BAB III

METODE DAN PERANCANGAN SISTEM

3.1 Metodologi Penelitian

Metodologi yang digunakan dalam pembuatan *dependency parser* bahasa Indonesia ini adalah sebagai berikut

1. Telaah Literatur

   Dalam rangka memahami seluk beluk dari *dependency parsing* dan menjawab pertanyaan seperti “apa itu *dependency parsing*?”, lalu “bagaimana *dependency parser* dibuat?” dan pertanyaan lainnya, diperlukan suatu upaya mencari materi yang menjawab semua pertanyaan tersebut. Materi yang terkait meliputi jurnal-jurnal, artikel, dan buku yang membahas *dependency parsing* dan *neural network* secara komprehensif.

2. Perancangan

   Tahapan ini adalah visualisasi dan pembuatan suatu *sketch* untuk mengkomunikasikan berbagai aspek dan karakteristik dari *dependency parser* yang ingin dibuat. Pada tahapan ini dijelaskan arsitektur dari *neural netowork* yang digunakan, flowchart.

3. Implementasi

   Arsitektur dan desain dari tahapan perancangan akan diterjemahkan menjadi *code*, bersama dengan itu implementasi dari *dependency parser* secara umum adalah sebagai berikut

   - Membuat sistem transisi yang akan memproses kalimat menjadi *dependency tree*. Sistem transisi yang digunakan adalah *arc-standard system*.
   - Mendefinisikan *feature function*, yang memilih elemen apa saja yang dianggap penting untuk merepresentasikan sekian banyak kombinasi dari konfigurasi yang dihasilkan oleh sistem transisi. *Feature function* pada penelitian ini mengikuti Chen dan Manning (2014).
   - Mengkonfigurasikan model *neural network*. Arsitektur yang dibuat merupakan *feed-forward network* dengan ReLU sebagai *activation function*. Proses belajar model menggunakan varian dari *gradient decent learning* yaitu Adam.
   - Menggunakan ELMo *word representation* sebagai *input representation* dari *feature* yang telah dipilih oleh *feature function*.
   - Melatih model dengan *traning* data yang diambil dari Universal Dependencies (Marneffe, dkk., 2015).

4. Pengujian

   Setelah *dependency parser* selesai dibuat, perlu dilakukan ujicoba untuk mengetahui apakah program sudah sesuai dengan kebutuhan program yang ditentukan di awal. Ketika *dependency parser* diyakini telah bebas dari error,

langkah selanjutnya adalah melakukan evaluasi terhadap hasil *parsing*. Tingkat akurasi *dependency parser* akan diuji menggunakan *testing dataset* yang disediakan oleh Universal Dependencies. Untuk itu tolak ukur yang dipergunakan adalah UAS dan LAS dari suatu *dependency parser*.

3.2 Gambaran Umum

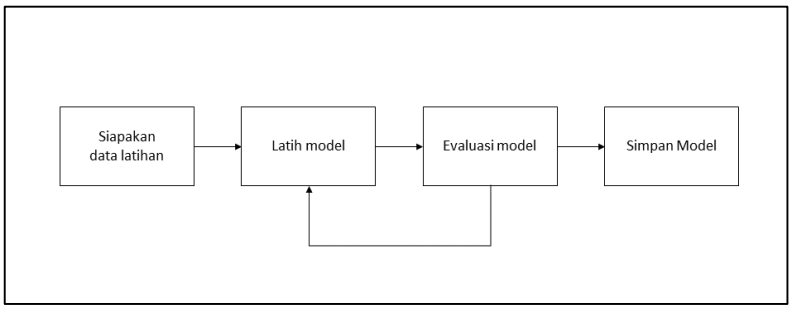
Proses pengembangan pengurai dependensi diawali dari fase pelatihan model FNN. Tahapan ini melibatkan persiapan data latihan. Data latihan berupa *Indonesian treebank* yang diperoleh dari UD. *Indonesian treebank* ini terdiri dari kalimat yang sudah disertai dengan pohon dependensi. Contoh data latihan dapat dilihat pada Gambar 3.1.

```
# sent_id = test-s14
# text = Kamila bertanya kenapa Eyang Tini menyuruhnya naik mobil?
1       Kamila  kamila  PROPN   X--     _       2       nsubj   _       MorphInd=^kamila<x>_X--$
2       bertanya        bertanya        VERB    VSA     Number=Sing|Voice=Act   0       root    _
MorphInd=^ber+tanya<v>_VSA$
3       kenapa  kenapa  ADV     W--     PronType=Int    6       advmod  _       MorphInd=^kenapa<w>_W--$
4       Eyang   eyang   PROPN   X--     _       6       nsubj   _       MorphInd=^eyang<x>_X--$
5       Tini    tini    PROPN   VSA     Number=Sing|Voice=Act   4       flat    _       MorphInd=^tin<n>+i_VSA$
6       menyuruhnya     mensuruh        VERB    VSA+PS3 Number=Sing|Number[psor]=Sing|Person[psor]=3|Voice=Act  2
ccomp   _       MorphInd=^meN+suruh<v>_VSA+dia<p>_PS3$
7       naik    naik    VERB    VSA     Number=Sing|Voice=Act   6       xcomp   _       MorphInd=^naik<v>_VSA$
8       mobil   mobil   NOUN    NSD     Number=Sing     7       obj     _       SpaceAfter=No|MorphInd=^mobil<n>_NSD$
9       ?       ?       PUNCT   Z--     _       2       punct   _       MorphInd=^?<z>_Z--$
```

Gambar 3.1 Data Latihan dari UD (McDonald, dkk., 2018)

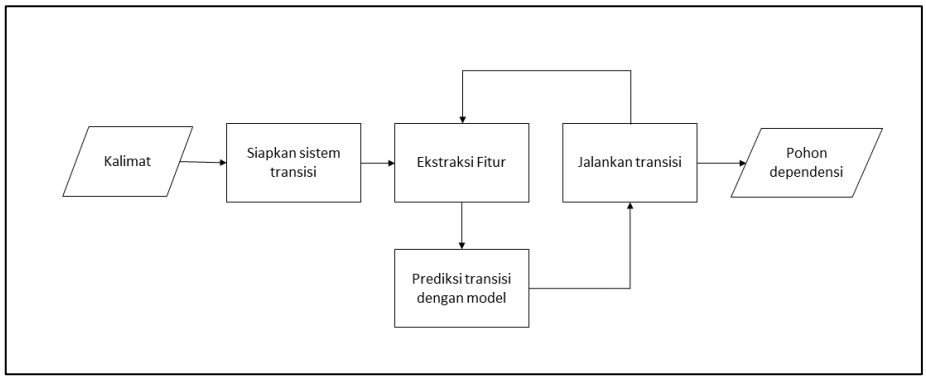
Suatu kalimat latihan disusun dalam format CoNLL. Setiap baris adalah kata-kata yang dipecah dari kalimat. Setiap kolumnya mendeskripsikan banyak hal seperti misalnya kata, kelas kata, dan hubungan dependensi. Format CoNLL pada Gambar 3.1 perlu diubah ke urutan konfigurasi dan transisi mengingat metode *parsing* yang digunakan adalah *transition based parsing*. Setelah data latihan siap, proses selanjutnya adalah

melatih model FNN dengan *gradient decent learning*. Proses pelatihan ini diikuti dengan evaluasi di setiap *epoch*. Model FNN yang telah terlatih kemudian disimpan parameternya agar bisa digunakan pada tahap inferensi. Keseluruhan proses pelatihan model FNN digambarkan pada Gambar 3.2.



Gambar 3.2 Tahapan Pelatihan Model

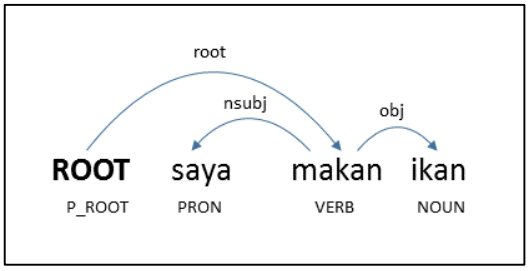
Tugas selanjutnya adalah melakukan penguraian kalimat untuk mengetahui pohon dependensi. Tahap pertama yaitu menyiapkan sistem transisi. Sistem transisi menerima kalimat dan menginisiasikan konfigurasi awal. Berikutnya sistem transisi ini akan meminta transisi dari model FNN. Prediksi transisi tersebut dijalankan oleh sistem transisi untuk berpindah ke konfigurasi selanjutnya. Proses ini berulang sampai konfigurasi terminal dicapai oleh sistem transisi. Pohon dependensi diketahui dari variabel *arc* yang menyimpan hubungan dependensi utuh di konfigurasi terminal. Tahapan penguraian kalimat ke bentuk pohon dependensi ditunjukan pada Gambar 3.3.



