≥ 75 then
    output("B")
else
    output("D")
```

Analisis Singkat, Hasil translasi menunjukkan bahwa skrip berhasil mengenali struktur if, elif, dan else serta mempertahankan hierarki indentasi dengan benar. Pengujian dinyatakan berhasil.

Gambar 9. Hasil Pengujian untuk Struktur if-elif-else

F. Pengujian Struktur repeat-until

Pengujian ini bertujuan untuk mengevaluasi kemampuan skrip dalam menangani logika state machine yang lebih kompleks untuk menerjemahkan pola while True...if-break.

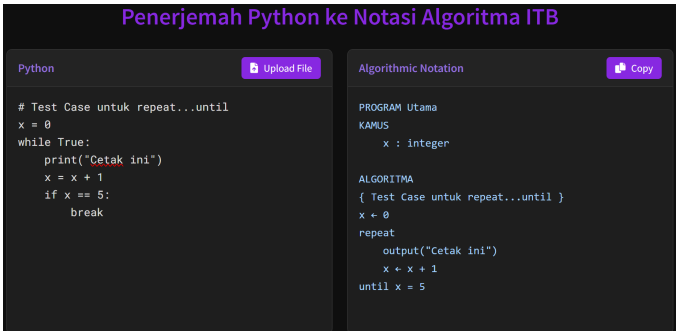
Kode Python Input:

```
# Test Case untuk repeat...until
x = 0
while True:
    print("Cetak ini")
    x = x + 1
    if x == 5:
        Break
```

Hasil Notasi Algoritmik yang Dihasilkan:

```
{ Test Case untuk repeat...until }
x ← 0
repeat
    output("Cetak ini")
    x ← x + 1
until x = 5
```

Analisis Singkat, Hasil translasi menunjukkan bahwa arsitektur yang dirancang mampu mengenali konteks multi-baris dari while True yang diakhiri dengan if-break dan mengubahnya menjadi notasi repeat-until yang benar. Pengujian dinyatakan berhasil.



Gambar 10. Hasil Pengujian untuk Struktur repeat-until

G. Pengujian Batasan Algoritma (iterate...stop)

Pengujian ini bertujuan untuk mendemonstrasikan batasan fundamental dari pendekatan berbasis Regex dalam memahami struktur semantik yang kompleks.

Kode Python Input:

```
# Test Case untuk iterate...stop
data = 0
while True:
    data = data + 10
    if data >= 50:
        break
    else:
        print(data)
```

Hasil Notasi Algoritmik yang Dihasilkan:

