Topics for today

□ Architecture of TTS systems

□ Festival's Utterance structure:
– utterances, relations, items and features

□ Festival's module structure:
– breakdown of TTS processes

TTS architecture

□ Large systems need structure

□ TTS Utterances:
– representation of words, syllables, phones, etc.

□ TTS processes:
– Lexical lookup, duration prediction,
– waveform generation

TTS architecture

□ Try to make things modular:
– but there will be dependencies

□ Allow swapping of modules:
– testing different technique in *same* environment

□ Don't build-in language specifics:
– no fixed phoneset
– allow things to be dynamic

□ Have a scripting language:
– You can't guess all the necessary params
– So allow the user to control things

Utterance architectures (1)

String model:
– A single string replaced with lower level items

□ Tokens
  – *Feb 25*

□ Words
  – *february twenty fifth*

□ Phones
  – *f eh b r ax r iy t w eh n t iy f ih f th*

Simple, but lose information about higher levels

Utterance architectures (2)

Table model:

– multi-leveled table model

<table>
<tr><td colspan="7">Feb</td><td colspan="10">25</td></tr>
<tr><td colspan="7">february</td><td colspan="6">twenty</td><td colspan="4">fifth</td></tr>
<tr><td colspan="3">1</td><td colspan="2">0</td><td>0</td><td>0</td><td colspan="4">1</td><td colspan="2">0</td><td colspan="4">1</td></tr>
<tr><td>f</td><td>eh</td><td>b</td><td>r</td><td>ax</td><td>er</td><td>iy</td><td>t</td><td>w</td><td>eh</td><td>n</td><td>t</td><td>iy</td><td>f</td><td>ih</td><td>f</td><td>th</td></tr>
</table>

– no tree structures

– one hierarchy

– no explicit connections between levels

Utterance architectures (3)

Hetrogeneous Relation Graphs:

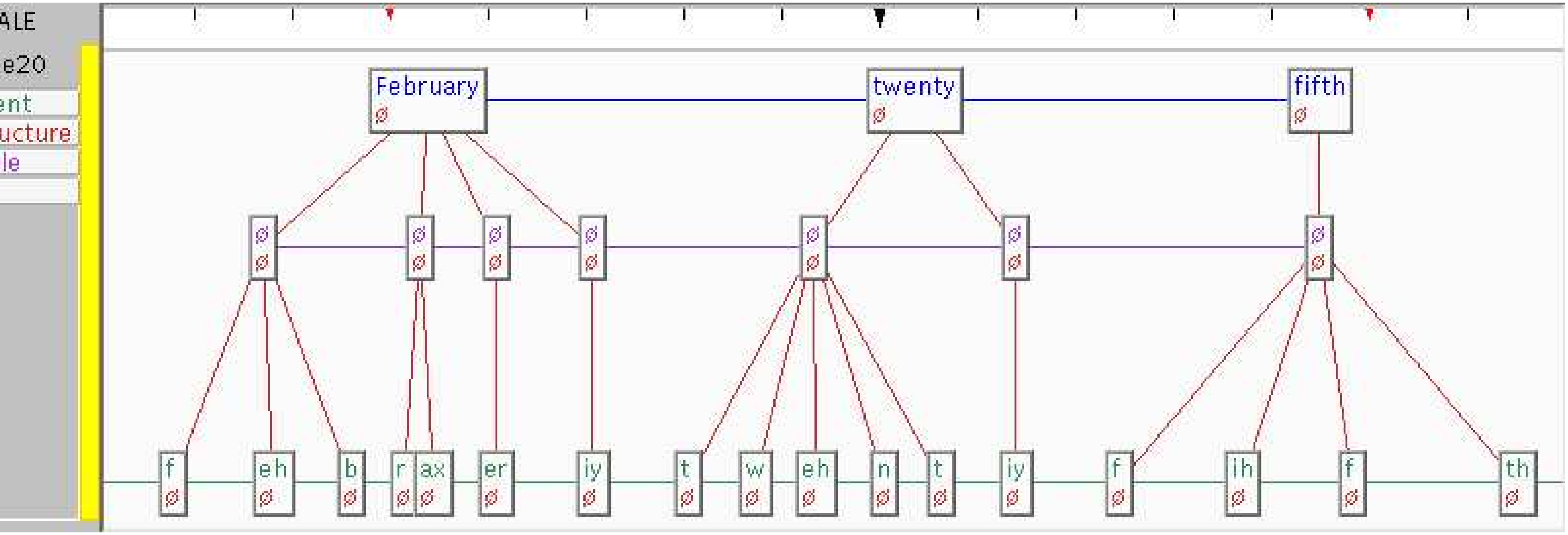
Utterances consist of a set of **items**.