Gambar 3.3 Tahapan Penguraian Kalimat

3.3 Arsitektur Model

Salah satu komponen utama yang dimiliki oleh pengurai dependensi adalah model *Neural Network* (NN). Model NN bertugas sebagai *classifier* untuk memprediksi transisi dari suatu konfigurasi. Untuk memahami model NN dalam memprediksi transisi, suatu kalimat yang sudah terurai serta urutan transisinya diberikan sebagai contoh.



Gambar 3.4 Contoh Kalimat yang Terurai

Gambar 3.4 memperlihatkan kalimat “saya makan ikan” yang disertai dengan *part of speech* (POS) berserta hubungan dependensinya. Untuk menghasilkan

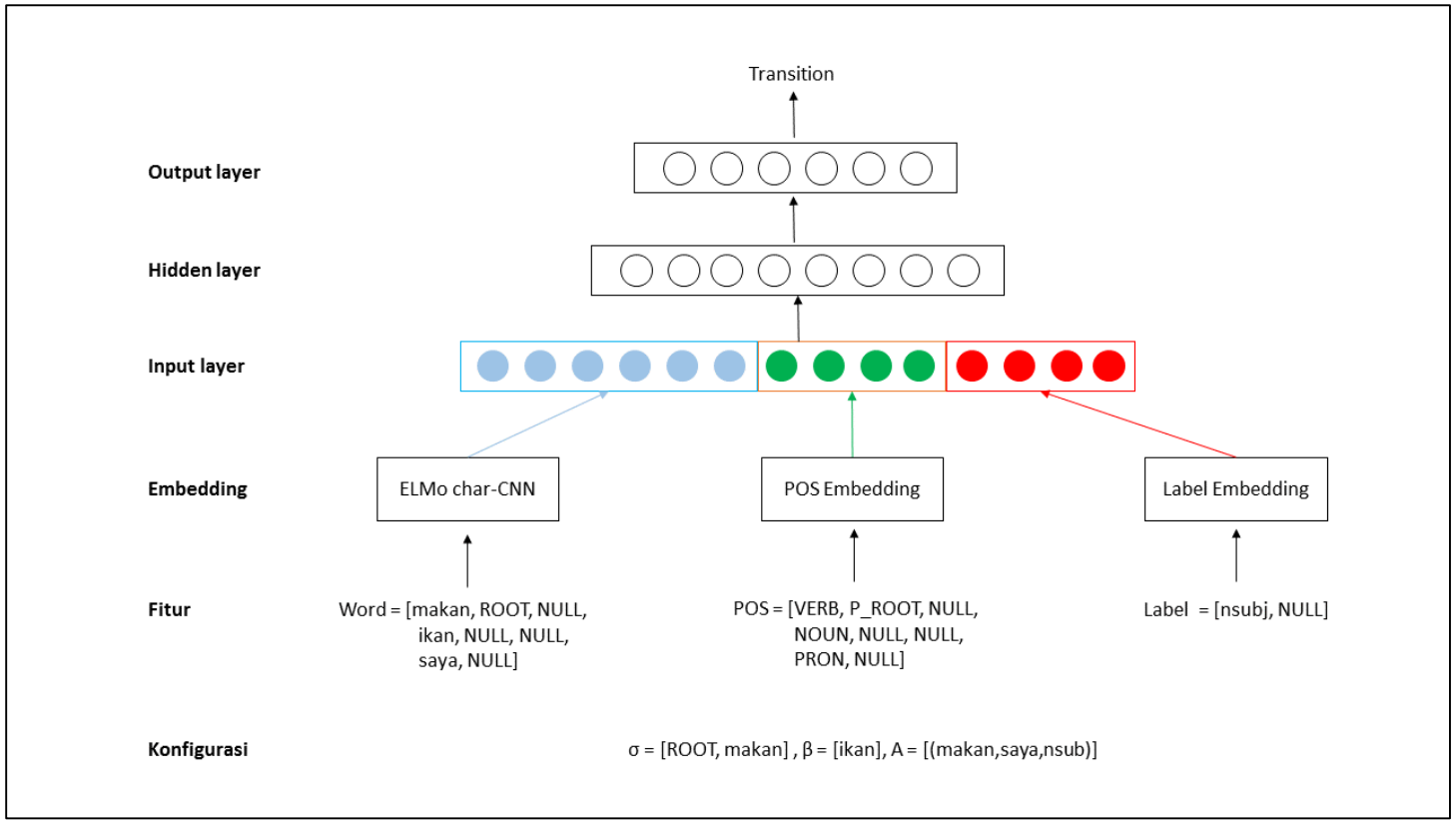
hubungan dependensi pada Gambar 3.4, sistem transisi menjalankan sederetan transisi. Urutan transisi diperlihatkan pada Tabel 3.1.

Tabel 3.1 Urutan transisi untuk Kalimat di Gambar 3.4

#	**Stack** σ	**Buffer** β	**Arc** A	**Action**
1	[ROOT]	[Saya, makan,ikan]	[]	init
2	[ROOT, Saya]	[makan,ikan]	[]	Shift
3	[ROOT, Saya, makan]	[ikan]	[]	Shift
4	[ROOT, makan]	[ikan]	[(makan,Saya,nsubj)]	LA
5	[ROOT, makan, ikan]	[]	[(makan,Saya,nsubj)]	Shift
6	[ROOT, makan]	[]	[(makan,Saya,nsubj), (makan,ikan,obj)]	RA
7	[ROOT]	[]	[(makan,Saya,nsubj), (makan,ikan,obj), (ROOT,makan,root)]	RA

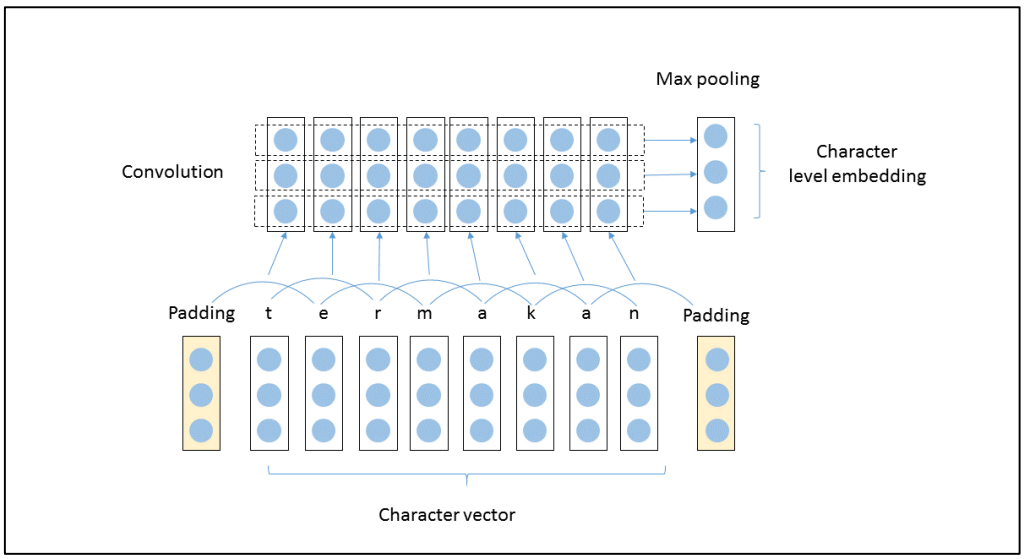
Konfigurasi yang ada pada Tabel 3.1 perlu diabstraksi melalui proses ekstraksi fitur. Proses ini memilih beberapa komponen dari konfigurasi yang dinilai merepresentasikan keseluruhan konfigurasi. Contoh komponen yang dipilih misalnya tiga kata teratas di *stack* σ, tiga kata teratas di *buffer* β, *left most child* dan *right most child* dari kata teratas di *stack* σ. Pada implementasinya penelitian ini menggunakan fitur yang digunakan penelitian Chen dan Manning (2014). Fitur lengkap bisa dilihat pada subbab 2.4.1. Setelah mengetahui konfigurasi serta representasi fitur maka *input features* bagi model NN dapat diketahui. Gambar 3.5 mendeskripsikan proses *forward pass* yang dilakukan oleh model NN pada konfigurasi ke-4 dari Tabel 3.1. Mulanya fungsi ekstraksi fitur menerima konfigurasi dan memilih kata-kata yang sesuai dengan representasi fitur. Setiap kata tersebut akan direpresentasikan dengan *word identity*,