```
{ Test Case untuk iterate...stop }
data ← 0
iterate
    data ← data + 10
    stop data ≥ 50
        output(data)
```

Analisis Singkat, Hasil translasi menunjukkan kegagalan total. Skrip salah menafsirkan if-break sebagai if-then biasa dan mencetak klausa else secara harfiah. Pengujian dinyatakan gagal, sesuai dengan prediksi batasan arsitektur.

Gambar 11. Hasil Uji Kegagalan pada Struktur iterate-stop

IV. Analisis Dan Kesimpulan

H. Analisis Hasil Uji

Dari serangkaian pengujian yang dilakukan, terlihat bahwa translator yang dibangun mampu menangani sebagian besar konstruksi sintaksis dasar dan beberapa pola kompleks, namun menemui kegagalan pada kasus yang memerlukan pemahaman semantik mendalam.

Analisis Keberhasilan,
Keberhasilan translator dalam menangani struktur seperti if-elif-else, while-do, dan traversal menunjukkan bahwa pendekatan Brute Force dengan mencocokkan setiap baris terhadap serangkaian pola Regex yang terdefinisi dengan baik sudah cukup untuk sintaks yang jelas dan tidak ambigu.

Keberhasilan yang lebih signifikan adalah pada penanganan repeat...until. Ini membuktikan bahwa dengan menambahkan logika state machine sederhana, pendekatan Regex dapat "diperluas" kemampuannya untuk mengenali pola multi-baris yang spesifik. Arsitektur Divide and Conquer juga terbukti efektif dalam memisahkan lingkup global dan fungsi, yang memungkinkan pemisahan KAMUS dan ALGORITMA secara struktural.

Analisis Kegagalan
Kegagalan paling signifikan terjadi pada test case iterate...stop. Hasil terjemahannya menjadi tidak bermakna karena skrip salah menginterpretasikan hubungan antara if, break, dan else. Kegagalan ini menyoroti batasan teoretis fundamental dari pendekatan yang digunakan,

1. Regex sebagai Finite Automaton (FA)
   Mesin Regex pada dasarnya adalah sebuah Finite Automaton yang tidak memiliki memori tumpukan (stack). Ia unggul dalam mengenali pola-pola reguler dalam satu baris.
2. Kebutuhan akan Konteks (CFG)
   Struktur iterate...stop adalah sebuah tata bahasa bebas konteks (Context-Free Grammar). Untuk memahaminya, translator perlu mengerti bahwa else tersebut terikat secara semantik pada if yang dua baris sebelumnya, yang mana if tersebut juga mengandung break. Pemahaman hierarki dan hubungan antar klausa ini adalah domain dari Pushdown Automaton (PDA) yang menggunakan stack, bukan Regex.

Kegagalan ini membuktikan bahwa secanggih apa pun "tambalan" atau state machine sederhana yang kita buat, pendekatan Regex akan selalu kesulitan dalam memahami makna semantik dari kode.

I. Kesimpulan

Berdasarkan perancangan, implementasi, dan pengujian yang telah dilakukan, dapat ditarik beberapa kesimpulan,

1. Sebuah translator fungsional dari kode Python ke Notasi Algoritmik ITB telah berhasil dibangun menggunakan pendekatan hibrida Divide and Conquer dan Pencocokan String berbasis Regex. Program ini mampu menangani sebagian besar struktur kontrol dasar dan beberapa kasus kompleks seperti repeat...until.
2. Telah terbukti bahwa pendekatan "hanya Regex", bahkan dengan arsitektur block-aware dan multi-pass, memiliki batasan teoritis yang tidak dapat diatasi dalam mem-parsing struktur bahasa pemrograman yang bersifat bebas konteks (context-free). Kegagalan dalam menerjemahkan konstruksi iterate...stop menjadi bukti nyata dari batasan ini.
3. Kontribusi utama dari makalah ini ada dua. Pertama, terciptanya sebuah perangkat lunak aplikasi web yang dapat menjadi alat bantu praktis untuk mahasiswa dalam proses dokumentasi dan pembelajaran algoritma. Kedua, yang lebih penting secara akademis, adalah demonstrasi dan analisis komparatif antara kemampuan Regex dan kebutuhan akan tata

bahasa formal (CFG) dalam menyelesaikan masalah parsing, yang memberikan wawasan praktis tentang teori bahasa dan automata.

Link Video Youtube

https://youtu.be/Btwk0x8gEDM?si=_OE3lrnMGxeckWwd

Link Kode Program

Link Repository :
https://github.com/FaqihMSY/Python-to-ITB-Algorithm-Notation-Translator

Link Projek:
https://python-to-itb-algorithm-notation-tr.vercel.app/

Ucapan Terima Kasih

Bismillahirrahmanirrahim.

Segala puji dan syukur penulis panjatkan kehadirat Allah SWT, Tuhan semesta alam, yang atas rahmat, hidayah, dan karunia-Nya telah memberikan kekuatan dan kemudahan sehingga penulis dapat menyelesaikan makalah yang berjudul "Otomatisasi Penerjemahan Kode Python ke Notasi Algoritmik ITB Menggunakan Pendekatan Hibrida Berbasis Divide and Conquer dan Regex" dengan baik dan tepat waktu. Shalawat serta salam semoga senantiasa tercurah kepada junjungan kita, Nabi Muhammad SAW, beserta keluarga, sahabat, dan para pengikutnya hingga akhir zaman.

Penulis juga mengucapkan terima kasih yang tulus kepada keluarga tercinta yang telah memberikan dukungan moril, materil, dan doa yang tak henti-hentinya sehingga penulis dapat menyelesaikan Makalah IF2211 Strategi Algoritma Sem. II Tahun 2024/2025 ini.

Terima kasih yang mendalam juga penulis sampaikan kepada dosen-dosen pengampu mata kuliah IF2211, terutama kepada Dr. Nur Ulfa Maulidevi, S.T, M.Sc. dan Dr. Ir. Rinaldi, M.T., yang telah dengan sabar membimbing, memberikan ilmu, serta menyediakan sumber belajar yang sangat berharga untuk memahami keilmuan dalam makalah ini.

Penulis berharap bahwa makalah ini dapat membawa manfaat, menjadi referensi bagi para pembelajar yang tertarik dengan keilmuan terkait, dan menjadi amal jariyah yang diridhai Allah SWT. Penulis menyadari bahwa makalah ini masih jauh dari kesempurnaan, oleh karena itu kritik dan saran yang membangun sangat diharapkan.

References

[1] Munir, Rinaldi, "Bahan Kuliah IF2211 Strategi Algoritma". https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-_2021/Pencocokan-string-2021.pdf (diakses pada 24 Juni 2024)

[2] Python Software Foundation, "Python Language Reference, version 3.13.5". [Online]. https://docs.python.org/3/ (diakses pada 24 Juni 2024).

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 24 Juni 2025

Faqih Muhammad Syuhada dan 13523057