Script generated by TTT

Title: Petter: Compiler Construction (25.06.2020) - 47: Structural Equivalence of Types

Date: Thu Jun 25 13:07:18 CEST 2020

Duration: 17:00 min

Pages: 9

Example: Type Checking – More formally:

$\Gamma = \{$

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

$\}$

$$\textsc{Struct}\ \dfrac{\textsc{Deref}\ \dfrac{\textsc{Var}\ \dfrac{}{\Gamma \vdash b : \text{struct\{struct list *c;\}}*}}{\Gamma \vdash *b : \text{struct\{struct list *c;\}}}}{\Gamma \vdash (*b).c : \text{struct list}*}$$

$$\textsc{Array}\ \dfrac{\textsc{Var}\ \dfrac{}{\Gamma \vdash a : \text{int}*[\,]} \quad \textsc{App}\ \dfrac{\textsc{Var}\ \dfrac{}{\Gamma \vdash f : \text{int(struct list*)}\ \checkmark} \quad \Gamma \vdash (*b).c : \text{struct list}*}{\Gamma \vdash f(b \to c) : \text{int}\ \checkmark}}{\Gamma \vdash a[f(b \to c)] : \text{int}*}$$

$$\textsc{Op}\ \dfrac{\textsc{Deref}\ \dfrac{\Gamma \vdash a[f(b \to c)] : \text{int}*}{\Gamma \vdash *a[f(b \to c)] : \text{int}} \quad \textsc{Const}\ \dfrac{}{\Gamma \vdash 2 : \text{int}\ \checkmark}}{\Gamma \vdash *a[f(b \to c)] + 2 : \text{int}}$$

but what do we do with $\boxed{\leq}$?

Equality of Types $\boxed{=}$

Summary of Type Checking

- Choosing which rule to apply at an AST node is determined by the type of the child nodes
- determining the rule requires a check for ⇝ *equality* of types

type equality in C:

- **struct** A { } and **struct** B { } are considered to be different
  - ⇝ the compiler could re-order the fields of A and B independently (*not* allowed in C)
  - to extend an record A with more fields, it has to be embedded into another record:

```
struct B {
    struct A;
    int field_of_B;
} extension_of_A;
```

- after issuing `typedef int C;` the types C and **int** are the same

Structural Type Equality

Alternative interpretation of type equality (*does not hold in C*):

semantically, two types t_1, t_2 can be considered as *equal* if they accept the same set of access paths.

Example:

```
struct list {
  int info;
  struct list* next;
}
```

```
struct list1 {
  int info;
  struct {
    int info;
    struct list1* next;
  }* next;
}
```

Consider declarations **struct** list* l and **struct** list1* l. Both allow

`l->info    l->next->info`

but the two declarations of l have unequal types in C.

Algorithm for Testing Structural Equality

Idea:

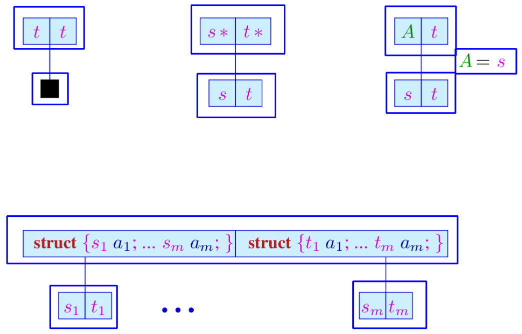
- track a set of equivalence queries of type expressions
- if two types are syntactically equal, we stop and report success
- otherwise, reduce the equivalence query to a several equivalence queries on (hopefully) simpler type expressions

Suppose that recursive types were introduced using type definitions:

$$\texttt{typedef}\ A\ t$$

(we omit the Γ). Then define the following rules:

Rules for Well-Typedness



Example:

$$\begin{array}{lll} \texttt{typedef} & \texttt{struct}\ \{\texttt{int}\ \text{info};\ A * \text{next};\} & A \\ \texttt{typedef} & \texttt{struct}\ \{\texttt{int}\ \text{info};\ \texttt{struct}\ \{\texttt{int}\ \text{info};\ B * \text{next};\} * \text{next};\} & B \end{array}$$

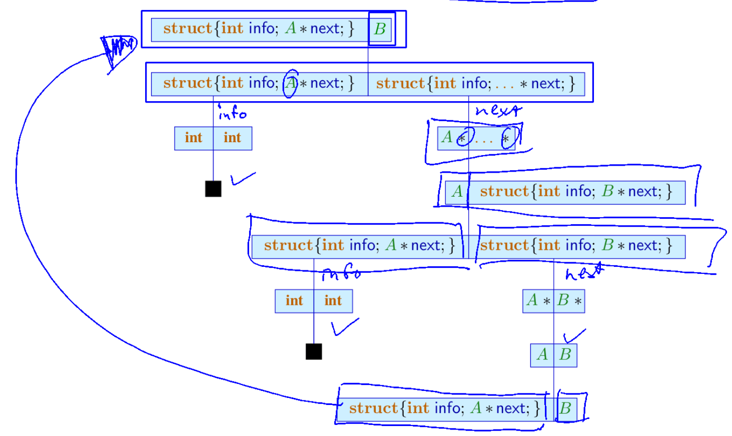
We ask, for instance, if the following equality holds:

$$\boxed{\texttt{struct}\ \{\texttt{int}\ \text{info};\ A * \text{next};\} = B}$$

We construct the following deduction tree:

Proof for the Example:

$$\begin{array}{lll} \texttt{typedef} & \texttt{struct}\ \{\texttt{int}\ \text{info};\ A * \text{next};\} & A \\ \texttt{typedef} & \texttt{struct}\ \{\texttt{int}\ \text{info};\ \texttt{struct}\ \{\texttt{int}\ \text{info};\ B * \text{next};\} * \text{next};\} & B \end{array}$$



Implementation

We implement a function that implements the equivalence query for two types by applying the deduction rules:

- if no deduction rule applies, then the two types are $\boxed{\textit{not equal}}$
- if the deduction rule for expanding a type definition applies, the function is called recursively with a *potentially larger* type
- in case an equivalence query appears a second time, the types are *equal by definition*

Implementation

We implement a function that implements the equivalence query for two types by applying the deduction rules:

- if no deduction rule applies, then the two types are *not equal*
- if the deduction rule for expanding a type definition applies, the function is called recursively with a *potentially larger* type
- in case an equivalence query appears a second time, the types are *equal by definition*

Termination

- the set D of all declared types is finite
- there are no more than $|D|^2$ different equivalence queries
- repeated queries for the same inputs are automatically satisfied

$\leadsto$ termination is ensured