POS tag dan *dependency label*. Ketiga fitur tersebut digabungkan bersama sebagai *input features* dari model NN. *Input features* tersebut diteruskan ke *hidden layer*. Dari *hidden layer* diteruskan lagi ke output layer. Output layer mengembalikan probabilitas atas semua kemungkinan transisi. Transisi yang diambil adalah transisi dengan nilai probabilitas tertinggi. Pada Gambar 3.5 dapat dilihat bahwa kata direpresentasikan menggunakan vektor tingkat karakter dari ELMo char-CNN. Kontras dengan penelitian Chen dan Manning (2014) yang menggunakan vektor di tingkat kata menggunakan GloVe.



Gambar 3.5 Arsitektur dari Model NN

Arsitektur dari ELMo char-CNN dapat dilihat pada Gambar 3.6. Contoh tersebut mengambil kata “termakan”. Untuk mendapatkan vektor dari kata “termakan”, langkah pertama yaitu memenggal kata karakter per karakter. Lalu dari *character embedding* ubah setiap karakter yang berbentuk *string* tersebut ke bentuk vektor. Vektor-vektor tersebut dijajarkan untuk membentuk matrik layaknya sebuah gambar 2d. Proses selanjutnya adalah konvolusi yang melibatkan perkalian matriks antara area konvolusi dengan filter. Hasil keluaran konvolusi tersebut diteruskan ke *max pooling layer* yang akan mereduksi dimensi keluaran dengan mencari nilai terbesar di setiap kolum. Hasil dari *max pooling layer* inilah yang menjadi representasi kata tingkat karakter dari kata “termakan”.

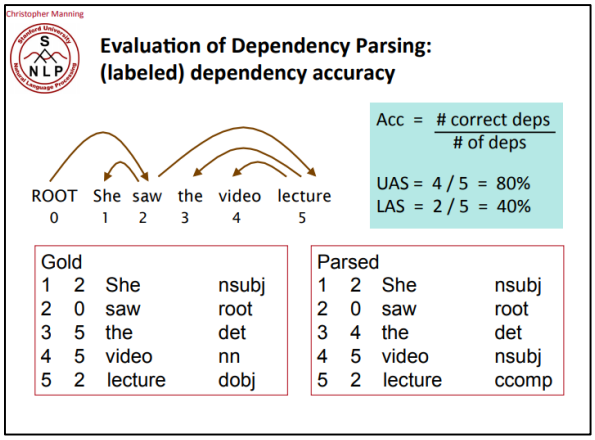


Gambar 3.6 Arsitektur dari ELMo char-CNN

3.3 Variabel Penelitian

Pengukuran standar yang digunakan oleh Conference on Computational Natural Language Learning (CoNLL) untuk mengetahui performa pengurai dependensi

adalah LAS dan UAS (Buchholz dan Marsi, 2014). UAS ialah persentase berapa banyak dependensi yang benar. LAS ialah persentase berapa banyak dependensi dan label dependensi yang benar.



Gambar 3.7 Nilai UAS dan LAS Suatu Dependency Parser (Manning, 2019)

Perhitungan UAS dan LAS untuk suatu kalimat ditunjukan pada Gambar 3.7. Kebenaran pohon dependensi suatu kalimat yang dihasilkan pengurai dependensi dibandingkan dengan pohon dependensi kalimat yang sudah dianotasi oleh ahli.

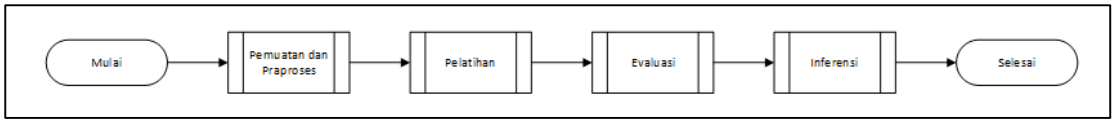
3.4 Perancangan Sistem

Pada tahap ini dilakukan perancangan sistem usulan dengan menggunakan diagram perancangan sistem.

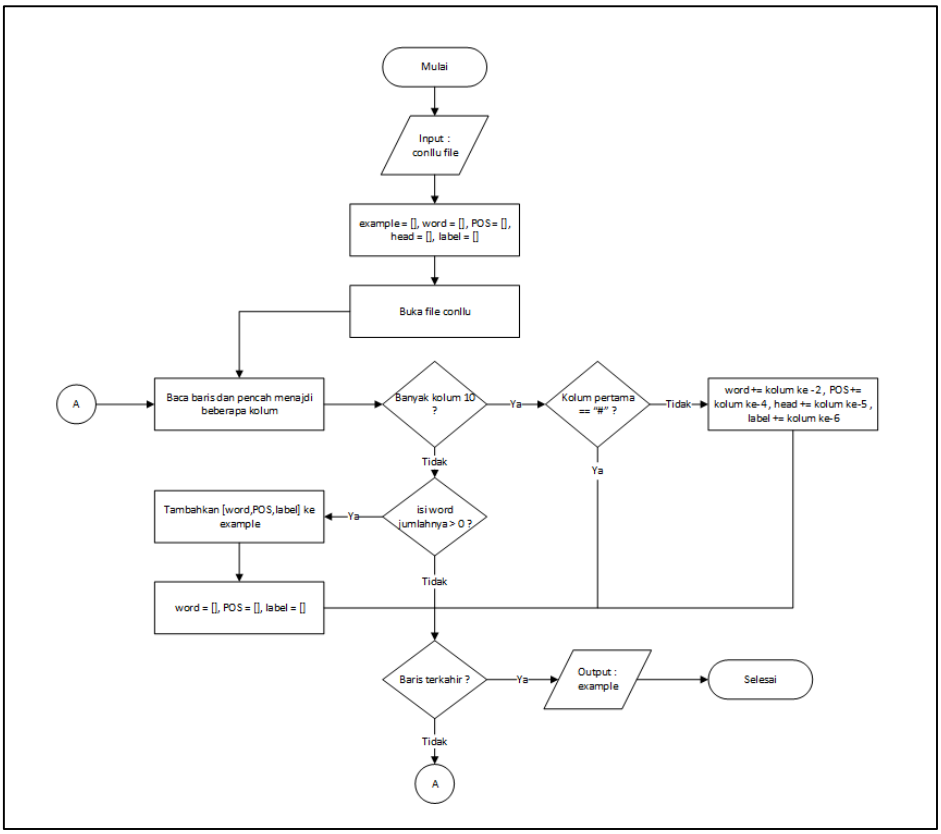
3.4.1 Flowchart

Flowchart atau bisa disebut diagram alur merupakan bagan-bagan yang menggambarkan langkah-langkah dan proses dari suatu sistem. Berikut penjabaran *flowchart* dari *dependency parser* yang dikembangkan. Secara garis besar, *dependency parser* ini melibatkan beberapa proses utama. Dimulai dengan proses

pemuatan dan praproses yang menyiapkan data training, development dan testing. Setelah data siap, proses pelatihan dapat dilakukan. Pada proses pelatihan ini model NN dilatih menggunakan training data dan algoritma *gradient decent learning*. Setelah model selesai dilatih, proses selanjutnya adalah proses evaluasi. Evaluasi dilakukan menggunakan testing data dan *metric* yang diujikan adalah UAS dan LAS. Setelah mengetahui performa dari *dependency parser*, langkah terakhir adalah proses inferensi. Model NN yang terlatih akan di-*deploy* ke aplikasi web. Aplikasi web ini menyediakan *UI* untuk melakukan inferensi dan memberikan visualisasi terhadap pohon dependensi yang dihasilkan. Keseluruhan proses utama pada pengembangan *dependency parser* ditunjukan pada Gambar 3.8.

