Sequence 3.4 – Types verification and inference

P. de Oliveira Castro S. Tardieu

Typing operations in the compiler

- Tiger is a *statically typed* language: every entity in the source code has a fixed type determined at compilation time.
- A type can be explicit (given by the programmer).
- A type may be implicit (determined unambiguously by the compiler).

```
let var a : int := 3  /* Explicit type: int */
    var b := a + 4    /* Implicit type: int */
    var c := "Hello"  /* Implicit type: string */
in
  a + b + strlen(c)   /* Implicit type: int */
end
```

The types in Tiger

In our Tiger implementation, three types are used:

- `int`: an 32 bit integer;
- `string`: a string;
- `void`: the absence of value

For example, function `inc` below returns void (no value):

```
let var a := 0
    function f() = a := a + 1
in
  f(); f(); f(); a  /* Returns 3 */
end
```

The type checker

- The *type checker* (or *typer* in short) is a compiler phase with three purposes:
  - assign a type to every expression;
  - assign a type to every entity (variables, function return values);
  - check for type consistency throughout the program.
- The type checker runs after the binder.
- The type checker can be implemented using a visitor walking the AST.

Type checking examples: expressions

- The expression `c := a + 2` is easy to type check:
  - `+` is an operation taking two `int` and returning an `int`;
  - therefore the type of `a` must be `int`;
  - the type of `c` must also be `int`;
  - 2 is an `IntegerLiteral`, which is of type `int`, so this is correct.
- The expression `c := a < b` offers more challenges:
  - `<` (lower than) is a comparaison operation, whose operands must have the same type (`int` or `string`), returning an `int`;
  - therefore the type of `a` and the type of `b` must be identical although unknown at this stage;
  - the type of `c` must be `int`, as `<` returns 1 (true) or 0 (false).

Type checking examples: if/then/else

In Tiger, everything is an expression, including `if`/`then`/`else`.

```
if test then something else something_else
```

- `test` must be an `int` (0 means false, anything else means true);
- `something` and `something_else` must have the same type (`int`, `string` or `void`) which will also be the type of the whole `if`/`then`/`else` expression.

For example, the following expression returns the smallest value of `a` and `b`, `a` and `b` having the same type (`int` or `string`):

```
if a < b then a else b
```

Type checking examples: if/then

It is possible to omit the `else` part.

```
if test then something
```

In this case:

- `test` is an `int`;
- `something` must be `void`: since there is no `else` branch, if `test` is 0 (false) then no value will be returned, so the type of the whole `if/then` expression is `void`, so the type of the `then` branch is void.

Note that in Tiger, `()` is of type `void`, so omitting the `else` part is equivalent to using `else ()`. This substitution (adding `else ()`) can even be done in the parser to reduce the complexity of the type various visitors.

Summarizing tests

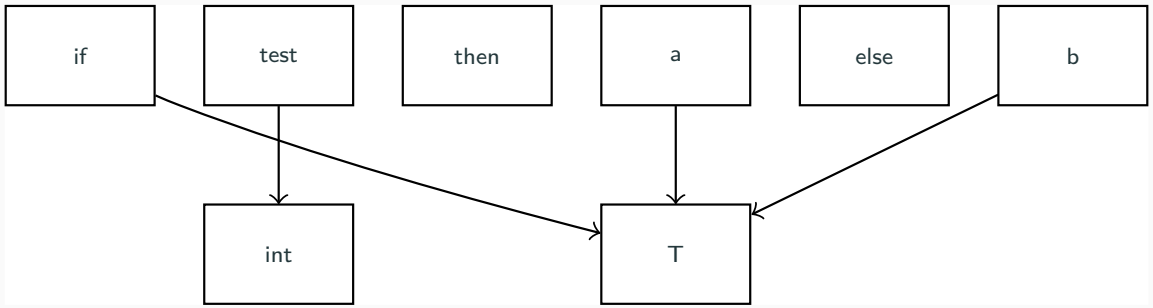

Figure 1: if/then/else

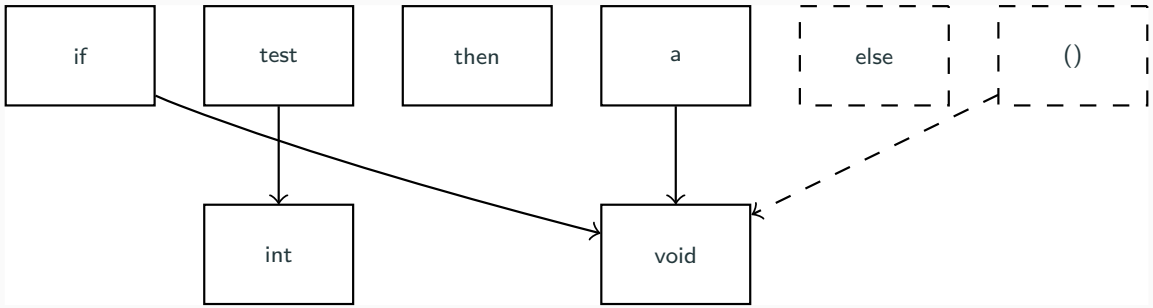

Figure 2: if/then

Tiger is simple to type check

Tiger is a simple language to type check while walking the AST:

- the type of every function parameter is explicit (`int` or `string`);
- the return type of every function is explicit (`int`, `string`, or nothing for `void`);
- any literal can be type checked (strings are surrounded by double quote characters, integers are not);
- any expression can be type checked since it only refers to previously defined (and type checked) entities;
- any new variable declaration can be type checked from its initialization expression.

Example: type checking of assignment (:=)

```
void TypeChecker::visit(Assign &assign) {
  // Recursively type check left hand side of :=
  // (which is necessarily an identifier)
  assign.get_lhs().accept(*this);
  // Recursively type check right hand side of :=
  assign.get_rhs().accept(*this);
  // Check that both sides of := have the same type
  // and exit with an error otherwise.
  if (assign.get_lhs().get_type() !=
      assign.get_rhs().get_type())
    error(assign.loc, "Type mismatch in assignment");
  // The assignment itself returns no value,
  // you cannot do a := (b := c).
  assign.set_type(t_void);
}
```

Some languages are harder to type check

- In some programming languages, types may be omitted at more places than in Tiger.
- In this case, the type checker group entities and expressions in *cliques*, which are sets of items sharing the same (possibly unknown yet) type.
- When the type checker determines that two items must have the same type, they merge the cliques those items belong to and check for consistency.
- When the type checker determine that an item must be of a given type, it assigns this type to every item of its clique, and merges the clique with an existing clique of the same type if any.

Conclusion

- The type checker (or typer) runs after the binder.
- Its role is to determine the type of expression and entities, and to check their consistencies according to the language rules.
- In some languages such as Tiger, type checking is a straightforward operations which only requires walking the AST once using a visitor.
- In some other languages, type checking requires the use of more advanced data structures such as cliques in order to determine the type of various items.