

The Festival Speech Synthesis System

Seminar Two:

Inside Festival



CSTR

Outline

This session first looks at data is represented inside Festival and how it can be manipulated.

We then look at some of the details of Unit Selection speech synthesis.

CSTR

The utterance

The utterance is the container object in which speech synthesis occurs.

Information is added to the utterance by each stage of the synthesis process.

Most of this information is stored in HRGs (heterogeneous relation graphs) as **relations**, **items** and **features**

CSTR

Items and features

Each chunk of data is stored as an **item**.

- phones, words, syllables, pitch accents,...

Each item is described by a set of **features**.

- name, end, ...
- e.g. for a word: pos
- e.g. for a syllable: stress

CSTR

Some example items

A word item

```
name: Peter
pos: nnp
pbreak: NB
```

A segment item

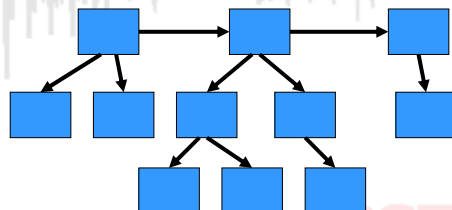
```
name: p
end: 1.2
ph_vc: -
ph_ctype: s
ph_cplace: l
```

CSTR

Items and Relations

Items don't exist on their own, but each item is part of one or more **relation**.

Relations come in 2 flavours, lists and trees.



CSTR

Some *standard* relations

Token – pre-processed input tokens

Word – actual words (e.g. eighty four)

Phrase – phrases

Syllable – syllables

Segment – phones

SylStructure* – syllabic structure

IntEvent – intonation events

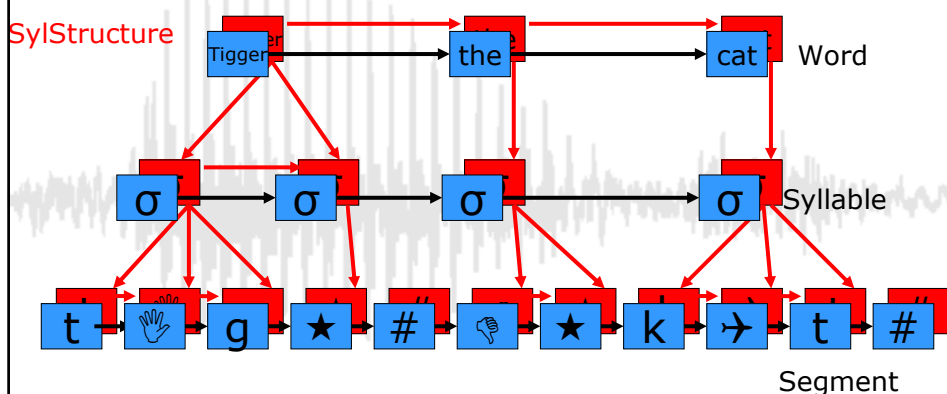
Intonation* – intonation structure

Unit – chosen unit sequence

* – tree relations

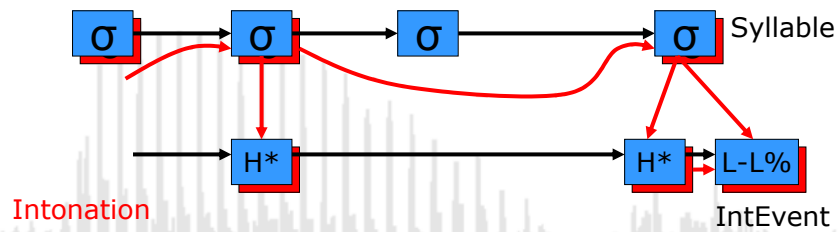
CSTR

When items are in more than one relation



CSTR

When items are in more than one relation



So a syllable item can be in up to 3 relations:

- Syllable, Sylstructure and Intonation

CSTR

Accessing utterances in scheme

```
> (set! utt (Utterance Text "Hello world."))
#<Utterance 0xa1843a0>

> (utt.synth utt)
...
#<Utterance 0xa1843a0>

> (utt.relationnames utt)
(Token Word Phrase ...)

> (set! segs (utt.relation.items utt 'Segment))
(#<item 0xb261a08> #<item 0xb247808> #<item 0xb2609d0>...)
```

CSTR

Accessing utterances in scheme

```
> (set! seg1 (car segs))
#<item 0xb261a08>
> (set! seg2 (car (cdr segs)))
#<item 0xb247808>

> (item.feats seg1 "name")
"pau"
> (item.feats seg2 "name")
"hh"
```