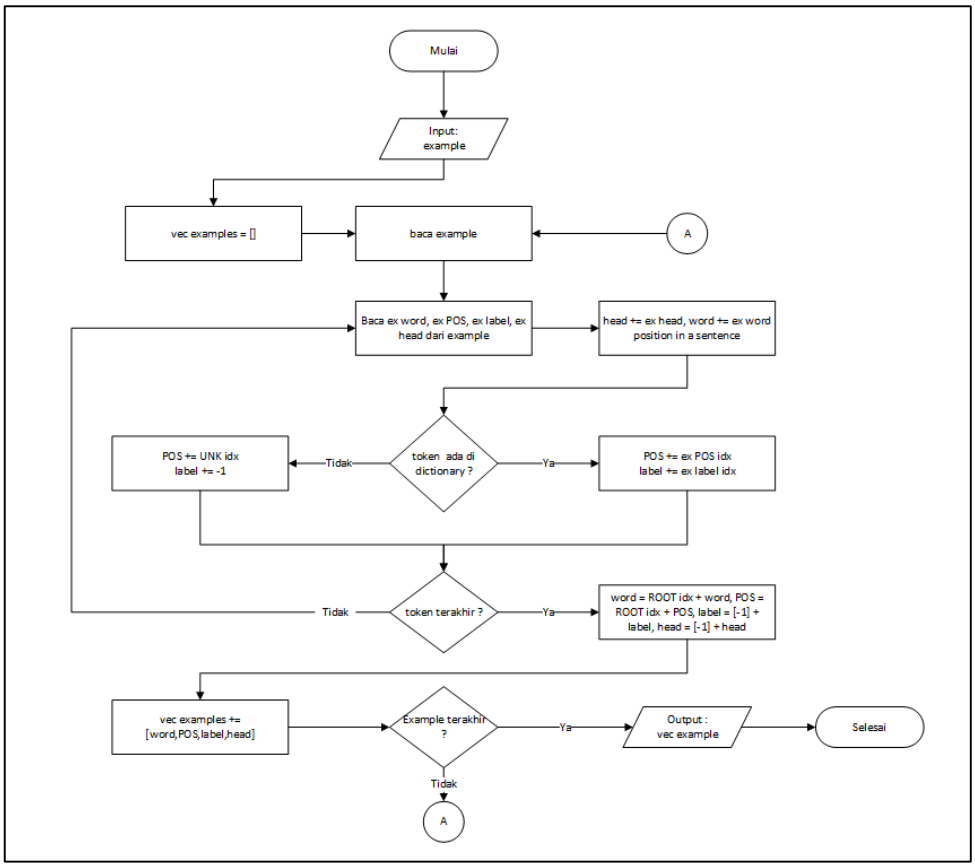


Gambar 3.8 Flowchart Proses Utama



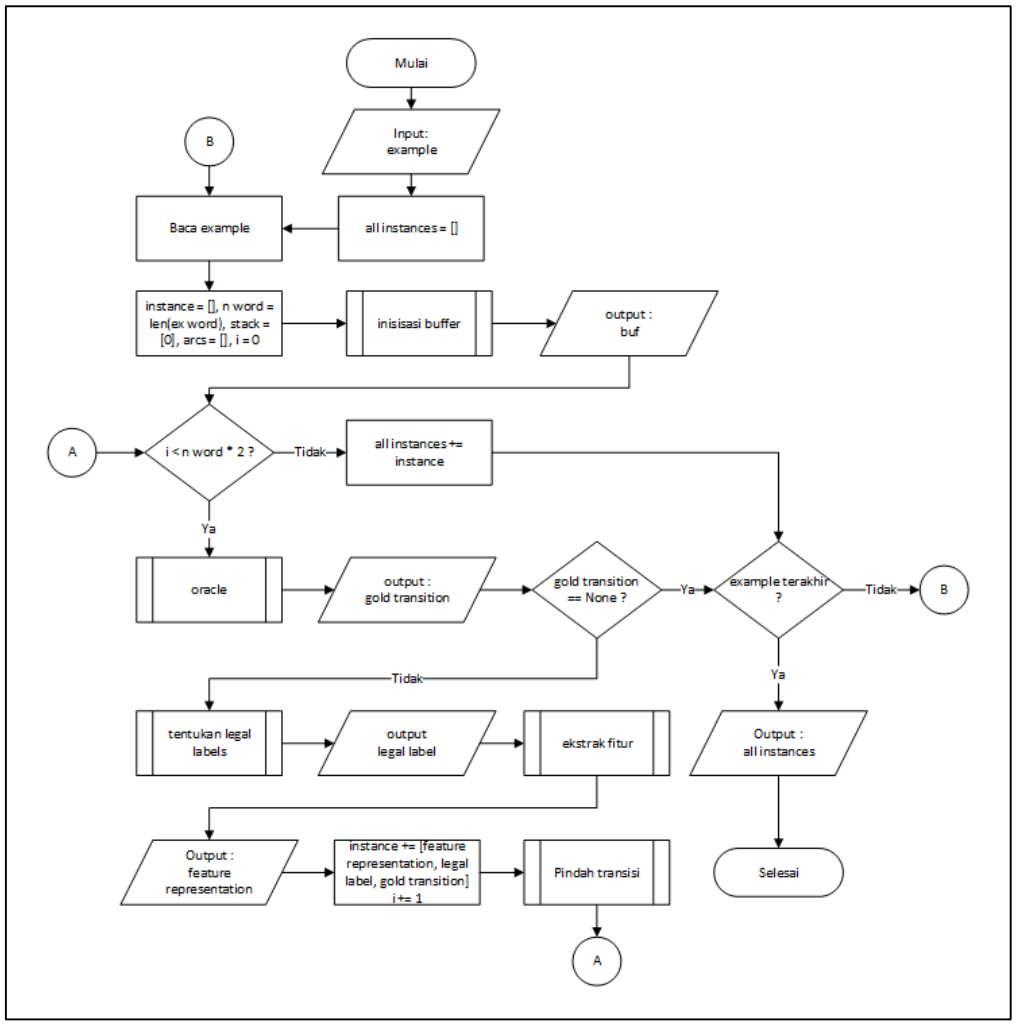
Gambar 3.9 Flowchart Baca Dataset

Sebelum memulai training dependency parser, langkah awal yang dilakukan adalah membaca dataset. Pada setiap example informasi yang dipakai untuk training adalah word, POS tags, head, dan label. Proses membaca dataset ini digambarkan pada Gambar 3.9.



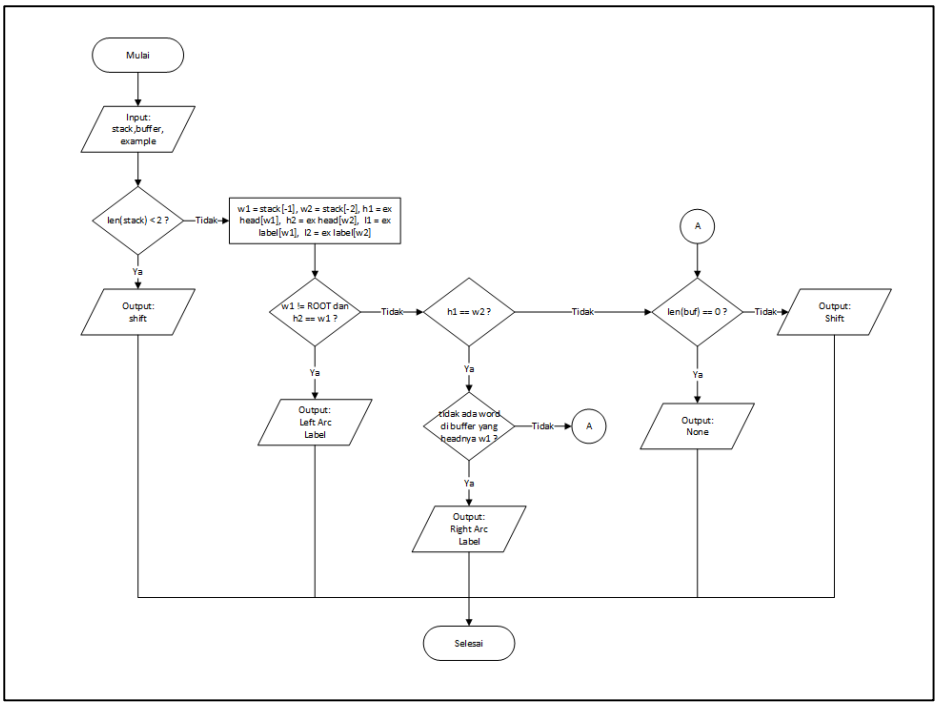
Gambar 3.10 Flowchart Vektorisasi Dataset

Dataset yang dibaca pada proses sebelumnya, disimpan dalam bentuk string berupa kata, untuk mempermudah mapping kata tersebut kedalam bentuk *vector embedding*-nya maka diperlukan upaya merubah bentuk string tersebut ke bentuk numerik. Prosesnya digambarkan pada Gambar 3.10.



