CSC 372, Spring 2025

SML and Types continued
Syntax and Semantics

Michelle Strout

THE UNIVERSITY OF ARIZONA

January 28, 2025

Plan

THE UNIVERSITY OF ARIZONA
Computer Science

- **Announcements**
  - SA2 is due tomorrow, Jan 29th, you will be writing 10 SML functions
  - Dr. Strout virtual office hour 11:30-12:30 on Wednesdays, see Piazza
  - Simon Peyton Jones will be talking with class on February 13th
- **Last time**
  - Review some pre-assessment quiz questions
  - ICA3: Participation Quiz
  - SML intro continued
- **Today**
  - Review previous quiz questions
  - SML intro continued and Types intro
  - Syntax and Semantics

TopHat Questions

- **ICA3:**
  - Go to gradescope to see how you did, see piazza announcement.
  - The class median on the 10 questions was 6.25/10.
- **Spaced repetition question**
  - Three people out of 54 answered this correctly.
- **Link languages to motivation in TopHat**
  - Python
  - Ruby
  - Swift
  - Rust
  - Perl

How should I study for 372?

- **Gather and create example questions**
  - Questions from class slides, TopHat, quizzes.
  - Recall practice questions at end of Ray Toal readings.
  - Create questions and answers about all the concepts that are in the class slides.
  - Create questions and answers about the code you are writing.
- **Dig in and work on understanding and learning**
  - Collaborate with others and AIs to create questions and formulate answers.
  - Verify things you are unsure about by looking for alternative sources of information on the web such as books, other course material, or asking on piazza.
- **Use spaced repetition to study**

Outline for today

- Unit testing and exceptions in SML (Hands On)
- Other things to try in SML
- Intro to Types: motivation and terminology
- Syntax and Semantics for specifying PLs
- Compiler/Interpreter Overview
- Tokens

Testing your SML functions

- **See sml-intro-in-class.sml**
  - Uncomment the 'use "Unit.sml"; ' code
  - Can check expected results and if an exception has occurred
- **Code: Some example usage**

```
val () =
  Unit.checkExnWith Int.toString
  "minlist [] should raise an exception"
  (fn () => minlist [])

val () =
  Unit.checkExpectWith
  (Unit.listString (Unit.pairString Int.toString Int.toString))
  "zip ([],[]) should be []"
  (fn () => zip ([],[]))
  []
```

Exceptions Example in SML

THE UNIVERSITY OF ARIZONA
Computer Science

- **Code**

```
exception ListTooShort

fun drop 0 lst = lst
  | drop n [] = raise ListTooShort
  | drop n (x::xs) = drop (n-1) xs

val res7 = drop 2 [1,2,3,4]
val res8 = drop 3 [10, 20, 30]
           handle ListTooShort =>
              (print "List too short!"; [])
```

- **Questions (TopHat)**
  - What is res7 going to be?
  - What is res8 going to be?

Other things to try

- **In the poly REPL, try the following:**

```
let x=3 and y=4 in x+y end; (* poly REPL balks, why? *)
real;
explode;
ord;
trunc;
floor;
ceil;
round
chr;
str;
(op +);
```

- **Questions for you to answer**
  - What do each of the above do?
  - Ask an AI how to fix the error you get for the first one.

Why Types Matter

- **Motivation for Type Checking**
  - Types catch errors at compile-time, reducing runtime bugs.
  - Improves program readability and maintainability.
- **Examples of mistakes Types can prevent**
  - Passing a string where a number is expected.
  - Misusing functions or operators.
- **Functional programming context**
  - Common in languages like SML, Haskell, and OCaml

Collab Reference: This slide and next 4 slides generated with help of ChatGPT on Jan 23, 2025

Key Terminology

- **Strong vs. Weak Typing**
  - Strong: Enforces strict type rules (e.g., SML, Haskell).
  - Weak: Allows implicit type conversions (e.g., JavaScript).
- **Static vs. Dynamic Typing**
  - Static: Types are checked at compile-time (e.g., SML).
  - Dynamic: Types are checked at runtime (e.g., Python).
- **Inferred vs. Annotated Types**
  - Inferred: Types deduced by the compiler or runtime (e.g., SML, Python).
  - Annotated: Types explicitly specified by the programmer (e.g., Java, C, C++, Chapel)

Void and Unit Types

THE UNIVERSITY OF ARIZONA
Computer Science

- **Void Types**
  - Represents "no value" or "nothing".
  - Often used in imperative languages (e.g., `void` in C).

