

Comp 524 - Assignment 4: Social Distancing in SML

Date Assigned: 10/14/21

Early Completion Date: Friday, 10/29/21

Completion Date: Tuesday, 11/2/21

As in the Prolog assignment, this assignment has two main parts. The first one acquaints you with the SML interpreter and how to run local tests on SML code you have written. In the second you will implement the social distancing application with and without lists. Because both SML and Prolog have lists and pattern matching, many aspects of the solution will be similar; so your Prolog solution of A3 will probably be a better model for this assignment than the Java solution of A1.

The assignment exercises the following ML concepts: (1) functional programming and pattern matching, (2) lists and recursion, (3) function parameters, and (4) function returns. No matter when you start this assignment, (1) would have been covered in class.

An excellent reference to SML syntax is [here](#). **You are restricted to constructs identified in this course.** (The [SML/NJ organization's website](#) provides more information than you'll ever need to know, if you're curious.)

The dates will be moved if we do not cover enough in the Tuesday class.

Syntax Peculiarities

1. A semicolon is an expression separator not an expression terminator.
2. Function application is a unary operator and has higher precedence than binary operators.

Please expand on this list as you find your own gotchas.

Install SML

Install an SML interpreter (and compiler) on your computer based on the reference material found [here](#).

SML Greeting

Create an SML project/folder with the sml file **Greeting.sml**.

Add the following text to this file, which defines the **greeting** function:

```
fun greeting() = print ("hello\ngoodbye\n");
```

Ask the interpreter to load this file using the command line, Eclipse SML plugin or some other system.

Once the file has been loaded, enter the following two lines on the console to execute the function and exit from the interpreter:

```
greeting();  
OS.Process.exit(OS.Process.success);
```

Verify that strings **hello** and **goodbye** are output in response to the function call.

SML LocalChecks (Optional)

Verify that you can execute **sml** from the command line, that is, it is in your PATH.

Again, in the Java project you created, create a runner for the SML project along the lines of the runner for the Prolog project. The assignment suite this time is:

```
gradingTools.comp524f21.assignment4.F21Assignment4Suite.
```

You need to include your sml files in a folder names src (see the end of the document for an example of what your file structure should look like)

You have to take one additional step if you are a Mac or Unix/Linux user. Before calling the main method of the assignment suite (not before the declaration of the runner main), call **smlIsBatFile(false)** on that suite, as shown below:

```
F21Assignment4Suite.smlIsBatFile(false); // Mac and Linux
```

Execute the runner and then invoke the only check in the suite.

On Macs, this may not work for the same reason it may not work for Prolog. **In that case, delete the call above.**

```
F21Assignment4Suite.smlIsBatFile(false);
```

Instead, take an action that is similar to the one you took for Prolog.

On the command line, execute, **which sml** to get the full name of sml.

Import the following class in the runner:

```
import grader.basics.execution.sml.SMLCommandGeneratorSelector;
```

Now in your runner main method, before you execute the main method of the suite execute the following call, passing to setUserBinary the full name of **sml**:

```
SMLCommandGeneratorSelector.getCommandGenerator().  
    setUserBinary(<full name of sml>);
```

Functional Programming and Pattern Matching

givenSafe

Implement a function named **givenSafe** that takes as parameters a **feature tuple**. A feature tuple is a 3-element tuple containing the distance, duration, and exhalation level, in that order. The function returns true if the combination represented by its argument is safe based on the tables 2 we saw in earlier assignments, reproduced below:

	Small	Medium	Large
Distance	6	13	27
Duration	15	30	120
Exhalation Level	10	30	50

Table 1 Small, Medium and Large Values

Distance	Duration	Exhalation Level
Medium	Medium	Medium
Small	Medium	Small
Large	Medium	Large
Medium	Small	Large
Medium	Large	Small
Large	Large	Medium
Small	Small	Medium

Table 2 Assumed Given Safe Combinations

Use pattern matching to write this function, which means you will have magic numbers. The implementation should have no if expressions. The following interaction shows its behavior.

```
- givenSafe;  
val it = fn : int * int * int -> bool  
- givenSafe (13, 30, 30);  
val it = true : bool  
- givenSafe(13, 29, 29);  
val it = false : bool
```

interpolatedSafe

Implement a function named **interpolatedSafe** that, like **givenSafe**, takes as a parameter a feature tuple. It returns the same Boolean value as your corresponding Java function calls and Prolog queries would:

```
- interpolatedSafe;  
val it = fn : int * int * int -> bool  
- interpolatedSafe (13, 29, 29);  
val it = true : bool  
- interpolatedSafe (12, 29, 29);  
val it = false : bool  
- interpolatedSafe(28, 30, 30);  
val it = false : bool
```

You can use the `maxInt` value given by SML (`valOf Int.maxInt`) or 200 as the max value.