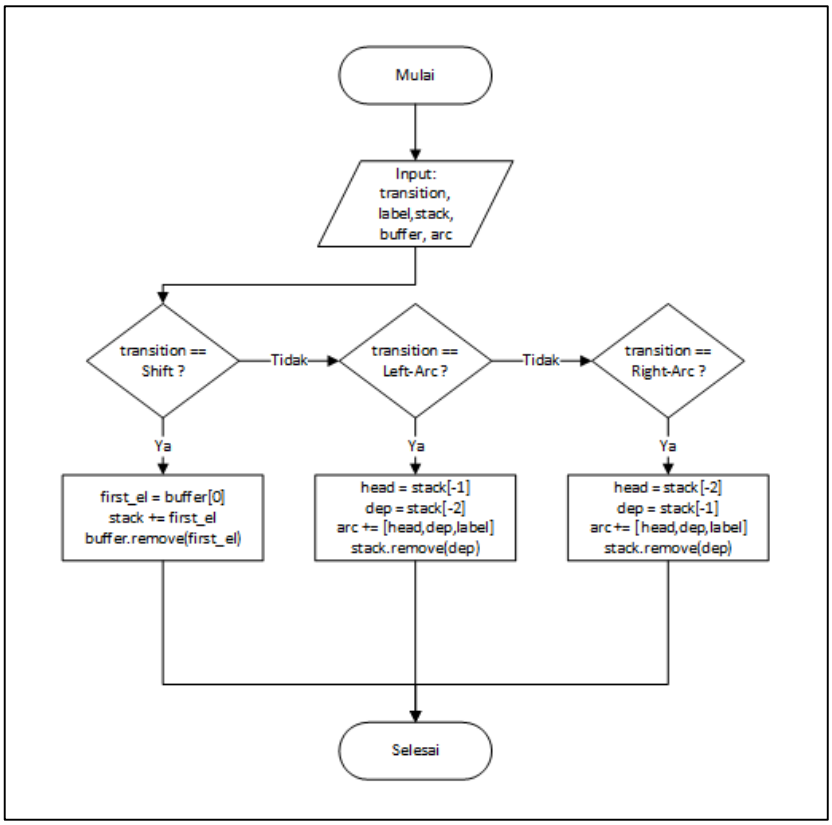
Gambar 3.11 Flowchart Buat Training Example

Training example yang ada pada dataset perlu dibuah kebentuk urutan transisi agar sesuai dengan metode *transition based* yang dipergunakan oleh *dependency parser*. Pembuatan *training example* ditunjukan pada Gambar 3.11. Setiap kalimat di training example diproses oleh suatu sistem transisi, dan dikonversikan menjadi urutran-urutan transisi yang terdiri dari konfigurasi awal sampai konfigurasi akhir.



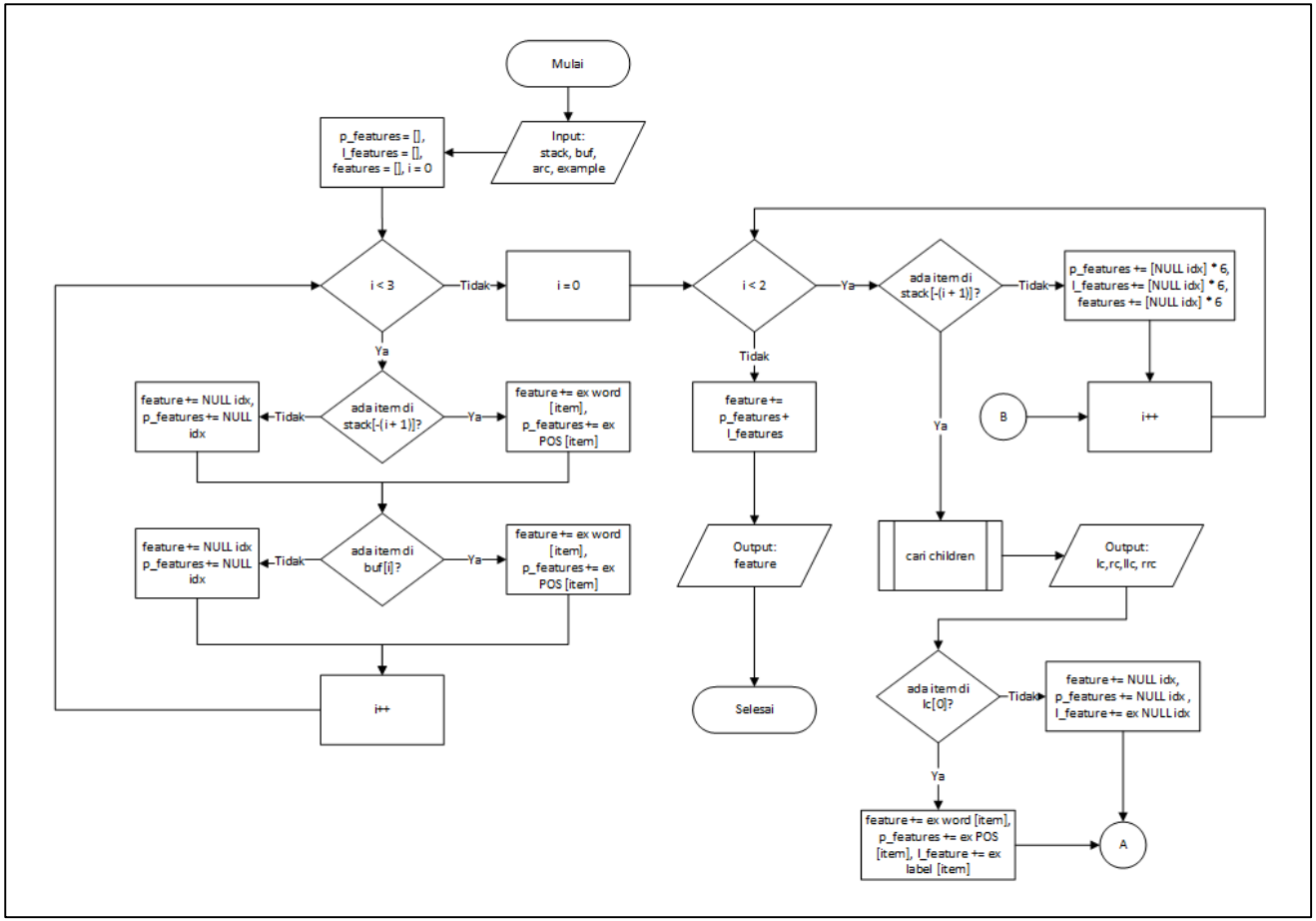
Gambar 3.12 Flowchart Oracle

Sistem transisi ini memerlukan oracle yang bertugas untuk memberikan instruksi atau transisi mana yang harus dijalankan agar bisa mencapai konfigurasi akhir. Oracle agar bisa memprediksi transisi, ia harus menerima suatu konfigurasi yang merupakan state dari sistem transisi yaitu berupa stack, buffer, dan arc. Cara kerja *oracle* dijabarkan pada Gambar 3.12 Saat menerima gold transition dari Oracle, sistem transisi perlu menjalankan atau beripindah ke transisi tersebut, bagaimana sistem transisi berpindah transisi lebih jelasnya digambarkan pada Gambar 3.13.