- **Unit Type in Functional Languages**
  - Represents a single value: '( )'.
  - Commonly used where a value is required but irrelevant.

```
fun printHello () = print "Hello"
```

What’s Coming in Future Classes

- **Polymorphism**
  - Parametric polymorphism (e.g., generics).
  - Ad-hoc polymorphism (e.g., function/method overloading).
- **Classes and Types**
  - Object-oriented concepts like classes and interfaces.
  - Runtime polymorphism through mechanisms like method overriding and dynamic dispatch
- **Advanced Topics**
  - Dependent typing, gradual typing, and typeclasses.
  - Design choices in string and array types.
  - Mixins, protocols, and type erasure.

Summary and Takeaways on Type Intro

- **Types protect programmers**
  - Compile-time checks catch many errors early.
- **Understanding key concepts**
  - Strong vs. weak
  - Static vs. dynamic
  - Inferred vs. annotated
- **Preview of advanced topics**
  - Future classes will cover polymorphism, type inference, some type theory, and type choices.

Outline for today

- Unit testing and exceptions in SML (Hands On)
- Other things to try in SML
- Intro to Types: motivation and terminology
- Syntax and Semantics for specifying PLs
- Compiler/Interpreter Overview
- Tokens

Syntax and Semantics

THE UNIVERSITY OF ARIZONA
Computer Science

- **What are syntax and semantics?**
  - **Syntax refers to the structure of a language. The rules that define how symbols can be combined to form valid statements or expressions.**
  - **Semantics refers to the meaning of a language. The interpretation or behavior associated with syntactically correct statements.**

- **They are interconnected:** The **meaning** (**semantics**) of a sentence depends on its **structure** (**syntax**).

- **Examples**
  - **Syntactically valid, but semantically invalid:**

```
int x = "Hello";
```

  - **Semantically ambiguous: "I saw the man with the telescope."**

Examples of Applications to PL Research

- **Program Synthesis**

From "Synthesizing Parallel Graph Programs via Automated Planning," by Prountzos, Manevich, and Pingali, PLDI 2015.

Meta Variable	Description
x	A program variable $x \in \mathit{Var}$
b	Boolean expression
r	Data range expression
rs	A sequence of range expressions
upd	State update

Attribute	Value Type (inherited/synthesized)
$bnd(\cdot)$	2^{Var} (inherited)
$vars(\cdot)$	2^{Var} (synthesized)

Production	Semantic Rules
$S ::= \textbf{for } x : r \textbf{ do}^L\ B_1 \textbf{ od}^L$	if $x \in bnd(S)$ then error, if $vars(r) \not\subseteq bnd(S)$ then error, $bnd(B_1) = bnd(S) \cup \{x\}$.
$\mid \textbf{while } b \textbf{ do}^L\ B_2 \textbf{ od}^L$	if $vars(b) \not\subseteq bnd(S)$ then error, $bnd(B_2) = bnd(S)$.
$\mid \textbf{if } b^L\ B_t \textbf{ else}^L \textbf{ skip}^L \textbf{ fi}^L$	if $vars(b) \not\subseteq bnd(S)$ then error, $bnd(B_t) = bnd(S)$.
$B ::= S$	$bnd(S) = bnd(B)$.
$\mid A$	$bnd(A) = bnd(B)$.
$A ::= \mathit{AtomUpd}$	$bnd(\mathit{AtomUpd}) = bnd(A)$.
$\mid \textbf{acq } rs_1 \textbf{ ctx } rs_2^L ; A_1$	if $vars(rs_1, rs_2) \not\subseteq bnd(A)$ then error. $bnd(A_1) = bnd(A)$.
$\mid \textbf{if } b^L\ A_t \textbf{ else}^L\ R \textbf{ exit}^L \textbf{ fi}^L$	if $vars(b) \not\subseteq bnd(A)$ then error, $bnd(A_t) = bnd(R) = bnd(A)$.
$\mid \textbf{for } x : r \textbf{ do}^L\ A_b \textbf{ od}^L$	if $\neg val(r)$ then error, if $vars(r) \not\subseteq bnd(A)$ then error, if $x \in bnd(A)$ then error, $bnd(A_b) = bnd(A) \cup \{x\}$.
$R ::= \epsilon$	
$\mid \textbf{rel } rs^L$	if $vars(rs) \not\subseteq bnd(R)$ then error.

