Script **generated by TTT**

Title: Petter: Compiler Construction (25.06.2020) - 45: Type Systems

Date: Thu Jun 25 12:18:35 CEST 2020

Duration: 18:18 min

Pages: 11

Chapter 3: Type Checking

Goal of Type Checking

In most mainstream (imperative / object oriented / functional) programming languages, variables and functions have a fixed type.
for example: `int`, `void*`, `struct { int x; int y; }`.

Types are useful to

- manage memory
- select correct assembler instructions
- to avoid certain run-time errors

Goal of Type Checking

In most mainstream (imperative / object oriented / functional) programming languages, variables and functions have a fixed type.
for example: `int`, `void*`, `struct { int x; int y; }`.

Types are useful to

- manage memory
- select correct assembler instructions
- to avoid certain run-time errors

In imperative and object-oriented programming languages a declaration has to specify a type. The compiler then checks for a type correct use of the declared entity.

Type Expressions

Types are given using type-*expressions*.
The set of type expressions T contains:

1. base types: **int**, **char**, **float**, **void**, ...
2. type constructors that can be applied to other types

Type Expressions

Types are given using type-*expressions*.
The set of type expressions T contains:

1. base types: **int**, **char**, **float**, **void**, ...
2. type constructors that can be applied to other types

example for type constructors in C:

- structures: **struct** $\{ t_1\ a_1; \ldots t_k\ a_k; \}$
- pointers: $t\ *$
- arrays: $t\ [\,]$
  - the size of an array can be specified
  - the variable to be declared is written between t and $[n]$
- functions: $t\ (t_1, \ldots, t_k)$
  - the variable to be declared is written between t and $(t_1, \ldots, t_k)$
  - in ML function types are written as: $t_1 * \ldots * t_k \rightarrow t$

Type Checking

Problem:

Given: A set of type declarations $\Gamma = \{t_1\ x_1; \ldots t_m\ x_m; \}$
Check: Can an expression e be given the type t?

Type Checking

Problem:

Given: A set of type declarations $\Gamma = \{t_1\ x_1; \ldots t_m\ x_m; \}$
Check: Can an expression e be given the type t?

Example:

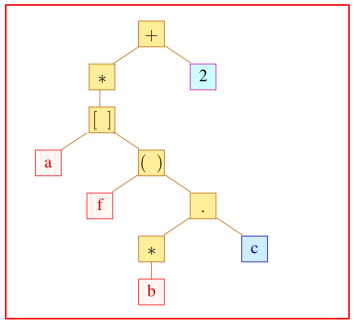
```
struct list { int info; struct list* next; };
int f(struct list* l) { return 1; };
struct { struct list* c;}* b;
int* a[11];
```

Consider the expression:

```
*a[f(b->c)]+2;
```

Type Checking using the Syntax Tree

Check the expression `*a[f(b->c)]+2`:



Idea:

- traverse the syntax tree bottom-up
- for each identifier, we lookup its type in Γ
- constants such as 2 or 0.5 have a fixed type
- the types of the inner nodes of the tree are deduced using *typing rules*

Type Systems for C-like Languages

Formally: consider *judgements* of the form:

$$\Gamma \vdash e : t$$

// (in the type environment Γ the expression e has type t)

Axioms:

Const: $\Gamma \vdash c : t_c$ (t_c type of constant c)
Var: $\Gamma \vdash x : \Gamma(x)$ (x Variable)

Rules:

Ref: $$\frac{\Gamma \vdash e : t}{\Gamma \vdash \&\, e : t*}$$

Deref: $$\frac{\Gamma \vdash e : t*}{\Gamma \vdash *e : t}$$

Type Systems for C-like Languages

More rules for typing an expression:

Array: $$\frac{\Gamma \vdash e_1 : t* \quad \Gamma \vdash e_2 : \mathbf{int}}{\Gamma \vdash e_1[e_2] : t}$$

Array: $$\frac{\Gamma \vdash e_1 : t[\,] \quad \Gamma \vdash e_2 : \mathbf{int}}{\Gamma \vdash e_1[e_2] : t}$$

Struct: $$\frac{\Gamma \vdash e : \mathbf{struct}\,\{t_1\, a_1; \ldots t_m\, a_m;\}}{\Gamma \vdash e.a_i : t_i}$$

App: $$\frac{\Gamma \vdash e : t(t_1, \ldots, t_m) \quad \Gamma \vdash e_1 : t_1 \;\ldots\; \Gamma \vdash e_m : t_m}{\Gamma \vdash e(e_1, \ldots, e_m) : t}$$

Op $\square$: $$\frac{\Gamma \vdash e_1 : t \quad \Gamma \vdash e_2 : t}{\Gamma \vdash e_1 \square e_2 : t}$$

Op $=$: $$\frac{\Gamma \vdash e_1 : t_1 \quad \Gamma \vdash e_2 : t_2 \quad t_2 \text{ can be converted to } t_1}{\Gamma \vdash e_1 = e_2 : t_1}$$

Explicit Cast: $$\frac{\Gamma \vdash e : t_2 \quad t_2 \text{ can be converted to } t_1}{\Gamma \vdash (t_1)\, e : t_1}$$

Type Systems for C-like Languages

More rules for typing an expression: with subtyping relation $\leq$

Array: $$\frac{\Gamma \vdash e_1 : t* \quad \Gamma \vdash e_2 : \mathbf{int}}{\Gamma \vdash e_1[e_2] : t}$$

Array: $$\frac{\Gamma \vdash e_1 : t[\,] \quad \Gamma \vdash e_2 : \mathbf{int}}{\Gamma \vdash e_1[e_2] : t}$$

Struct: $$\frac{\Gamma \vdash e : \mathbf{struct}\,\{t_1\, a_1; \ldots t_m\, a_m;\}}{\Gamma \vdash e.a_i : t_i}$$

App: $$\frac{\Gamma \vdash e : t(t_1, \ldots, t_m) \quad \Gamma \vdash e_1 : t_1 \;\ldots\; \Gamma \vdash e_m : t_m}{\Gamma \vdash e(e_1, \ldots, e_m) : t}$$

Op $\square$: $$\frac{\Gamma \vdash e_1 : t_1 \quad \Gamma \vdash e_2 : t_2}{\Gamma \vdash e_1 \square e_2 : t_1 \sqcup t_2}$$

Op $=$: $$\frac{\Gamma \vdash e_1 : t_1 \quad \Gamma \vdash e_2 : t_2 \quad t_2 \leq t_1}{\Gamma \vdash e_1 = e_2 : t_1}$$

Explicit Cast: $$\frac{\Gamma \vdash e : t_2 \quad t_2 \leq t_1}{\Gamma \vdash (t_1)\, e : t_1}$$