CSTR

Accessing utterances in scheme

```
> (utt.relation.print utt 'Segment)
()
id_17 ; name pau ; dur_factor 0 ; end 0.22 ; source_end 0.081826 ;
id_7 ; name hh ; dur_factor -0.296956 ; end 0.277954 ; source_end 0.188655 ;
id_8 ; name ax ; dur_factor -0.317324 ; end 0.320176 ; source_end 0.289519 ;
id_9 ; name l ; dur_factor 0.240634 ; end 0.399659 ; source_end 0.378457 ;
id_11 ; name ow ; dur_factor 0.0696307 ; end 0.550046 ; source_end 0.550021 ;
id_13 ; name w ; dur_factor 0.636568 ; end 0.625551 ; source_end 0.690708 ;
id_14 ; name er ; dur_factor 0.520952 ; end 0.725881 ; source_end 0.800834 ;
id_15 ; name l ; dur_factor 0.520952 ; end 0.813381 ; source_end 0.912022 ;
id_16 ; name d ; dur_factor 0.730381 ; end 0.883052 ; source_end 1.09058 ;
id_18 ; name pau ; dur_factor 0 ; end 1.10305 ; source_end 1.37287 ;
Nil
```

CSTR

Moving around a relation

```
(item.next ITEM)  
(item.prev ITEM)  
(item.parent ITEM)  
(item.daughter1 ITEM)  
(item.daughter2 ITEM)
```

```
> (item.feats (item.next seg2) "name")  
"ax"  
> (item.feats (item.next (item.next seg2))  
"name")  
"l"
```

CSTR

Moving between relations

Recall that an item can be in more than one relation.

- Any instance of an item is considered to be held with respect to a single relation at any time.
- The functions like `item.next` only allow you to move around that relation.

CSTR

This is important so lets look at it again!

- Each item can be in multiple relations
- In each relation each item has certain links to other items
 - Segment items link to other segments
 - Syllable items link to other syllables
- If you want to move from syllables to segments you need to reference with respect to the SylStructure relation
 - The parent and daughter links are only available in this relation

CSTR

Moving between relations

We can change the relation which an item is being held in reference to:

(item.relation ITEM RELATIONNAME)

And there are some short cuts for moving around:

(item.relation.next ITEM RELATIONNAME)

(item.relation.prev ITEM RELATIONNAME)

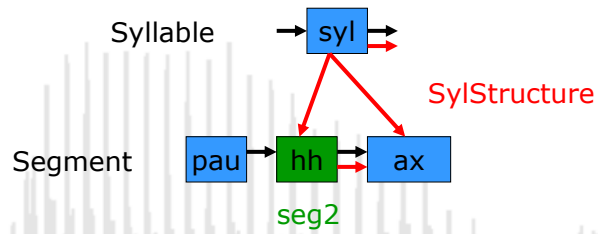
(item.relation.parent ITEM RELATIONNAME)

(item.relation.daughter1 ITEM RELATIONNAME)

(item.relation.daughtern ITEM RELATIONNAME)

CSTR

Moving between relations



[Recall seg2 is a Segment item]

```
> (item.feats (item.parent seg2) "name")
```

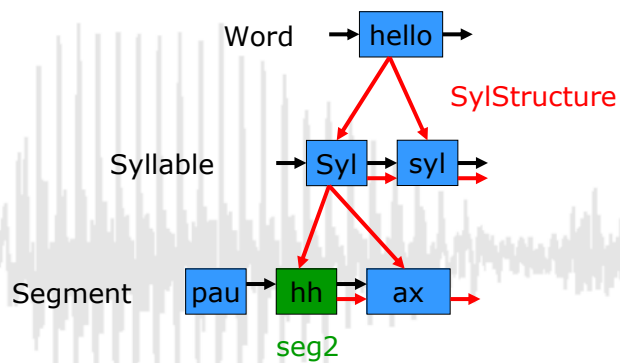
```
nil
```

```
> (item.feats (item.relation.parent seg2 'SylStructure) "name")
```

```
"syl"
```

CSTR

Moving between relations



```
> (item.feats (item.parent (item.relation.parent seg2 'SylStructure)) "name")
```

```
"hello"
```

CSTR

More on features

Features come in a number of types:

- Real features
 - Physical data in the item
- Feature functions
 - A predefined function
- User defined feature functions
 - A user defined function

A segment item

```
name: p
end: 1.2
ph_vc: -
ph_ctype: s
ph_cplace: l
lisp_myfeat: hi
```

CSTR

Feature paths

Sometimes it is possible to move around a relation using special path directives in a feature name

```
(item.featseg
  "R:SylStructure.parent.parent.name")
```

CSTR

Feature paths

R:<relationname>.	ref. wrt. this relation
parent.	
daughter1.	first daughter
daughtern.	last daughter
n.	next
p.	previous
nn.	next next
pp.	previous previous
lisp_<functionname>	user defined function

CSTR

The synthesis process

Synthesis is an 13 step process.

utt.synth (called by SayText) calls 13 functions, one after the other, on the given utterance structure.