- **Proving Security Properties**

Theorem 4.3 (View Non-Interference). *Consider a sensitive value $V = \langle E_l | H \rangle_\ell$ in a Jeeves expression E. Assume:*

$$E[H \mapsto E_h] \hookrightarrow e \qquad \vdash \langle \emptyset, \emptyset, e \rangle \to^* \langle \Sigma, \Delta, \sigma \rangle$$
$$E[H \mapsto E_h'] \hookrightarrow e' \qquad \vdash \langle \emptyset, \emptyset, e' \rangle \to^* \langle \Sigma', \Delta', \sigma' \rangle$$

For any context value v, if

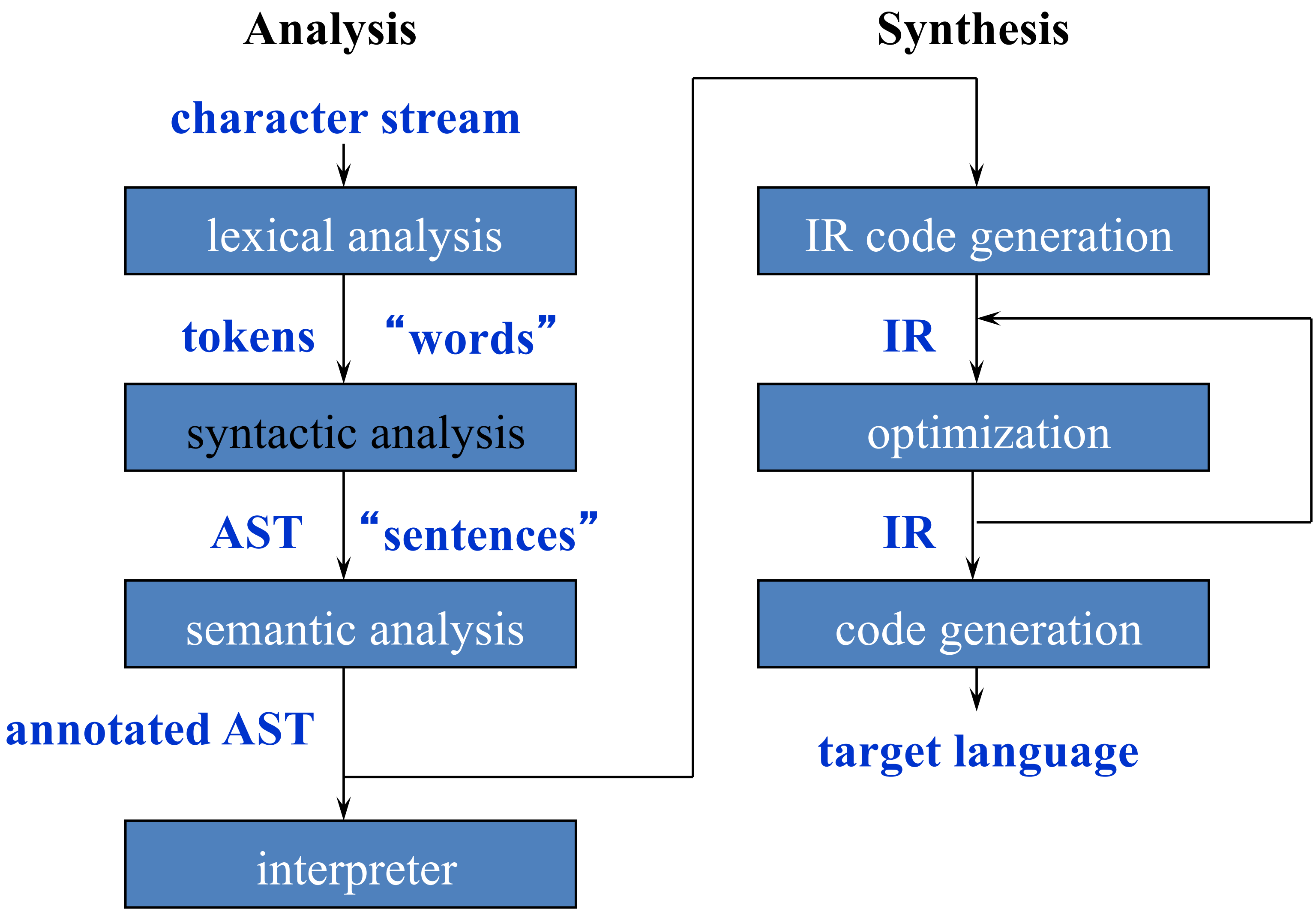
$$\Sigma \cup \{\textbf{context} = v\} \vdash \ell = \bot$$
$$\Sigma' \cup \{\textbf{context} = v\} \vdash \ell = \bot$$

