OCalm: Week 2

Trung Vuong Hugo Frezat Justin Lelouedec Anh Tho Le Ahmed Zrigui

January 2017

1 Tasks allocation

We kept the roles we attributed and tried to implement the important features than gives us the possibility to link front-end and back-end.
For the front-end team, it was mainly focusing on α-conversion, β-reduction, let-reduction, closure conversion and virtual code generation whereas for the back-end team, it was improving and polishing the ASML structure (so that people from front-end don't have to make a lot of changes on virtual code generation every time), working on the register allocation and trying to generate an ARM assembly output (for simple expressions first). Here is in detail how the progression was made during this week:

1.1 Front-end

1.1.1 Hugo Frezat

During this week, I mainly worked on the type inference/checking and the k-normalization. I first started by implementing a function that generates a list of type equations (Tuples of Types) from expressions.
The function is implemented in `typing.ml` and is first called on the program (ast). For example, the following simple program:

```
let a = 1 + 2 in let c = a + a in _;;
```

generates a set of equations (left) that are solved during unification (right):

$$\begin{array}{rcl} int, < undef > & & int, int \\ < undef >, int & & int, int \\ < undef >, int & \Longleftrightarrow & int, int \\ int, < undef > & & int, int \\ < undef >, unit & & unit, unit \end{array}$$

Difficulties: The first version of type inference was not including function declarations and function calls. Later during the week, I tried to add support for these subexpressions and went into some trouble for function declarations that don't give any hints about anything (neither return type or parameters), for example:

```
let rec fn x = x in _;;
```

Actually here, OCaml would have say that the return type of this function is polymorphic, as well as the parameter s. In our case, we just assume that the type is unknown (equations during the unification gives $< undef >, < undef >$) and return `Int` as default.
A second case is when this unknown function is called later on, with an argument whose type is known, for example:

```
let rec fn x = x in let r = fn 1.0 in _;;
```

In this case there shouldn't be any unknown types since the function fn is called with a parameter of concrete type (Float here). The problem is that at the moment of the declaration, types are unknown but should not be solved immediately. But on the other hand, when we try to generate equations for the function call, we should already know the definition of the function, and we should be able to retrieve it.
Status: The status right know is that in the previous code example, we are able to infer that fn is of type $(float \rightarrow float)$ but we cannot infer that r is also of type Float.

K-Normalization: Since Trung has designed a structure for the K-Normal form that is used for optimizations (until closure conversion), this step was important and was a priority. There is not much difficulties in this pass, and in the same was than in the type inference, I focused on simple expressions first, just to have it working.
Later on I just had a remark from Justin saying that in ASML, there is no Bool type, and we thought it should be doable to remove it from K-Normal and do the conversion into Int ($false \rightarrow 0$ and $true \rightarrow 1$). The other modification that is suggested in the mincaml article is to transform if expressions into two specific types of comparisons (equals and less or equal). This was simply guided by the fact that in our If expression, we have access to the boolean part and we can just test whether it is an Equal or a LE.
End of the week: I tried to bridge the front-end and back-end and since Trung was working on closure conversion, I just worked on the last part, which is virtual code generation. The idea here was to translate the closure type into ASML. It was quite hard to grasp the concept because the data structure for ASML is quite complex. Fortunatly, we have already much work done in the previous parts and the type that we have in closure is quite suited for this conversion. I implemented quickly simple expressions (variable declarations, operators and variables) which let us to a "quasi-complete" pipeline.

1.1.2 Trung Vuong

My work this week focused mainly on α-conversion, β-reduction and closure conversion. So far α-conversion and β-reduction work for almost every syntax in the language, except tuple and array. For closure conversion, there is only success in conversion for direct function (without free variables) calls, which means for now, we are only able to move function definitions into top level that can be translated into direct call for the back-end. Due to difficulties with closure conversion for closures, this part is temporarily suspended to focus on tasks with higher priority.

I also worked on k-normalization for function definition (`LetRec`) and application, improved type checking for function application. A major difficulty in this part for now is to perform type checking with function application in which the function label is not a simple identifier, but a complex expression that yields a function.