Lists and Recursion

SAFETY_TABLE

Define a global variable, **SAFETY_TABLE** that captures the values in Table 2 as a list of tuples:

```
val SAFETY_TABLE =  
  [(13,30,30),(6,30,10),(27,30,50),(13,15,50),(13,120,10),(27,120,30),  
   (6,15,30)] : (int * int * int) list
```

listDerivedSafe

Implement a function named **listDerivedSafe** that, like **interpolatedSafe** and **givenSafe**, takes as a parameter a feature tuple. It uses the safety table above to return the same Boolean value as your corresponding Java function calls and Prolog queries would:

```
- listDerivedSafe;  
val it = fn : int * int * int -> bool  
- listDerivedSafe (13, 29, 29);  
val it = true : bool  
- listDerivedSafe (13, 31, 31);  
val it = false : bool  
listDerivedSafe(28, 30, 30);  
val it = true : bool
```

Function Parameters

printSafety

Implement a function, **printSafety**, that takes as an argument a 2-element tuple. The first element of the tuple is a **safety computer** function, and the second element is a feature tuple. The safety computer has the same signature as **interpolatedSafe** and **listDerivedSafe**. The function **printSafety** calls the safety computer with the feature tuple to determine the safety of the feature tuple and then uses the format illustrated below to print the components of the feature tuple and the calculated safety:

```
- printSafety;  
val it = fn : (int * int * int -> bool) * (int * int * int) -> unit  
  
- val testFeatures = (28, 29, 29);  
val testFeatures = (28,29,29) : int * int * int  
  
- printSafety (interpolatedSafe, testFeatures);  
Distance:28 Duration:29 Exhalation:29 Safe:false  
val it = () : unit  
  
- printSafety (listDerivedSafe, testFeatures);  
Distance:28 Duration:29 Exhalation:29 Safe:true  
val it = () : unit
```

concisePrintSafety

concisePrintSafety is just like **printSafety** except it uses a different print format illustrated below:

```
- concisePrintSafety;  
val it = fn : (int * int * int -> bool) * (int * int * int) -> unit  
  
- val testFeatures = (28, 29, 29);  
val testFeatures = (28,29,29) : int * int * int  
  
- concisePrintSafety (interpolatedSafe, testFeatures);  
(28, 29, 29, false)  
val it = () : unit  
  
- concisePrintSafety (listDerivedSafe, testFeatures);  
(28, 29, 29, true)  
val it = () : unit
```

listPrintSafety

Implement a function, **listPrintSafety**, that takes as an argument a 3-element tuple. The first element of the tuple is a **print safety** function, the second element is a **safety computer** function, and the third element is a **feature list**.

The **printSafety** function argument has the same signature as the **printSafety** and **concisePrintSafety** functions above.

The safety computer has the same signature as the corresponding argument of the **printSafety** and **concisePrintSafety** functions above, that is, it has the same signature as **givenSafe**, **interpolatedSafe** and **listDerivedSafe**.

The feature list is a list of feature tuples.

The function uses the print safety and safety computer arguments to print the safety of each element of the feature list. In other words, whether each element is safe is determined by the safety computer argument and how the tuple and its safety are displayed is determined by the print safety argument, as shown below:

```
- listPrintSafety;
val it = fn : ('a * 'b -> 'c) * 'a * 'b list -> unit

- val testFeatures = [(13, 30, 30), (13, 29, 29), (13, 31, 31), (28, 29, 29)];
val testFeatures = [(13,30,30),(13,29,29),(13,31,31),(28,29,29)]
: (int * int * int) list

- listPrintSafety (printSafety, interpolatedSafe, testFeatures);
Distance:13 Duration:30 Exhalation:30 Safe:true
Distance:13 Duration:29 Exhalation:29 Safe:true
Distance:13 Duration:31 Exhalation:31 Safe:false
Distance:28 Duration:29 Exhalation:29 Safe:false
val it = () : unit

- listPrintSafety (concisePrintSafety, interpolatedSafe, testFeatures);
(13, 30, 30, true)
(13, 29, 29, true)
(13, 31, 31, false)
(28, 29, 29, false)
val it = () : unit

- listPrintSafety (printSafety, listDerivedSafe, testFeatures);
Distance:13 Duration:30 Exhalation:30 Safe:true
Distance:13 Duration:29 Exhalation:29 Safe:true
Distance:13 Duration:31 Exhalation:31 Safe:false
```

```
Distance:28 Duration:29 Exhalation:29 Safe:true
val it = () : unit
```

```
- listPrintSafety (concisePrintSafety, listDerivedSafe, testFeatures);
(13, 30, 30, true)
(13, 29, 29, true)
(13, 31, 31, false)
(28, 29, 29, true)
val it = () : unit
```

`matchingSafe`, `matchingDerivedSafe`, `derivedSafeMatcher`, `matchingGivenSafe` and `givenSafeMatcher`

Using the implementation of **listDerivedSafe** as a model, write a function, **matchingSafe** that takes as a parameter a 2-element tuple. The first element of the tuple is a **matcher function**, and the second element is a feature tuple.

