Introduction to SML
Summary on Functions

Anne E. Haxthausen

ah@imm.dtu.dk

Computer Science and Engineering

Informatics and Mathematical Modelling

Technical University of Denmark

Syntactic issues

Function types: ty_1 `->` ty_2

f : ty_1->ty_2 is of *higher-order*, if ty_1 or ty_2 contain a function type.

Function expressions:

- lambda expressions `fn` pat_1 `=>` exp_1 | ... | pat_n `=>` exp_n
- f, where f is the id of a declared function (e.g. `fact`)
- `op` op, where op is an operator (e.g. `op +`)
- function compositions exp_1 `o` exp_2
- ...

Declaration of functions f:

`fun` f $pat_{11} \ldots pat_{1n}$ `=>` exp_1

$\vdots$

| f $pat_{k1} \ldots pat_{kn}$ `=>` exp_n

Semantic issues

SML has *static binding* of identifiers.

Function expressions evaluate to *closures*.

When functions are applied, their bodies are evaluated in the environment found when the function was defined (overwritten with bindings of formal parameters to actual values).

Pragmatic issues

Higher-order functions is a powerful abstraction mechanism:

Assume given a family of functions f_i that have a common pattern and individual specifics that can be described by functions $g_{i1}, \ldots, g_{in}$. Then the f_i's can be obtained by applying an appropriate higher-order function F to $g_{i1}, \ldots, g_{in}$: $f_i = F(g_{i1}, \ldots, g_{in})$

A collection of useful predefined higher-order functions on lists have been given: `map, exists, ...`