1.1.3 Anh Tho Le

My work during this week focuses on two front-end transformation using K-Normal form and used right after : nested let reduction and inline expansion. The nested let reduction transformation has passed several test cases involving functions with nested let as follows:

```
let rec f x = x + 1 in
let g = x + y in
f y
```

The inline expansion has been tested with functions whose body size is greater than 0, for example

```
 let rec f x =
    let y =
        let v1 = 1 in
        let v2 = 2
        in v1 + v2
        in x + y in
    let x = 2 in
f x
```

1.2 Back-end

1.2.1 Justin Lelouedec

My work during this week was to focus on back end, and in particular on finalizing ASML data structure, creating a simple register allocation and generating simple Assembly code from the two previous parts.
For the Asml type, I continued what I started the week before and tried to make it clearly understandable for the rest of the team while I was adapting the type to make the assembly code generation easier. For example, after discussing with Trung I changed the $format\_arg$ type, from a recursive one to a list of Identifiers.
For the register allocation I created a data struture to simplify the allocation for every variable. I decided to use two hashtables:

- The first one maps for every functions a hashtable of its variables.
- The second one maps variable to its register in the function environment

Using this type of data structure I can use in a first time a basic allocation technique which is mapping variables to a pseudo infinite number of registers, and divide this number by the number of functions. Of course, this technique is far away from optimality but I keep its improvement for the following week.
Finally, for the Assembly generation, I started its implementation for simple programs with only one function and simple operations. I will complete it next week as I am currently working on the stack and heap management.
Difficulties Because Ocaml is completly new for me, it took me a bit of time to understand it and to learn how to use it properly. Also I encountered some difficulties for the Assembly code generation because of the heap and stack management.

1.2.2 Ahmed Zrigui

During this week we finished the ASML datatype, we implemented a basic register allocation strategy supposing that we have an infinite number of registers, and we generated assembly code for basic operations, Actually, I was supposed to contribute and write some code but Justin did most of it, because I took much time to understand, and it wasn't easy for me to contribute. I was working on creating some testing examples (expressions in the ASML datatype) for the backend.

2 Supported features at the moment

We managed to output some ASM from an input file in mincaml just this afternoon. It is quite simple at the moment, and the ASM contains some flaws in the register allocation but we are on a good way, Our pipeline starting from the following mincaml file:

```
let a = 1 in let c = a + (-a) in _;;
```

Applying type inference : $?v1 -> int \text{ — } a -> int \text{ — } c -> int$

Applying $\alpha/\beta/let$ reductions :

```
(let (?v2 : int) = 1 in
(let (?v4 : int) = (neg ?v2) in
(let (?v3 : int) = (?v2 = ?v4) in
())))
```

Applying virtual code generation :

```
Let _ =
    Let ?v2 = 1 in
    Let ?v4 = neg (?v2) in
    Let ?v3 = ?v2 + ?v4 in
```

Applying register allocation and code generation :

```
_main : MOV R1, #1
        ADD r4, R0, R1
        ADD r5, R0, R1
        ADD r4, r4, r5
        ADD r6, R0, R1
```

3 Next week overview

- Implement a new allocation strategy for the register allocation
- Finish the Assembly generation and work on the heap and stack
- Refine type checking
- Continue work with closure conversion