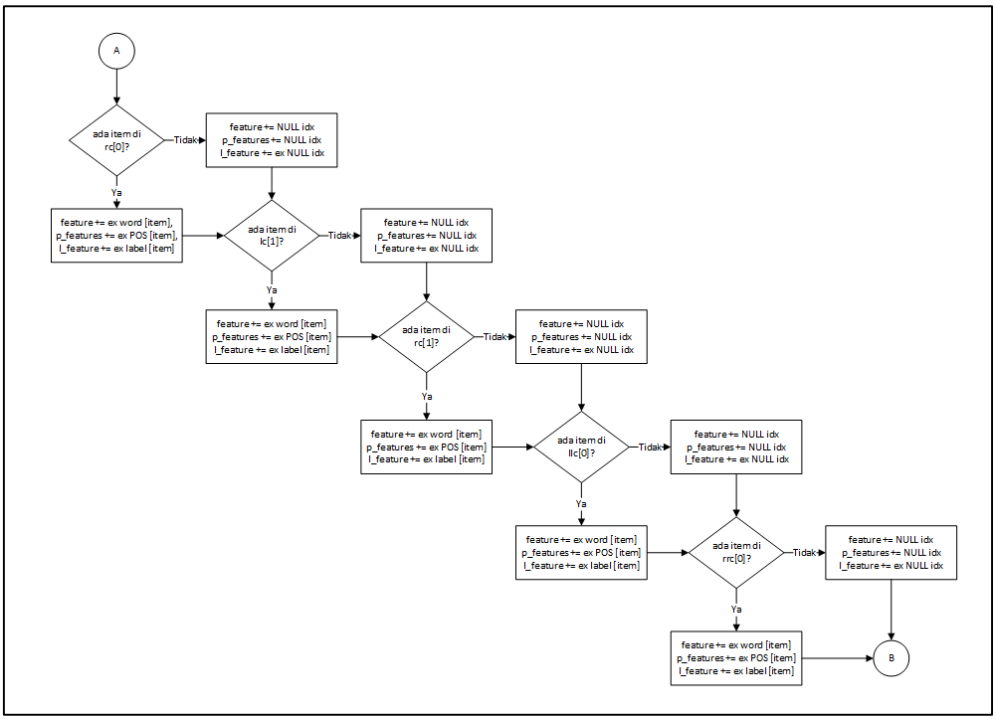


Gambar 3.13 Flowchart Pindah Transisi

Setelah uratan transisi milik masing-masing training example sudah terbentuk, langkah selanjutnya adalah mengekstrak fitur dari konfigurasi atau transisi itu. Representasi Fitur ini layaknya sebuah abstraksi terhadap segala kombinasi konfigurasi yang mungkin muncul. Ektraksi fitur dijabarkan pada Gambar 3.14. dan Gambar 3.15.

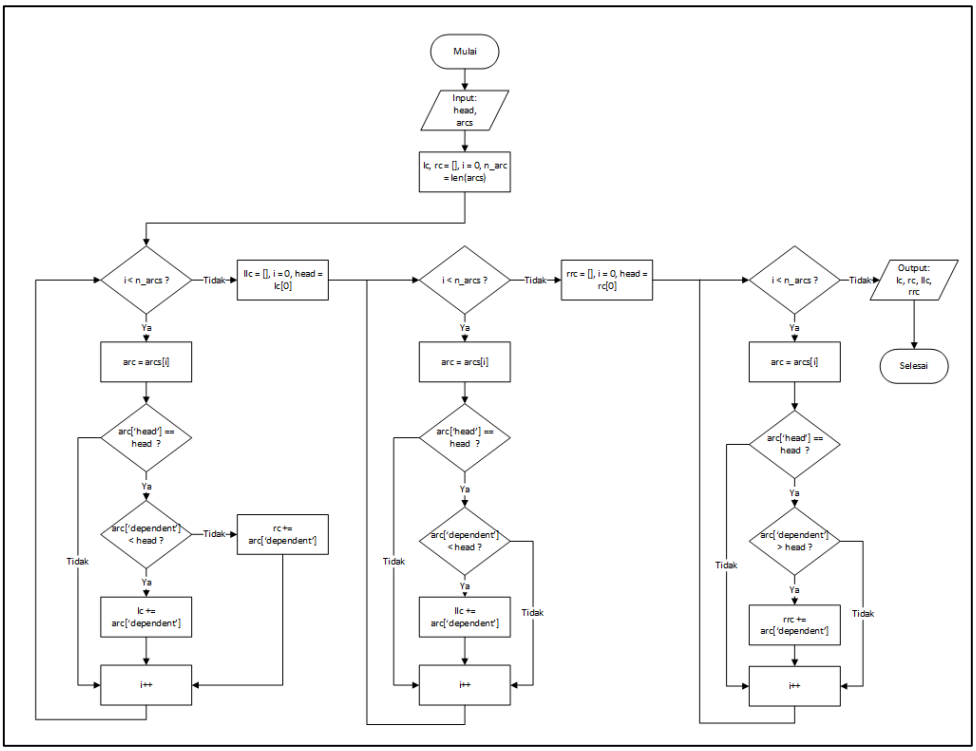


Gambar 3.14 Flowchart Ekstrak Fitur



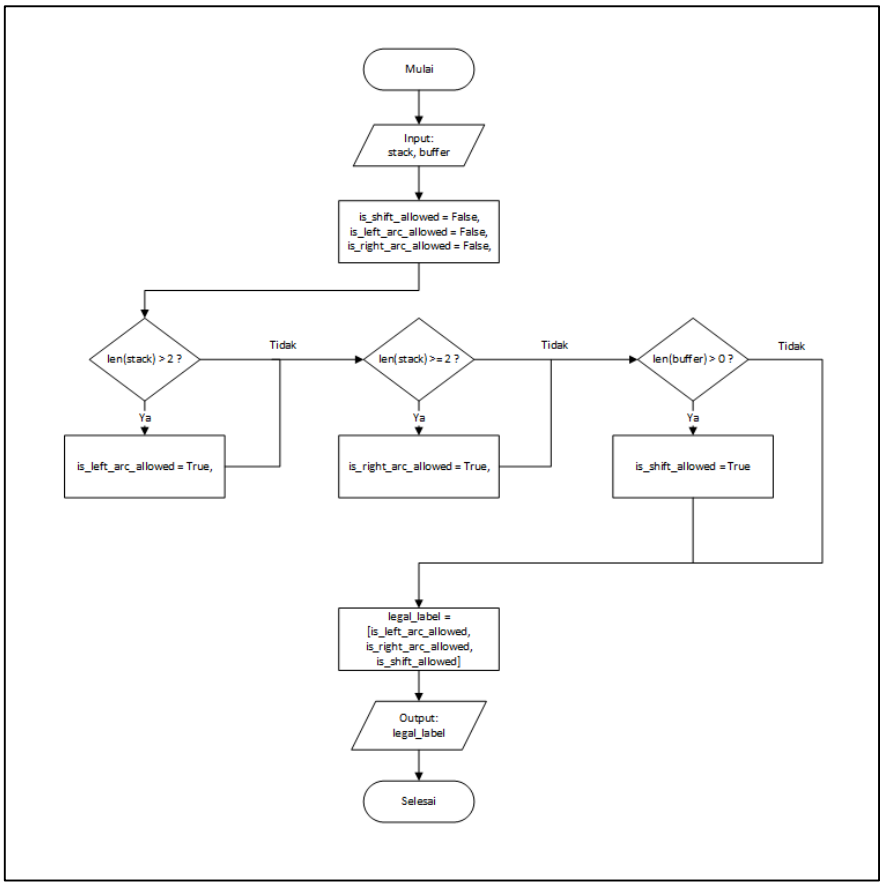
Gambar 3.15 Flowchart Ekstrak Fitur Lanjutan

Salah satu representasi fitur yang diikutsertakan adalah keberadaan *leftmost* dan *rightmost children* daripada suatu konfigurasi. Mereka memberitahukan *dependency relation* atau biasa disebut arc, yang sudah terbentuk pada konfigurasi sebelumnya. Langkah-langkah untuk mencari *lefmost* dan *rightmost children* diterangkan pada Gambar 3.16.



