BEHAVIORAL ABSTRACTION

A sound event can behave differently according to the context in which it is instantiated.

Temporal Semantics and Behavioral Abstraction

- Extensions to ordinary (Lisp, SAL) semantics:
  - Behaviors
  - Evaluation environment
  - Transformations
  - Temporal combination: SEQ and SIM

Behaviors

- Nyquist sound expressions denote a whole class of behaviors
- The specific sound computed by the expression depends upon the *environment*
- Transformations like STRETCH and TRANSPOSE alter the behavior.
- Behaviors vs. linear transformation: when you play a longer note, you don't simply stretch the signal! The behavior concept is critical for music.

Evaluation Environment

- To implement behavior concept, *all* Nyquist expressions evaluate within an *environment.*
- Nyquist environment includes: starting time, stretch factor, transposition, legato factor, loudness, sample rates, and more.
- Environment is "hidden" and changed or accessed using special function-like constructs.

Manipulating the Environment

- Example:

```
osc(c4) ~ 3
```

- Within STRETCH, all expressions see altered environment and behave accordingly
- Scoping is *dynamic:*

```
function tone() return osc(c4)
play tone() ~ 3 → <? second sound>
```

- Transformations can be nested*:*

```
function tone() return osc(c4) ~ 2
play tone() ~ 3 → <? second sound>
```


Manipulating the Environment

- Example:

```
osc(c4) ~ 3
```

- Within STRETCH, all expressions see altered environment and behave accordingly
- Scoping is *dynamic:*

```
function tone() return osc(c4)
play tone() ~ 3 → <3 second sound>
```

- Transformations can be nested*:*

```
function tone() return osc(c4) ~ 2
play tone() ~ 3 → <6 second sound>
```


Absolute Transformations

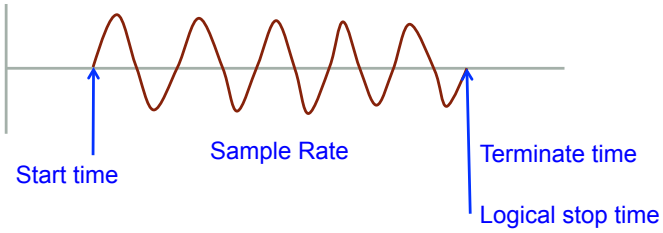
- You can override the "inherited" environment:

```
function tone2() return osc(c4) ~~ 2
play tone2() ~ 100 → <2 second tone>
```

- Even though TONE2 is called with a stretch factor of 100, its STRETCH-ABS transformation overrides the environment and sets it to 2
- *Once sound is computed by OSC(C4), the sound is immutable, i.e. not subject to transformation!!!!!*

The SOUND Type

- `osc(c4) ~~ 2` ← this is an expression
- When evaluated, osc() uses the environment (especially start time and stretch factor) and returns a SOUND:



Example

- ```
  begin
     with x = osc(c4)
     play x ~ 3
  end
  ```
  → <? second tone>
- ```
  function x() return osc(c4)
  play x() ~ 3
  ```
  → <? second tone>

Example

- ```
  begin
     with x = osc(c4)
     play x ~ 3
  end
  ```
  → <1 second tone>
- ```
  function x() return osc(c4)
  play x() ~ 3
  ```
  → <3 second tone>

Transformations

- STRETCH, STRETCH-ABS (~, ~~)
- AT, AT-ABS (@, @@)
- LOUD, LOUD-ABS
- SUSTAIN, SUSTAIN-ABS
- ABS-ENV – use default environment
- See manual for others.
- Maybe we'll talk about *time-varying* transformations later in semester.

Practical Notes

- In practice, the most critical transformations are AT (@) and STRETCH (~), which control *when* sounds are computed and how long they are.
- Technically, transformations are not functions because they do not evaluate their arguments in the normal order: instead, they manipulate the environment, evaluate the behavior, then restore the environment.
- Implemented as macros in XLISP