4 Git

As you may have noticed, there is only one push to the forge. This is because we decided to work on Github on a private repository (we have student accounts that allow us to create unlimited private repos), since we found the forge not practical to use.
Here is a log of the commits we've done (mainly during this week, quite active though). We worked on separate branches, frontend and backend, that we just merged yesterday when working on virtual code generation.

```
    *   f6b42cc Merge branch 'integration' of https://github.com/hrkz/ocalm into integration
    |\
    | * 1c65a00 use flag -asml to print out asml and asm
    * |   5c96be6 Merge branch 'integration' of https://github.com/hrkz/ocalm into integration
    |\ \
    | |/
    | * 769029a update type checking for fun app when label is not a simple string
    | * ec41999 handle type checking for fun app when nb of supplied args are different from that
in fun def
    | * 7a168e5 add type checking for arguments in external function call
    | * 496e97b output asm from input file (full pipeline)
    | * 8d3f6af Improved readibility of register alloc tests
    * | 8bbfa22 inline expansion added (no test yet)
    |/
    * 09274e2 bug resolved for register allocation
    *   1007a8a Merge branch 'integration' of https://github.com/hrkz/ocalm into integration
    |\
    | * ffdf2ba type inference able to solve function types but not return type..
    | * 2879cd4 add first try with virtual (gonna try harder this noon
    | *   ffc76e2 Merge branch 'integration' of https://github.com/hrkz/ocalm into integration
    | |\
    | | * 14d5fb3 add system test for typing
    | | * 3538fd5 update test parser runner & add some syntax tests
    | * | 6c8bb74 Merge
    | |/
    | * 08020cd Add test backend
    | * ce6c596 add command line options
    * | 6174c48 some simple assembly generation working and function call changed
    |/
    * 8a68ab7  main restored
    * 74f514a  main changed to include back end
    * e303f00 merging back end and front end CORRECTLY
    * 594d387 add external function case for closure-conversion
    * f446653 differentiate app for known and unknown (lib) functions in k-norm form to avoid name
replacement on a-conversion
    * e4aa7ec remove bin file from svn
    * 177d578 correct syntax in main
    *   8fa9b46 Merge branch 'backend' into integration
    |\
    | * 6eda112  final change on call label so we can put integer as parameter and variable too (
easier for assembly conversion
    | * dd8cce9  starting on assembly and modification on call label and call closure type
    | * 3abebe5 bug found and resolved in register allocation and asml
    | * e08f0dc improvement on register allocation + tests
    | * 45619ea improvement on register allocation
    | * 3bca73a  register allocation improved
    | * 26d7d05 Including some basic tests  in main
    | * 501582c Add files via upload
```

```
    | * fdb17f9 Add files via upload
    | * 41941a7 Basic tests
    | * 667612b asml type modified and finished , register allocation for basic technique working
but can be improved
    | * 253e813 improvment on the asml type and display functions
    | * eb82911  asml type created and functions to display it ready to use
    | * f2a11aa  type maybe not complete but working and can be dsplayed
    | * 7265df5  asml added nd standalone code for reading asml file
    | * af9abc4  data type for asml started
    | * 3fa0eb1  added asml type to Ocaml directory
    * | ab8525b let reduction modified
    * | c08bf21 LetRec typing test + fixed other LetRec declarations
    * | 44a514d impl b-reduction for let-rec and function app
    * | 1f048e0 impl a-conversion of function app
    * | b4dd589 impl a-conversion for let-rec
    * | 21f3135 minor code formatting
    * | 70a74ae rename file & reference in let_test to avoid failed build
    * | b6f4db3 remove Bool pattern for closure-conversion
    * | b020489 clarify failure message for not-defined-yet syntax for a-conversion & b-reduction
    * | 29def8f correct the arguments order of App after k-normalization
    * | 3f5faa4 make k-normalization correct for App with any numbers of arguments
    * | 42ca11e impl k-normalization for app & let rec (wip)
    * | a842f76 minor update to avoid warnings on type/struct from external module
    * | 118bd1d Add type inference for LetRec
    * | 2069710 Commit bcuz we got kicked from the room
    * | af36ebf nested let reduction added
    * | bbd5584 Converting If to IfEq and IfLE
    * | b7153ff Removed Bool from kNormal and added support for Not/Neg
    * | d3afa04 compute set of free variables in a LetRec exp
    * | bdeac6c add helper function to list free variables in a (k-normed) exp
    * | 60d3ca1 make to_string of function type clearer
    * | f9db04e add a test that requires make closure & apply closure instead of direct call
    * | ae4dc9c Added first try with simple let-reductions (TODO:tests)
    * |   078e09a Merge branch 'frontend' of https://github.com/hrkz/ocalm into frontend
    |\ \
    | * | 3ffcb86 revert change in typing on main
    | * | 316c16c minor update such that typing won't reject expression of type non-unit
    | * | 146ae2e apply closure-conversion for direct fn call and test with to_string
    * | | 09b3bdc merge kNormal
    |/ /
    * | e9b65a9 try performing closure-conversion
    * | 09aa652 init typing for closures
    * | 571c043 rm test bin on clean
    * | 5f40df5 add to_string for fn app / def in knorm form
    * | 9eb8c34 use the same map data-structure for a-conversion & b-reduction
    * | ca7ff3b Simple k-normalization for operators, variables and let
    * | 3b8dd47 Make symbol tables references (accessible from outside - see main.ml
    * | f1dce6d extend b-reduction for all arithmetic operations
    * | 1e9a30e update b-reduction for let' with rhs being a complex expression
    * | 9c815fb improve makefile for tests
    * | 8903aab add beta-reduction for let expression with a single variable on rhs
    * | c0f1985 Added support to free/nonfree variable uses inference
    * | 3ec9542 Remove mincamlc, ignore file
    * | ff2d5a7 Simple type inference for let and operators
    * | 998d623 refine a-conversion to always replace vars for simplicity
    * | 990965e a-convert for + and - operators
    * | 0b087ba a-conversion for let with simple arithmetic
    * | f1a7908 implement alpha-conversion on Let (var) expression (wip)
    * | 180c8c6 start with alpha-conversion
    * | a81c3bf add k-normalization typing
    |/
    * e72a0af Add get_env - returning a set of variables
    * e51b7bc Fix README markdown
    * 47114f9 Push base archive
    * c20eee3 Initial commit
```