Each function (think of them as modules) adds to the utterance in some way.

CSTR

Step 1: Initialisation

This step just does some housework to get ready for synthesis.

Step 2: Text

Parses the input text into **Tokens**

- Tokens include: "hello", "world", "123"
- Token relation is created

CSTR

Step 3: Token POS

This assigns a *part of speech* tag to tokens before they are split up.

- Mainly to deal with numbers.
- A combination of statistically trained and hand written rules.
- Tags are things like: cardinal, ordinal
- Tags are added to items in the token relation

CSTR

Step 4: Tokenisation

Converts the tokens into words

- Words are things that we can look up in a pronunciation dictionary.
- This mainly deals with working out how to say numbers and symbols
 - Nineteen eighty four vs. one thousand nine hundred and eighty four
- The default rules are specific to English, and not always perfect.
- The **Word** relation is created

CSTR

Step 5: Part of Speech

A statistical POS tagger assigns POS to each word

- Default is for English

POS is used in diphone synthesis to determine duration lengths and aids prosody generation.

POS may not be as important for unit selection

POS tags are added to items in the Word relation

CSTR

Step 6: Phrasify

Add phrase breaks

Can be either trivial rules or a statistical model

- Punctuation is the main predictor
- Some maximum number of syllables without a break.

Generally 3 levels of break are predicted:

- BB, B, NB (big break, break, no break)

Phrase relation is created and break features are added to items in the Word relation

CSTR

Step 7: Word module

This module creates the Syllable, Segment and SylStructure relations

- Word pronunciation is determined by a pronunciation lexicon and/or letter to sound rules.
- This phase can be really difficult (e.g. for English) or reasonably easy (e.g. for Spanish)

CSTR

Step 8: Pauses

Provision for pauses are made in appropriate places (at phrase breaks etc...)

- Quite trivial hand written rules
- Silence segments are inserted in the Segment relation

CSTR

Step 9: Intonation

For unit selection, nothing is currently done here!

For diphones:

- Statistical model predicts ToBI accent symbols
- **Intevent** and **Intonation** relations are created

CSTR

Step 10: Postlexical rules

A series of postlexical rules are applied

- Some hand written
 - "the" vs "thee"
- Some statistical
 - Vowel reduction

These rules usually affect the **Segment** relation, but can actually be defined to do anything

CSTR

Step 11: Segment Durations

Again, for unit selection nothing is done here.

Diphones:

- A duration for each segment is predicted.
 - Start from an average value
 - Adjust statistically (based on linguistic features)
- End times feature are added to the Segment items

CSTR

Step 12: Intonation Targets

Nothing is done for unit selection

Diphones:

- A model generates a synthetic pitch contour, resulting in a series of pitch targets for each syllable
- Creates the f0 relation

CSTR

Step 13: Waveform Synthesis

A series of diphones are selected, and joined together.

Pitch and duration modification are carried out where appropriate

A number of relations are created

- Unit: diphones for synthesis
- Wave: a single item containing a the waveform

CSTR

Unit selection speech synthesis

Record a database of speech with each diphone in many different contexts, and use find the most appropriate diphone in each case.

In fact do we even need to use diphones? Would a different unit size be better? Could we use units of variable sizes?

We shall stick with diphones for now.

CSTR

Confusing terminology...

Diphone speech synthesis: A Concatonative speech synthesis method where one example of each diphone is used to produce any speech.

This does not mean that all other speech synthesis method do not use diphones.


A unit selection speech synthesiser may use diphones. However, we tend to not call it a diphone synthesiser to avoid confusion.


CSTR

How what you record affects synthesis

400 sentence database 

2000 sentence database 

No data from the domain 

With data from the domain 

CSTR

Other linguistic structure

Phone labels are not enough, we also need to know about the context for each phone.

What we tend to do, is process the text of each utterance as we would do when synthesising it, and save the linguistic structure to provide the context.