then

$$\{c \mid \ \vdash \langle \Sigma, \Delta, \textbf{concretize } \sigma \textbf{ with } v \rangle \to \langle \Sigma_0, \Delta_0, c \rangle\} =$$
$$\{c' \mid \ \vdash \langle \Sigma', \Delta', \textbf{concretize } \sigma' \textbf{ with } v \rangle \to \langle \Sigma_0', \Delta_0', c' \rangle\}$$

From "A Language for Automatically Enforcing Privacy Policies," by Yanh, Kuat, and Solar-Lezama, POPL 2012.

Structure of a Typical Compiler



Activity

THE UNIVERSITY OF ARIZONA
Computer Science

- **Consider Java. Write an example of:**
  - A syntactically correct statement.
  - A syntactically incorrect statement.
  - A semantically invalid but syntactically correct statement.

Grammar and Syntax Rules

THE UNIVERSITY OF ARIZONA
Computer Science

- **What is a Grammar?**

- **A grammar defines a language by specifying:**
  - Tokens - The smallest units of a language (e.g., keywords, literals).
  - Rules - How tokens combine into valid statements.

- **Backus-Naur Form (BNF):**
  - Developed by John Backus and Peter Naur (~1960).
  - Defines structure using:
    - Terminals: Basic symbols (tokens).
    - Non-terminals: Higher-level constructs made from terminals.

Example formal grammar in SML

- **Syntax rules**

```
<expr> ::= <value> | <expr> <op> <expr>
<value> ::= int | bool
<op> ::= + | * | and | or
```

- **Semantic rules**
  - Integers can be added or multiplied.
  - Booleans can be combined using logical operators.

Learning by doing

- Large assignment 1 is a compiler from shapes to svg

```
CIRCLE 120 150 60 white
```

```
<circle cx="120" cy="150" r="60" fill="white" />
```

- Going to specify the input with a context free grammar and tokens with regular expressions

- Going to implement a lexer, parser, AST (abstract syntax tree), and "codegen" in SML

Identifying Tokens Exercises

THE UNIVERSITY OF ARIZONA
Computer Science

- **At your tables, list out the different tokens in the below "shapes" file using the provided SML datatype**

```
CIRCLE 120 150 60 white
LINE 0 0 300 300 black
RECTANGLE 30 20 300 250 blue
```

```
(* Token datatype *)
datatype token =
    TokenCIRCLE
  | TokenLINE
  | TokenRECTANGLE
  | TokenNUM of int
  | TokenCOLOR of string
```