Each **item** is related by one or more **relations**.

Each **item** contains a set of **features**.

Relations define lists, trees or lattices of **items**.

Hetrogeneous Relation Graphs



Items and features

Accessing information in an utterance.

-
  ```
  (set!  utt1 (SayText "The book is on the table"))
  ```

-
  ```
  (set!  firstword (utt.relation.first utt1 'Word)))
  ```
- Part of speech from word
  ```
  (item.feat firstword "pos")
  ```
- First syllable's stress
  ```
  (item.feat firstword
  "R:SylStructure.daughter1.stress")
  ```
- Next word's punctuation
  ```
  (item.feat firstword "n.pos")
  ```
- All words and pos in utterance
  ```
  (utt.features utt1 'Word '(name pos))
  ```

Feature access

□ clean traversal of the structure allows:
- feature-based prediction models
- CART, linear regression, ANN etc.

□ For example each item in a Segment relation dump:
- dur name n.name p.name R:SylStructure.parent.stress ...
- 0.15 pau dh pau 0
- 0.08 dh ax pau 0
- 0.09 ax dh b 0
- 0.07 b ax oy 1
- ...

Feature pathnames

□ Item based:
- *always* gives an answer.
- default value 0

□ `( IN-REL-MOVE | NEW-REL-MOVE ) * FEATNAME`
- `IN-REL-MOVE` := n. p. parent. daughter. ...
- `NEW-REL-MOVE` := R:RELATIONAME.
- `FEATNAME` := name pos duration
- `FEATNAME` := lisp_* ph_*

FEATNAME can also be built-in feature functions

Features and feature functions

□ Apply to an items
– *always* give an answer

□ Direct features:
– pos, name, stress

□ Calculated *Feature Functions*:
– start_duration := if (p == 0) 0 else p.end
– duration := end - start_duration
– num_syllables := ...
– prev_content_word := ...

□ Feature Functions :
– C++ functions (plus registration)
– Lisp based (slower but good for initial study)

Examples

assume `seg` is an item in the Segment relation

□ `name`
– the name of the segment

□ `n.name`, `p.name`
– names of next and previous segments (or “0”)

□ `R:SylStructure.parent.stress`
– the stress marking on the syllable of *seg*

□ `R:SylStructure.parent.parent.name`
– word name

□ `n.R:SylStructure.parent.parent.name`
– word on next segment

□ `R:SylStructure.parent.parent.R:Word.n.name`
– next word

HRG databases

Not just used at synthesis time

- □ Utterance in speech databases:
  - – converted to HRGs
  - – (semi-)automatically.
- □ Have *same* representations as if synthesized:
  - – can extract features for modelling
  - – can test sub-parts of the system
  - – on “natural” data.
- □ Can hold complex relationships

Festival relations

Different synthesizers in Festival may use different relations
"Standard" relations are

□ `Text`: character string of utterance.

□ `Token`: list of trees relating tokens to zero or more words. Leaves are in `Word` relation.

□ `Word`: a list of words.

□ `Phrase`: a list of trees over words.

□ `Syntax`: a single tree over words.

□ `SylStructure`: list of trees, roots are `Words`, middles are `Syllables`, leafs are `Segments`.

□ `Syllable`: a list of syllables.

□ `Segment`: a list of phones.

Festival relations

- □ `IntEvent`: a list of intonation events (accents and boundaries).
- □ `Intonation`: a list of trees rooted with syllables whose leafs are `IntEvents`.
- □ `F0`: a single F0 contour (as a track)
- □ `Unit`: list of diphones
- □ `Wave`: a single waveform.

Utterances and Modules

Each **Utterances** is declared with a **type**:

Types are declared with a list of **modules**.

Synthesis is defined in terms of the type of an utterances

```
(Utterance Text "Hello")
(Utterance Segment ((h 0.058) (@ 0.039)
           (l 0.069) (ou 0.219)))
```

```
(defUttType Word              (DetUttType Segment
      (Initialize utt)             (Initialize utt)
      (Word utt)                   (Wave_Synth utt))
      (Intonation utt)
      (Duration utt)
      (Int_Targets utt)
      (Wave_Synth utt))
```