The result is a database of linguistically annotated speech

CSTR

Using the database for synthesis

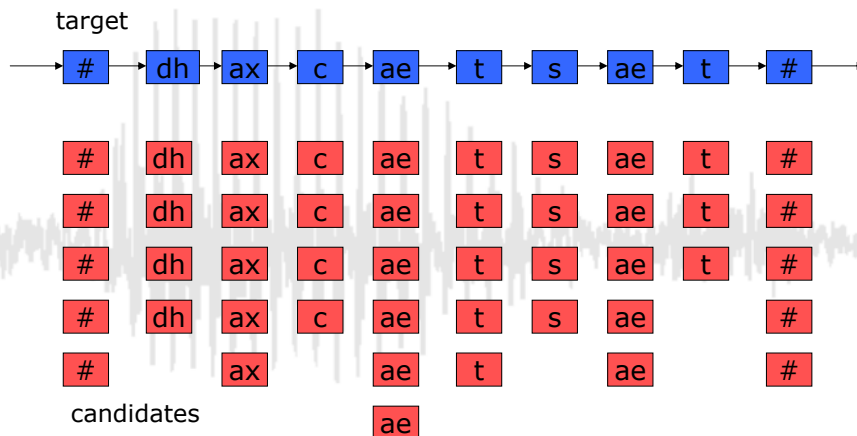
At synthesis time we are given a text and we are required to find the most suitable phone sequence from our database to concatenate together to produce suitable speech

First we carry out linguistic analysis as we would for standard diphone synthesis

The result is a suitable linguistic structure which is similar to the structure our database is annotated with.

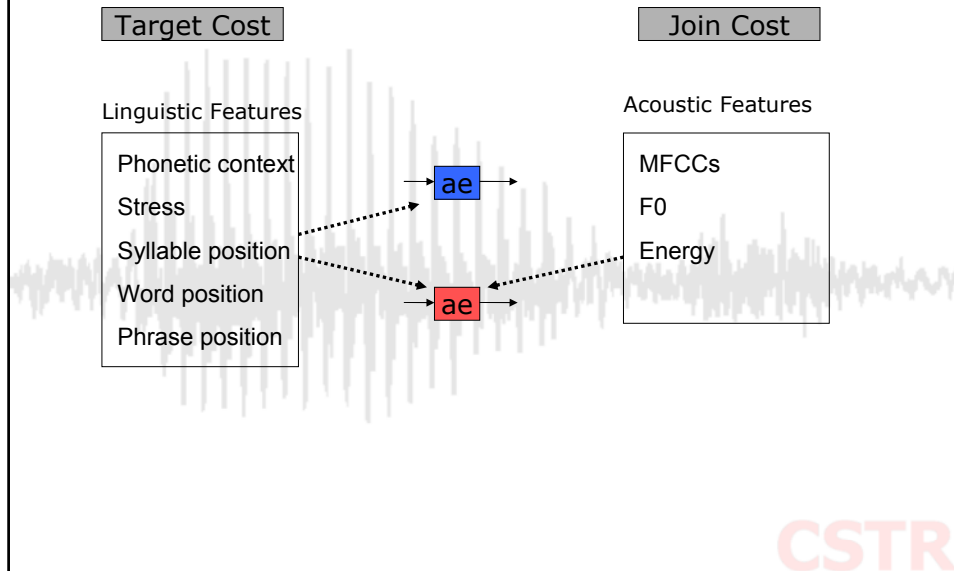
CSTR

Unit selection speech synthesis



CSTR

How do we do this?



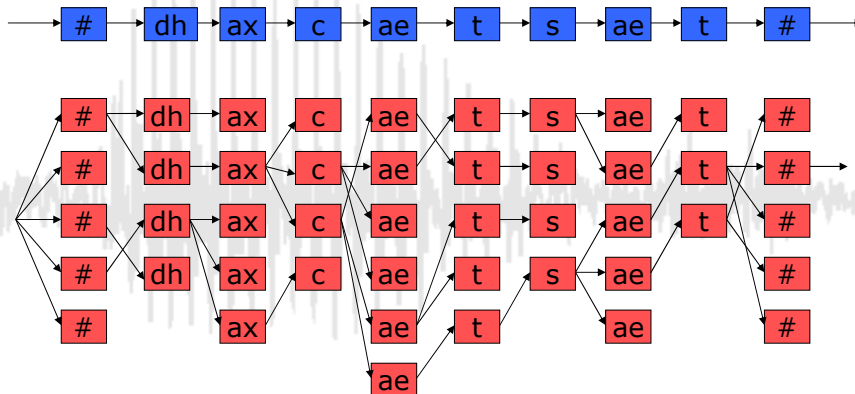
A standard search procedure

We want the sequence of candidates which minimises the cost.

You've seen this type of set up before, we use a Viterbi search to find the best sequence using the join and target costs.

We could even use token passing!

Unit selection speech synthesis



Join cost in detail

The join cost determines how well two adjacent pieces of speech join together.