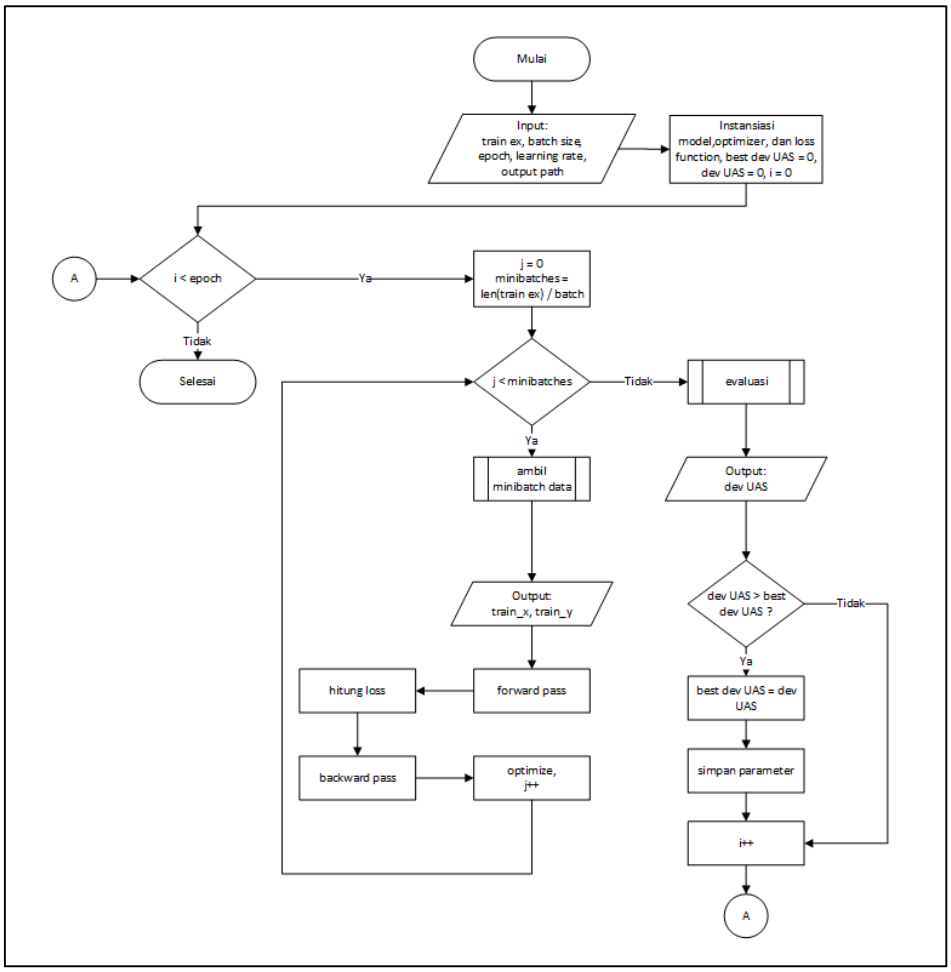
Gambar 3.16. Cari Children

Transisi yang diberikan oleh Oracle terjamin kebenarannya karena prediksinya didasarkan pada gold parse yang berasal dari dataset. Dalam proses training, NN classifier akan mencoba menirukan fungsi Oracle, namun prediksinya mungkin masih belum sempurna dan kadang kala memberikan transisi yang tidak valid, oleh sebab itu setiap training example harus disertai dengan *valid transition* yang memberitahukan transisi mana saja yang boleh dijalankan pada sistem transisi. Penentuan akan transisi yang valid ditunjukan pada Gambar 3.17.



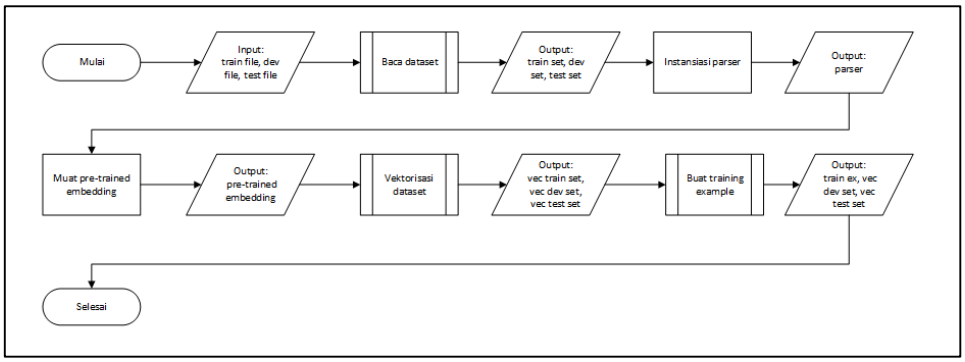
Gambar 3.17 Tentukan Legal Label

Semua informasi yang dibutuhkan untuk merubah training example menjadi urutan transisi sudah tersedia mulai dari *gold transition Oracle*, *feature representation*, dan juga *legal labels*. Pada saat nanti tranining example diberikan ke NN untuk training, input yang diterima adalah *feature representation* dan dengan *expected output* berupa *gold transition*. Proses training dapat dilihat pada Gambar 3.18.



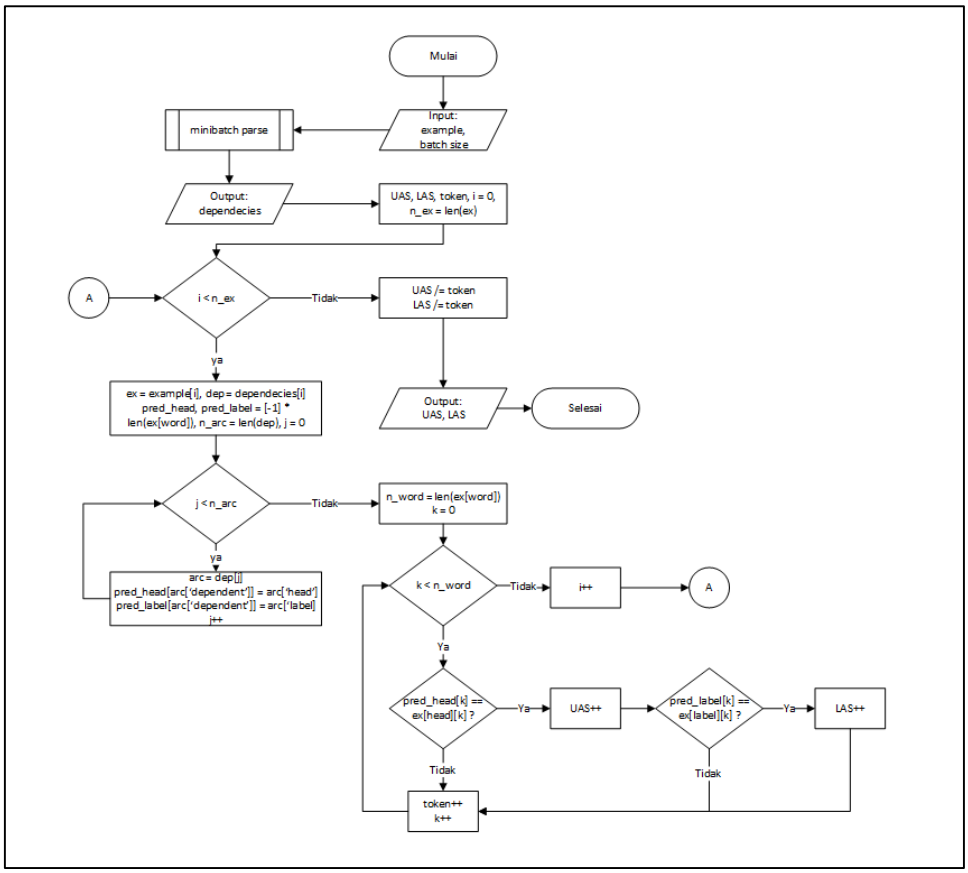
Gambar 3.18 Training

Proses traning diawali dengan memuat segala komponen pendukung yang telah dijabarkan pada *flowcharts* sebelumnya, dan beberapa hal lain seperti instansiasi objek Parser dan loading pre-trained word embedding. Fungsi ini jelasnya ada pada Gambar 3.19.