TTS modules

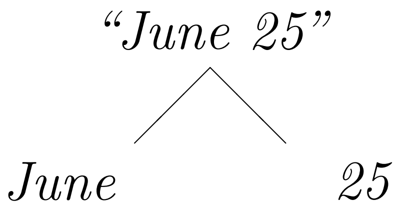
- □ Text: tokenize strings of chars into tokens.
- □ Token_POS: identify and tag homographs
- □ Token: convert Tokens to Words
- □ POS: part of speech tagger
- □ Phrasify: statistical phraser.
- □ Word: lexical lookup, builds Syls and Segs
- □ Pauses: adds silences
- □ Intonation: predicts accents and boundaries
- □ PostLex: post-lexical rules (swha, 's etc)
- □ Duration: segmental durations
- □ Int_Target: F0 prediction
- □ Wave_Synth: waveform generation
  – get_diphones, map_prosody
  – reconstruct waveform (RELPC)

modules

relations

“June 25”

Text



modules

Tokenize

"June 25"

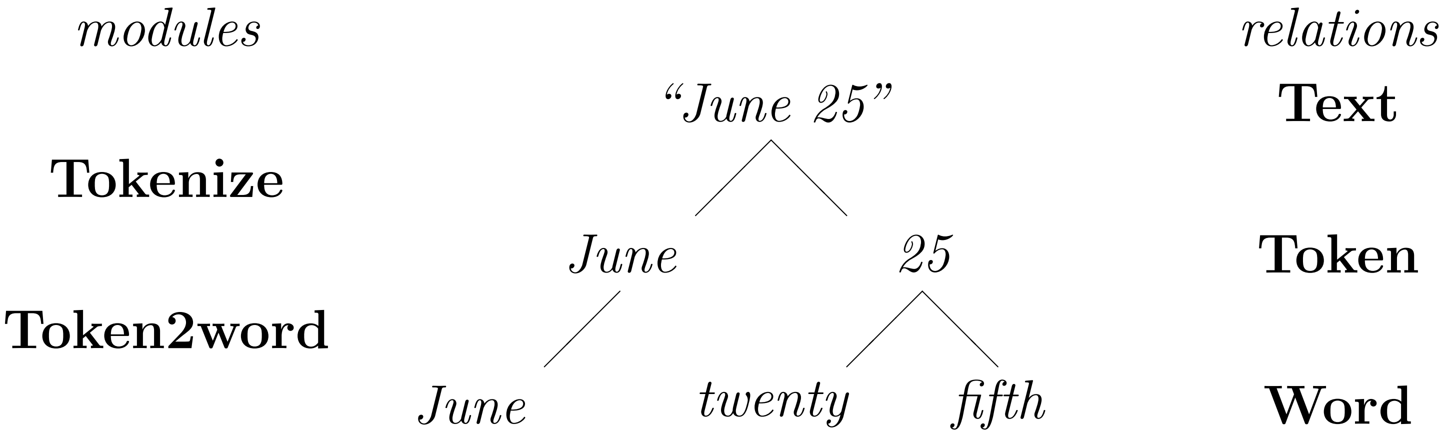
June

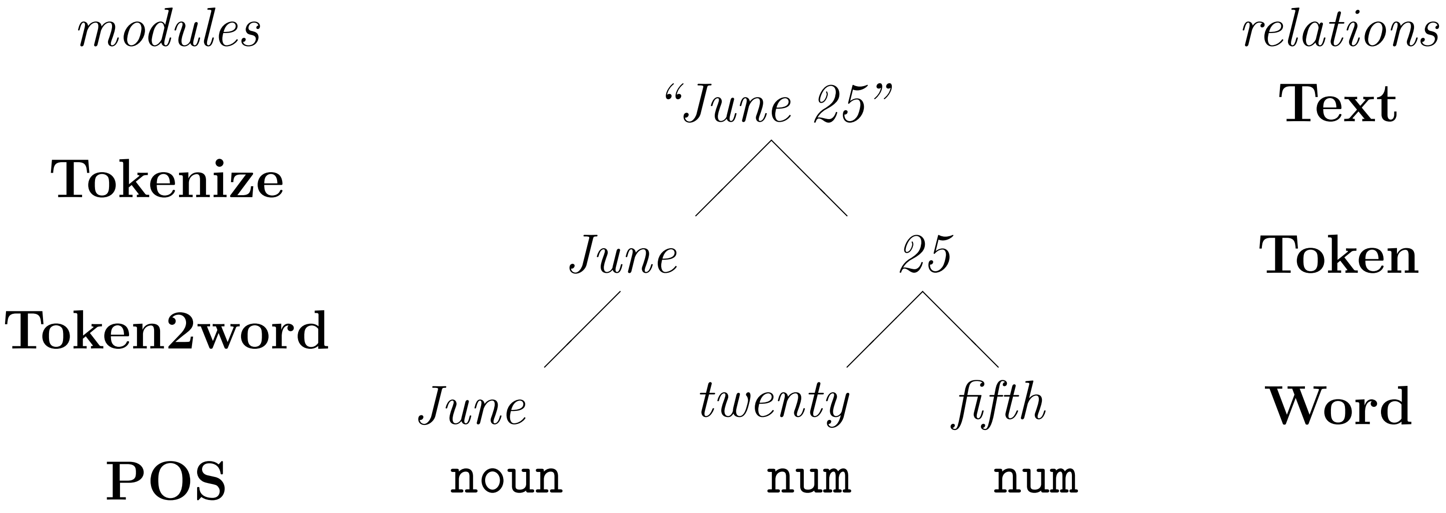
25

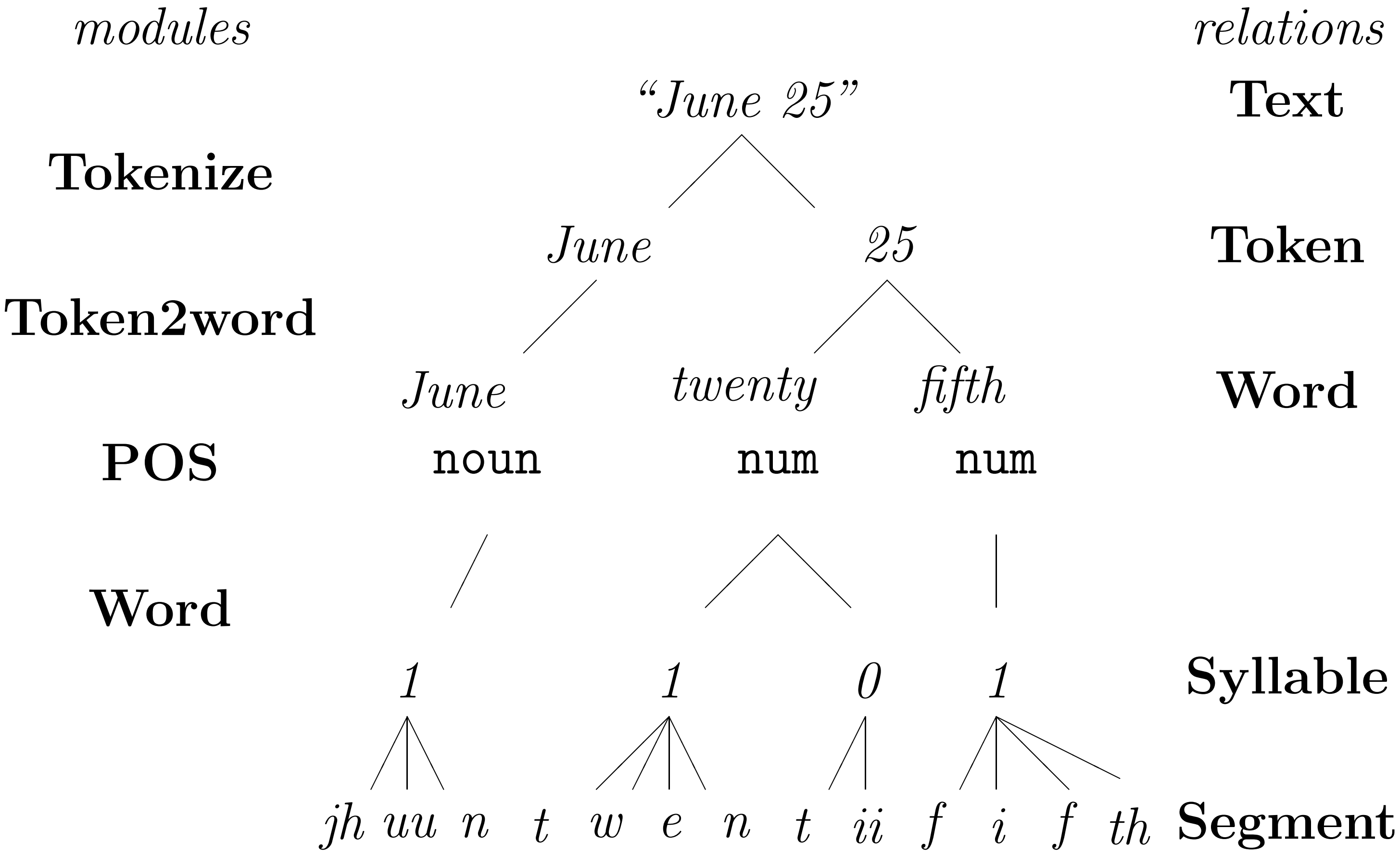
relations

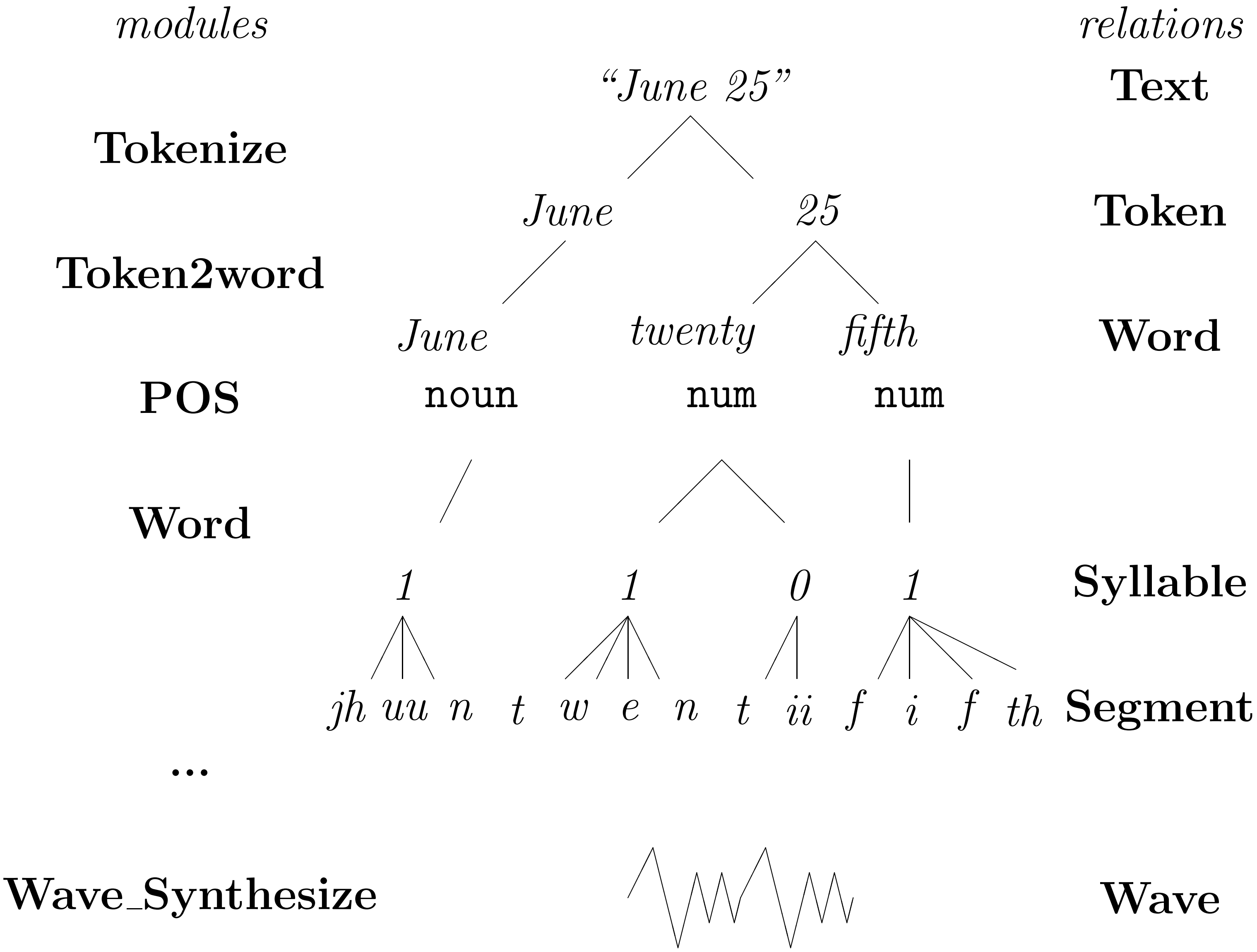
Text

Token


modules
relations
“June 25”
Text
Tokenize
June
25
Token
Token2word
June
twenty
fifth
Word


modules
relations
"June 25"
Text
Tokenize
June
25
Token
Token2word
June
twenty
fifth
Word
POS
noun
num
num


modules
relations
"June 25"
Text
Tokenize
June
25
Token
Token2word
June
twenty
fifth
Word
POS
noun
num
num
Word
1
1
0
1
Syllable
jh
uu
n
t
w
e
n
t
ii
f
i
f
th
Segment


modules
relations
"June 25"
Text
Tokenize
June
25
Token
Token2word
June
twenty
fifth
Word
POS
noun
num
num
Word
1
1
0
1
Syllable
jh uu n t w e n t ii f i f th
Segment
...
Wave_Synthesize
Wave