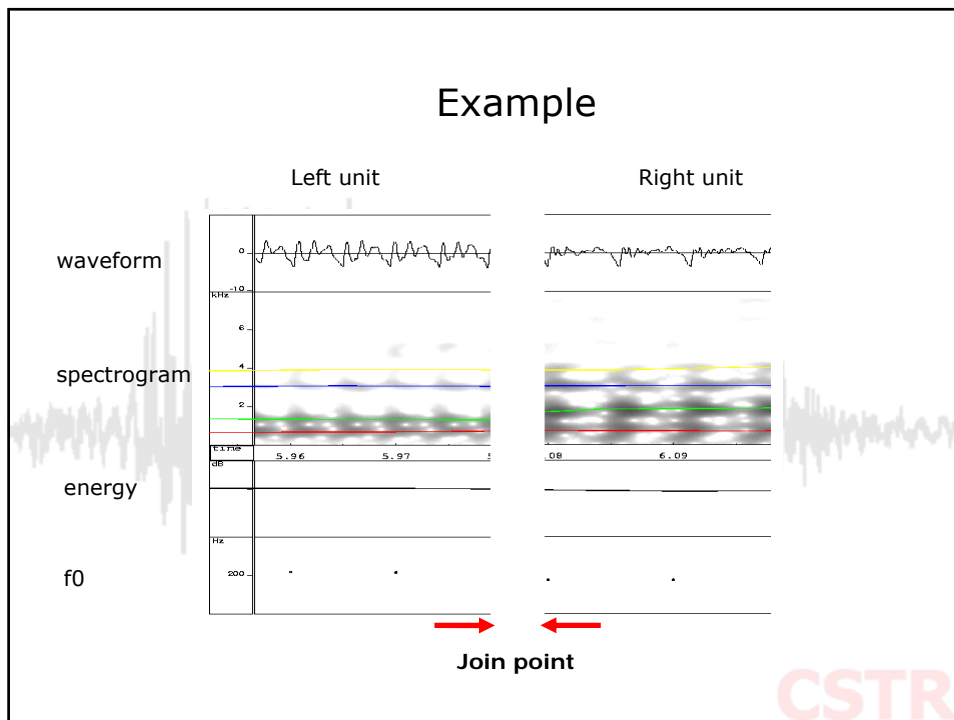
A simple join cost would locally compare:

- Spectral characteristics
- F0
- Energy

For a diphone join, the join position is supposed to be *stable*.

Units recoded sequentially in the database receive a zero join cost (this favours using whole syllables, words, sentences,... where possible).

Example



Using a join cost

Festival's *multisyn* engine simply uses normalised versions of these, weighted equally.

Designing a good join cost is an active area of research, some current ideas include:

- What is the best spectral representation to compare: MFCCs, LSFs, MCA etc... (Vepa & King)
- Data-driven (CART) Perceptually-based join costs (Syrdal & Conkie)

CSTR

Target cost in detail

The target cost is supposed to determine how well a target unit matches a given candidate unit.

How a target cost should be implemented again is an interesting research question.

- Festival's *multisyn* engine uses hand written rules that compare linguistic features of the target and candidate.
 - These rules each are assigned a weight which determines how important they are with respect to each other.

CSTR

Festival's target cost rules

Rule	Weight	Description
stress	10	Primary, secondary or no stress
syllable position	5	Position of diphone in syllable (initial, medial, final, inter)
Word position	5	Position of diphone in word (initial medial, final, inter)
Part of speech	6	noun, verb, function word etc...
Left context	7	Phone to left of diphone is?
Right context	4	Phone to right of diphone is?
Bad duration	10	Does Candidate have a spurious duration?
Bad f0	25	Does candidate have spurious F0?

Our weights are chosen heuristically, ideally we would want to train them from data.

The implementation of some of the components is complicated by the fact that we are comparing diphones rather than phones.

CSTR

Training the target cost weights

Two ways target cost weights could be trained from data:

- Use perceptual testing, train weights (or even the target cost) to match perceptual ratings
 - Requires a lot of perceptual testing
- Synthesis a held-back test-set of your database, train weights to match the acoustics of this test data.
 - Computationally expensive, but no human subjects required.

CSTR

Units other than diphones?

Smaller units:

- half-phones (AT&T, and others)

Bigger (variable sized) units:

- Prosodic structure matching (Taylor)
 - A form of pre-selection.

Units suitable for a particular language

- Moras, Syllables, etc....

CSTR

Post processing

Once we have a chosen sequence of units we need to concatenate them.

We need to decide if we should do the following:

- Spectral smoothing
- Pitch smoothing or modification
- Duration modification

CSTR