Class 9: RTPL systems

- Partial evaluation vs. program construction
- List of systems we will discuss
- Implementation of DynJava
- Implementation of `C ("tick-C")"
Partial evaluation vs. program construction

Implicit p.e.

- Explicit p.e. and the "erasure property"

Program has two inputs, s and d

\[ \uparrow \quad \uparrow \]

\text{static} \quad \text{dynamic}

Given value for s, automatically

generate "residual" function of d

E.g. take generic dot-prod, provide value for v, obtain dot-by-v automatically
List of systems

- MIX (c. 1985) - late 80's (self-applicable)
- MetaML/MetaOCaml (c. 2000-present)
- Fabius (1996) - implicit
- DyC (1998)
- Tempo (1998) - uses existing C compiler
- C (1995)

- DynJava (2001)
- Jumbo (2004)
- Cyclone (2003) - a "safe" variant of C
- CodeBricks (2003) - library of AST-like operators
Implementation of RTPG

- "Template-based"
- **Language:** Use `{ ... }` for quotation, @c for antiquotation.

- Variables in fragments are declared, e.g.

```java
    code_spec<break; int x> c1 =
    ' { if (x == 5) break; }

    code_spec<> c2 =
    ' { for(int x = 0; x < 10; x++) {
        System.out.print(" "+x);
        @c1;
    }
    };
```
DynJava implementation

- Each fragment becomes a function, e.g.

```java
- c1: void t1()
    int x;
    while (true) {
        if (x == 5) break;
    }

- c2: t2()
    for --
        t1();
    }
```