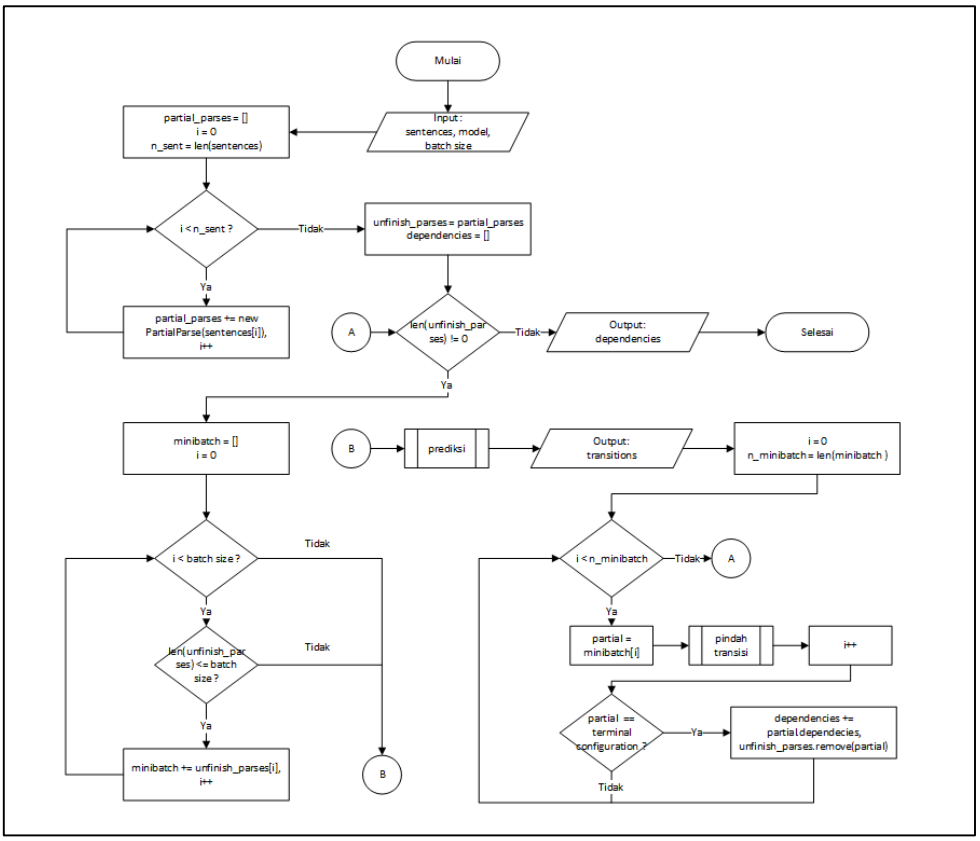
Gambar 3.19 Pemuatan dan Praproses

Tak lupa, sebelum melaksanakan training ada beberapa hyperparameter yang perlu ditetapkan mulai dari epoch, batch size, learning rate, dan output path untuk menyimpan parameter selepas training. Di samping itu, *optimizer* dan *loss function* juga ikut ditetapkan. Training akan diiterasi sebanyak jumlah epoch, dan di setiap epochnya training example akan dibuatkan minibatch datanya, agar tidak perlu menunggu 1 *epoch* selesai untuk melakukan update. Mini batch data yang merupakan pasangan input dan output ini, akan diberikan kepada NN *classifier* untuk memprediksi transisi keluaran. Hasil prediksi akan dihitung nilai lossnya terhadap *gold transition*. Lalu error tersebut akan diteruskan ke layer-layer dibelakangnya ini mengacu pada proses *backpropagation*, untuk kemudian dilakukan parameter update untuk meminimalkan loss tersebut. Setelah iterasi sudah dilakukan sesuai jumlah epoch, artinya sudah selesai training, langkah selanjutnya adalah melakukan evaluasi untuk mengetahui UAS dan LAS dari parser ini. Evaluasi ditunjukan pada Gambar 3.20.



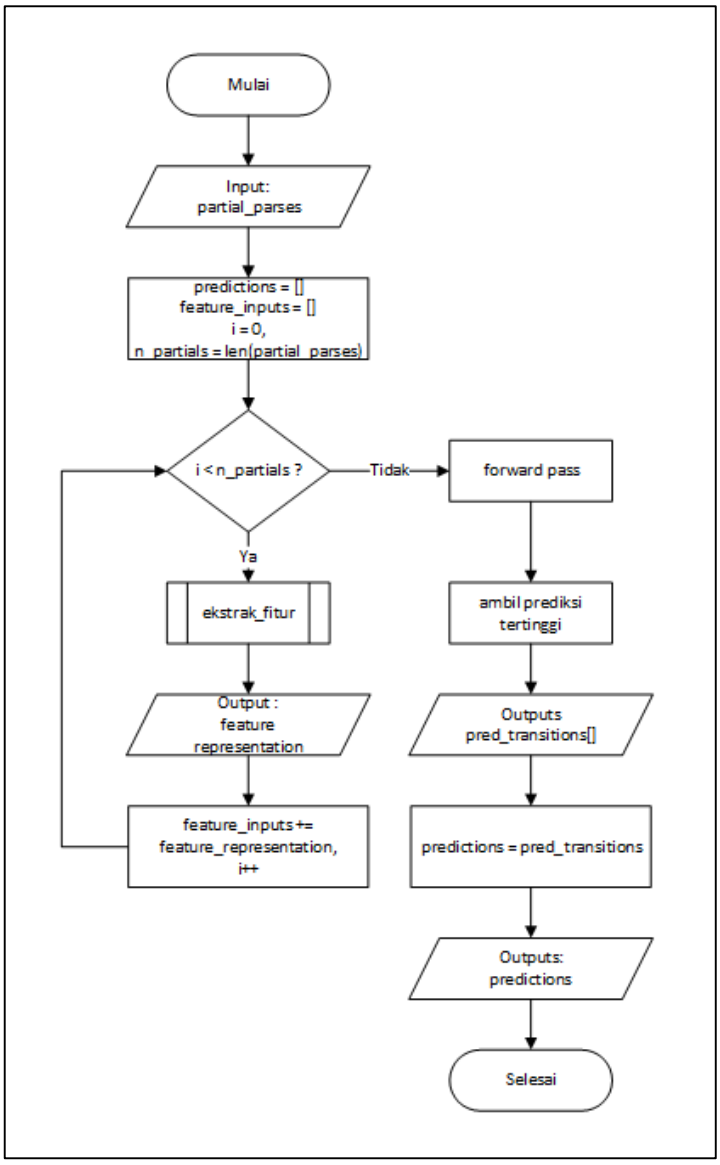
Gambar 3.20 Evaluasi

Sama halnya pada proses training, proses evaluasi melibatkan parser untuk memprediksi transisi hanya saja dilakukan pada dev dan test set, bukan dengan train set. Prediksi akan dilakukan terhadap minibatch data yang dijelaskan pada Gambar 3.21.



Gambar 3.21 Minibatch Parse

Setiap example atau kalimat akan dibuatkan sistem transisi beserta konfigurasinya, untuk kemudian secara batch diberikan kepada parser untuk diprediksi transisinya masing-masing. Prediksi ini terus dilakukan untuk setiap konfigurasi yang belum menemui konfigurasi terminal. Proses predkisi digambarkan pada Gambar 3.22.



Gambar 3.22 Prediksi

Predikisi yang dilakukan mirip pada saat proses training, dan hanya sebatas *forward pass* saja. Prediksi dari NN tersebut kemudian akan dijalankan pada setiap sistem transisi, dan setelah sistem transisi tersebut mencapai konfigurasi terminal maka *dependency relation* yang utuh bisa diketahui. Ketika *dependency relation* milik semua

example sudah diketahui, maka proses evaluasi dapat dilakukan, dimulai dengan menghitung UAS yaitu berapa banyak arc atau head-dependent relation yang sesuai dengan gold parsed tree. LAS juga ikut dihitung, selain arc-nya yang harus benar, LAS juga memperhitungkan kesesuain label atau dependency type dari arc tersebut.