The matcher function takes as an argument a tuple consisting of two feature subtuples and returns a Boolean. The first argument **matches** the second argument if the function returns true.

The `matchingSafe` function returns true if its feature tuple argument matches any of the feature tuples in `SAFETY_TABLE`. It uses its matcher function to determine if the match occurs.

Implement a function named **matchingDerivedSafe** that has the same behavior as **listDerivedSafe** except that it computes safety by calling **matchingSafe** with a matcher function implemented by you named **derivedSafeMatcher**

The signatures of these two functions are given below:

```
val derivedSafeMatcher = fn : (int * int * int) * (int * int * int) -> bool
val matchingDerivedSafe = fn : int * int * int -> bool
```

Implement a function named **matchingGivenSafe** which has the same behavior as `givenSafe` except that it calls `matchingSafe` with a matcher function implemented named **givenSafeMatcher**.

The signatures of these two functions are given below:

```
val givenSafeMatcher = fn : ("a * "b * "c) * ("a * "b * "c) -> bool
val matchingGivenSafe = fn : int * int * int -> bool
```

(Why are the signatures of `givenSafeMatcher` and `derivedSafeMatcher` different?)

Function Return Values

`curryableInterpolatedSafe`, `curriedOnceInterpolatedSafe`, `curriedTwiceInterpolatedSafe`

Implement a function named **curryableInterpolatedSafe** which is a curryable version of `interpolatedSafe`:

```
- curryableInterpolatedSafe;  
val it = fn : int -> int -> int -> bool  
- curryableInterpolatedSafe 13 30 30;  
val it = true : bool  
- curryableInterpolatedSafe 13 29 29;  
val it = true : bool
```

Implement a function named **curriedOnceInterpolatedSafe** that is generated from **curryableInterpolatedSafe** by fixing the distance argument of the latter to medium distance:

```
- curriedOnceInterpolatedSafe;  
val it = fn : int -> int -> bool  
- curriedOnceInterpolatedSafe 30 30;  
val it = true : bool  
- curriedOnceInterpolatedSafe 29 29;  
val it = true : bool  
- curriedOnceInterpolatedSafe 31 31;  
val it = false : bool
```

Implement a function named **curriedTwiceInterpolatedSafe** that is generated from **curriedOnceInterpolatedSafe** by fixing the first duration argument of the latter to medium duration:

```
- curriedTwiceInterpolatedSafe;  
val it = fn : int -> bool  
- curriedTwiceInterpolatedSafe 29;  
val it = true : bool  
- curriedTwiceInterpolatedSafe 31;  
val it = false : bool
```

`curryableMatchingSafe`, `curriedMatchingDerivedSafe`, `curriedMatchingGivenSafe`

Implement a curryable version of **matchingSafe** called **curryableMatchingSafe** .

Implement **curriedMatchingDerivedSafe** by generating from **curryableMatchingSafe** by fixing the first matcher argument to **derivedSafeMatcher**.

Implement **curriedMatchingGivenSafe** by generating from **curryableMatchingSafe** by fixing the first matcher argument to givenSafeMatcher .

For fun and learning, no grading


















For fun, implement curryable versions of printSafety and listPrintSafety and generate from listPrintSafety a function for each combination of print safety and safety computer functions. This part will not be graded.

Style Checks

In order for style checks to be run through local checks a few additional set up steps are needed: First in your java local checks project directory add a folder named config and in that folder either clone or download this git repository:

<https://github.com/benknoble/smlnj-parser-style.git>

With this in place your project directory should look like this:

- ▼  Assignment
 - >  Logs
 - ▼  src
 -  Greeting.sml
 -  SocialDistance.sml
- ▼  Assignment Grader
 - >  bin
 - ▼  config
 - ▼  smlnj-parser-style
 - >  524-f20-a3
 - >  style-utils
 -  524-f20-a3.cm
 -  README.md
 - >  Logs
 - ▼  src
 - ▼  a4
 -  A4LocalProjectGrader.java

Submission Material

As always, submit the entire project folder to both **Sakai** and **Gradescope**; include **png** files with **localchecks** result; and include (**.png**) one or more screenshots demonstrating the functionality you have implemented.