Compiler Construction

Overloading

Overloading

Overloading: Homonyms

Several entities bearing the same name, but statically distinguishable, e.g., by their arity, type etc.

Aliasing: Synonyms

One entity bearing several names.

```
// foo is overloaded.
int foo(int);
int foo(float);

// x and y are aliases.
int x;
int& y = x;
```

Operator Overloading

Overloading is meant to simplify the user's life. Since FORTRAN!

No Overloading in Caml (but type inference)

```
# 1.0 + 2.0;;
Characters 0-3:
  1.0 + 2.0;;
  ^^^
This expression has type float
but is here used with type int
```

Overloading in C

```
int   a = 1   + 2;;
float b = 1.0 + 2.0;;
```

Function Overloading

Usually based on the arguments

Ada

```
I :  INTEGER;
X :  REAL;
...
PUT("results are: ");
PUT(I);
PUT(X);
```

Overloading is Syntactic Sugar

Overloaded

```
void foo(int);
void foo(char);
void foo(const char*);
void foo(std::string);

int main ()
{
  foo(0);
  foo('0');
  foo("0");
  foo(std::string("0"));
}
```

Overloading is Syntactic Sugar

Overloaded

```
void foo(int);
void foo(char);
void foo(const char*);
void foo(std::string);

int main ()
{
  foo(0);
  foo('0');
  foo("0");
  foo(std::string("0"));
}
```

Un-overloaded

```
void foo_int(int);
void foo_char(char);
void foo_char_p(const char*);
void foo_std_string(std::string);

int main ()
{
  foo_int(0);
  foo_char('0');
  foo_char_p("0");
  foo_std_string(std::string("0"));
}
```

Overloading is Syntactic Sugar

Usually solved by renaming/mangling.

g++-2.95, como

```
f__Fi  -> int f(int);
f__FPc -> int f(char*);
```

g++-3.2, icc

```
_Z1fi  -> int f(int);
_Z1fPc -> int f(char*);
```

Overloading in Tiger

Ordering <, <=, >, and >=
overloaded for pairs of integers, or strings.

Identity = and <>
overloaded for (type coherent) pairs of integers, strings, arrays or records.

Summary