Motivation

- Manual unit conversion in the code can be cumbersome and error-prone
- Problem in the history: Loss of Mars Climate Orbiter - A catastrophic industrial accident due to unit problems
- Our goal: a general purpose language with user-friendly unit features
Language Overview

**Static**
- Statically typed
- Statically scoped
- Data type and unit explicitly specified

**Types**
- int, float, boolean, string, void
- int array, float array

**C-like Syntax**
- Syntactically mimic C with new data type and unit syntax

**Unit**
- Supports variables with units, Build-in base and user-defined units, automatic unit conversion, supported units: m, cm, s, kg, m/s ...
2nd Overview
Compiler Architecture
Features
Demo
Teamwork
Compiler Architecture

Focus on the normal semantic check like type check and expr validation. Walk over AST and generate SAST.

*An additional layer dedicated to check units for variables, exprs and functions; modify sexprs to do unit conversion, no change in type.
A SCIC program is composed of:

- Unit declarations
- Global variable declarations
- Function declarations

Declared units are global.

Inside function declarations, we can define local variables (must initialize).

Static type and static unit:

Must specify legal type and unit (or no-unit) upon variable/function declaration.

Only float and float array has unit, other types (bool, string, etc) no-unit.

Support float array with static unit:

Every element in one array has the same unit.

Support for, while, if-else stmts.

```
int func main()
{
    int i = 0;
    float[] 'm' dx = [2.3, 4.5, 3.4, 0.7];
    float[] 's' dt = [0.5, 0.2, 1.7, 0.5];
    float[] 'm/s' res = [0.0, 0.0, 0.0, 0.0];
    for (; i < 4; i = i + 1) {
        res[i] = dx[i] / dt[i];
        printf(res[i]);
    }
    return 0;
}
```
Flexible local variables

Unit System

\[
\text{'km} = 0.001 \text{'m};
\]

float \text{'cm} x = 30.0;

Local Declare-and-assign

Flexible local variables

Array Support:

Support array and operations
Base units: S.I. Unit

<table>
<thead>
<tr>
<th>S.I. Unit</th>
<th>Name</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>kilogram</td>
<td>kg</td>
</tr>
<tr>
<td></td>
<td>metre</td>
<td>m</td>
</tr>
<tr>
<td></td>
<td>second</td>
<td>s</td>
</tr>
<tr>
<td></td>
<td>ampere</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>kelvin</td>
<td>K</td>
</tr>
<tr>
<td></td>
<td>mole</td>
<td>mol</td>
</tr>
<tr>
<td></td>
<td>candela</td>
<td>cd</td>
</tr>
</tbody>
</table>

User defined unit is a scaling of an existing unit. The scale is a float number.

```c
'|{us} = 1.0E+6 \{s}\};
float '{us} ua;
'|{mm} = 1000.0 \{m}\};
'|{km} = 0.001 \{m}\};
```

Derived unit (a simple expr of existing units) can be used directly.

```c
'|{g} = 1000.0 \{kg}\};
float '{m*g} x;
```

Non-base units:
centimeter: cm
Others leave to the users

The conversion to target unit is automatic!
(Here prints 0.998)
Unit System: Unit declaration

- Only float and float array have units. Unit default to none for other types (bool, int, string ...). If a float is defined without '{unit}', the compiler interpret it as no-unit.

- Same rule applies to function formals and function return value’s unit.

- All element in a float array have the same unit. If an element is accessed, the element keeps its unit.

```plaintext
float '{km} func foo(float '{m} x) {
    float '{km} y = x;
    return y;
}

{ms} = 1000.0 '{s};

int func main() {
    float[] '{cm} x = [1.1, 2.2, 3.3];
    float '{m} y = 4.4;
    y = x[1];
    printf(y); /*2.2cm = 0.022m, prints 0.022*/
    return 0;
}
```

Keep a Set for base_units and a Map for units. Manage the set and map in unitcheck.ml
Unit System: Unit conversion

Implementation:

Auto-conversion between exprs:

Unit propagation

When unchecking exprs, recursively
- check and determines the unit for current expr
- calculate the scale (a float number)
- apply the scale by wrapping the expr with a multiplication expr

```c
/* test local var declaration with unit */
float '{cm} y;
int func main(){
    float '{m} z = 0.05;
    printf(z); /* 0.05 m*/
    y = z;     /* auto-conversion cm<-m*/
    printf(y); /* 5 cm*/
    printf(z); /* 0.05 m<- z unchanged*/
    return 0;
}
```
Unit System: Unit conversion

At assign, function arguments and function return, the conversion can be seen explicitly.

```c
// '{km}' = 0.001 '{m}';

/* parameters can be auto-converted
   when passed into function */
float '{km}' func foo(float '{m}' x) {
    printf(x); /* 30 (m) */
    float '{m}' y = x;
    return y; /* at return: convert to return unit km */
}

int func main() {
    float '{cm}' tmp = 3000.0;
    printf(foo(tmp)); /* 0.03 (km) */
    return 0;
}
```

Some of the rules in unicheck:

