Einführung in die Programmierung
Introduction to Programming

Prof. Dr. Bertrand Meyer

Exercise Session 13
Today

- Mock exam 2 review
- Tuples and agents
- Reminder: no exercise session next week
A tuple of type `TUPLE[A, B, C]` is a sequence of at least three values, first of type `A`, second of type `B`, third of type `C`.

In this case possible tuple values that conform are:

- `[a, b, c], [a, b, c, x],...`
  where `a` is of type `A`, `b` of type `B`, `c` of type `C` and `x` of some type `X`

Tuple types (for any types `A, B, C, ...`):

- `TUPLE`
- `TUPLE[A]`
- `TUPLE[A, B]`
- `TUPLE[A, B, C]`
- ...
tuple_conformance

local

t0: TUPLE
t2: TUPLE [INTEGER, INTEGER]

do

create t2
t2 := [10, 20]
t0 := t2

print (t0.item (1).out + "%N")

print (t0.item (3).out)

end

Not necessary in this case

Implicit creation

Runtime error, but will compile
Labeled Tuples

- Tuples may be declared with labeled arguments:
  ```haskell
tuple: TUPLE [food: STRING; quantity: INTEGER]
  ```

- Same as an unlabeled tuple:
  ```haskell
  TUPLE [STRING, INTEGER]
  ```
  but provides easier (and safer!) access to its elements:
  May use
  ```haskell
  Io.print (tuple.food)
  ```
  instead of
  ```haskell
  Io.print (tuple.item (1))
  ```
What are agents in Eiffel?

- Objects that represent operations
- Can be seen as operation wrappers
- Similar to
  - delegates in C#
  - anonymous inner classes in Java < 7
  - closures in Java 7
  - function pointers in C
  - functors in C++
Agent definition

- Every agent has an associated routine, which the agent wraps and is able to invoke.

- To get an agent, use the `agent` keyword.
  e.g. `a_agent := agent my_routine`

- This is called `agent definition`.

- What’s the type of `a_agent`?
EiffelBase classes representing agents

- ROUTINE
- FUNCTION
- PROCEDURE
- PREDICATE

* call
+ item
Agent Type Declarations

\[ p: \text{PROCEDURE}[\text{ANY, TUPLE}] \]
Agent representing a procedure belonging to a class that conforms to ANY. At least 0 open arguments

\[ q: \text{PROCEDURE}[\text{C, TUPLE}[\text{X, Y, Z}]] \]
Agent representing a procedure belonging to a class that conforms to C. At least 3 open arguments

\[ f: \text{FUNCTION}[\text{ANY, TUPLE}[\text{X, Y}], \text{RES}] \]
Agent representing a function belonging to a class that conforms to ANY. At least 2 open arguments, result of type \text{RES}
Open and closed agent arguments

- An agent can have both “closed” and “open” arguments:
  - closed arguments are set at agent definition time
  - open arguments are set at agent call time.
- To keep an argument open, replace it by a question mark

```
u := agent a0.f (a1, a2, a3)            -- All closed
v := agent a0.f (a1, a2, ?)
w := agent a0.f (a1, ?, a3)
x := agent a0.f (a1, ?, ?)
y := agent a0.f (?, ?, ?)
z := agent {C}.f (?, ?, ?)              -- All open
```
Agent Calls

An agent invokes its routine using the feature “call”

\[ f(x_1: T_1; x_2: T_2; x_3: T_3) \]

-- defined in class C with
-- a0: C; a1: T1; a2: T2; a3: T3

\[ u := \text{agent } a0.f(a1, a2, a3) \]
\[ v := \text{agent } a0.f(a1, a2, ?) \]
\[ w := \text{agent } a0.f(a1, ?, a3) \]
\[ x := \text{agent } a0.f(a1, ?, ?) \]
\[ y := \text{agent } a0.f(?, ?, ?) \]
\[ z := \text{agent } \{C\}.f(? ,?, ?) \]

What are the types of the agents?
Doing something to a list

Given a simple ARRAY [G] class, with only the features `count' and `at', implement a feature which will take an agent and perform it on every element of the array.

```
do_all (do_this: PROCEDURE[ANY, TUPLE[G]])
  local
    i: INTEGER
  do
    from i := 1
    until i > count
    loop
      do_this.call([at (i)])
      i := i + 1
    end
  end
```

Hands-On
For-all quantifiers over lists

\[\text{for\_all } (\text{pred} : \text{PREDICATE [ANY, TUPLE[G]]}) : \text{BOOLEAN}\]

\[
\begin{align*}
\text{local} & \quad i : \text{INTEGER} \\
\text{do} & \\
\text{Result} & := \text{True} \\
\text{from} & \\
& \quad i := 1 \\
\text{until} & \\
& \quad i > \text{count or not Result} \\
\text{loop} & \\
& \quad \text{Result} := \text{pred.item ([at (i)])} \\
& \quad i := i + 1 \\
\text{end} & \\
\text{end} &
\end{align*}\]
Using inline agents

We can also define our agents as-we-go!

Applying this to the previous `for_all` function we made, we can do:

```plaintext
for_all_ex (int_array : ARRAY [INTEGER]): BOOLEAN
  local
    greater_five: PREDICATE [ANY, TUPLE [INTEGER]]
  do
    greater_five := agent (i : INTEGER): BOOLEAN
      do
        Result := i > 5
      end
    end
  Result := int_array.for_all (greater_five)
end
```
Problems with Agents/Tuples

We have already seen that \text{TUPLE} [A,B] conforms to \text{TUPLE} [A]. This raises a problem. Consider the definition:

\[
f (\text{proc} : \text{PROCEDURE} \ [\text{ANY}, \ \text{TUPLE} \ [\text{INTEGER}]])
\]

\[
\text{do }
\]

\[
\text{proc.call} \ ([5])
\]

\[
\text{end}
\]

Are we allowed to call this on something of type \text{PROCEDURE} \ [\text{ANY}, \ \text{TUPLE} \ [\text{INTEGER}, \ \text{INTEGER}]]?\]

Yes! Oh no... that procedure needs at least TWO arguments!
Questions (TUPLEs, agents, course, …)?