`eu -> u1 + u2`:
- if `u1` = no-unit,
  - follow `u2`;
- else if both have units,
  - `eu = u1`; scale `u2`;
- if `u2` cannot convert to `u1`,
  - raise error

`eu -> u1 * u2`:
- if `u1` or `u2` is no-unit:
  - follow the other;
- else:
  - `eu = u1 * u2`
Unitcheck.ml layer takes in SAST, generate Unit-checked SAST, put into codegen
Unit Checker Data Structures

```haskell
let check (udecls, globals, functions) =
  (* base unit set - static *)
  let base_units =
    List.fold_right SS.add ["m"; "s"; "kg"; "A"; "K"; "mol"; "cd"; "1"]; SS.empty
  in

  (* unit mapping key: non-base unit, value: (base unit, scale)*)
  let units =
    StringMap.add "cm" ("m", 100.0) StringMap.empty
    in

let add_unit_table (u1, u2, c) =
  match SS.find_opt u2 base_units with
  | Some bu -> StringMap.add u1 (u2, float_of_string c) table
  | None -> match StringMap.find_opt u2 table with
  | Some (bu, c2) -> StringMap.add u1 (bu, c2 *. (float_of_string c)) table
  | None -> raise (Failure ("The reference unit not existed " ^ u2))
  in

let add_unit_decls (unit_decls: unit_decl list) table =
  ignore(check_udecls "new unit" unit_decls);
  List.fold_left add_unit_table table unit_decls
  in

let units = add_unit_decls udecls units in
```

- **Basic Units Set**: `List.fold_right SS.add ["m"; "s"; "kg"; "A"; "K"; "mol"; "cd"; "1"]; SS.empty`
- **Unit Conversion Table**: `StringMap.add "cm" ("m", 100.0) StringMap.empty`
- **Add User Defined Unit and Conversion Rule to Unit Conversion Table**: `ignore(check_udecls "new unit" unit_decls);
  List.fold_left add_unit_table table unit_decls`
Unit System: derived units  e.g. m*g/s

**Algorithm for Scaling derived units:**
1. Decompose units by regular expression
2. Loop and reduce non-basic units like km, cm, ms to basic units like m, s for both sides of assignment
3. Get a map counter of base units, e.g. m*kg /s*s = {“m”: 1, “kg”: 1}, {“s”: 2}
4. Compare counter map of both sides to check if assignment is valid
5. Get conversion rate from two sides

```haskell
let derived_get_scale lunit runit = 
 let (sn, sd) = decompose_unit lunit base_units units and (sn', sd') = decompose_unit runit base_units units 
 in 
 let (snb, cn) = reduce_to_base sn and (snb', cn') = reduce_to_base sn' and (sdb, cd) = reduce_to_base sd and (sdb', cd') = reduce_to_base sd' 
 if (StringMap.equal (fun a b -> a = b) snb snb' 
 && StringMap.equal (fun a b -> a = b) sdb sdb') 
 then (cn /. cd) /. (cn' /. cd') 
 else raise(Failure("No conversion rules between unit " ^ lunit ^ " and " ^ runit)) 
```

Local variable declare and assign

Local variables inside function declarations:
- Allow Dassign anywhere in the function
- Declare and assign the variable at the same time
- Add to flexibility of the language

Implementation:
- Treat declare-and-assign as a stmt;
- Modify expr and stmt by adding a table so that the state can be memorized
Array Manipulation

- Support array initialization, element access, and element modification. Array syntax is similar to C array.
- Support for, while, as well as if-else stmts to better manipulate the array.
- Float array also support unit. Every element in the same array has the same unit. At array initialization, every expr on the right side is assumed no-unit.

```c
/* example of recording experiment result */
int func main() {
    int i = 0;
    float[] '{m} dx = [2.3, 4.5, 3.4, 0.7];
    float[] '{s} dt = [0.5, 0.2, 1.7, 0.5];
    float[] '{m/s} res = [0.0, 0.0, 0.0, 0.0];
    for (; i < 4; i = i + 1) {
        res[i] = dx[i] / dt[i];
        printf(res[i]);
    }
    return 0;
}
```

An example of array manipulation w units in a simple physics experiment.
Overview
Compiler Architecture
Features
Demo
Teamwork
/* example of recording experiment result */

int i = 0;
float['m'] dx = [2.3, 4.5, 3.4, 0.7];
float['s'] dt = [0.5, 0.2, 1.7, 0.5];
float['mm/s'] res = [0.0, 0.0, 0.0, 0.0];
for (; i < 4; i = i + 1) {
    res[i] = dx[i] / dt[i] + base;
    /* 2.3m / 0.5s = 4.6 m/s + 0.0122 m/s = 4.6122 m/s = 4612.2 mm/s */
    printf(res[i]);
}

int func main() {
    float ['cm/s'] base = 1.22;
    foo(base); /* 1.22 cm/s = 0.0122 m/s */
    return 0;
}
Teamwork

Zhengyuan Dong: Project Manager / Tester

Yucen Sun: Language Guru / Tester

Eleven Li: Compiler Architect / Tester

Future work

- Support for more complicated units (user defined mapping to derived units)
- Equation-like functions.
THANKS

Eleven Li, Yucen Sun, Zhengyuan Dong