Einführung in die Programmierung
Introduction to Programming

Prof. Dr. Bertrand Meyer

Exercise Session 13
Today

- Mock exam 2 review
- Tuples and agents
Tuples

- 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

tuple_conformance

local

\[ t_0: \text{TUPLE} \]
\[ t_2: \text{TUPLE} \ [\text{INTEGER, INTEGER}] \]

do

create \[ t_2 \]

\[ t_2 := [10, 20] \]
\[ t_0 := t_2 \]

print (t_0.item (1).out + "%N")
print (t_0.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:
  
  ```
  tuple: TUPLE [food: STRING; quantity: INTEGER]
  ```

- Same as an unlabeled tuple:
  
  ```
  TUPLE [STRING, INTEGER]
  ```
  
  but provides easier (and safer!) access to its elements:

  May use
  
  ```
  Io.print (tuple.food)
  ```

  instead of
  
  ```
  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. For example: `a_agent := agent my_routine`.

- This is called the `agent definition`.

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

- **ROUTINE**
  - **PROCEDURE**
  - **FUNCTION**
  - **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
-- \( a_0: C; a_1: T_1; a_2: T_2; a_3: T_3 \)

\[ u := \text{agent } a_0.f(a_1, a_2, a_3) \]
\[ v := \text{agent } a_0.f(a_1, a_2, ?) \]
\[ w := \text{agent } a_0.f(a_1, ?, a_3) \]
\[ x := \text{agent } a_0.f(a_1, ?, ?) \]
\[ y := \text{agent } a_0.f(?, ?, ?) \]
\[ z := \text{agent } \{C\}.f(?, ?, ?) \]

What are the types of the agents?
Given a simple `ARRAY [G]` class, with only the features `\texttt{`count'}` and `\texttt{`at'}`, implement a feature which will take an agent and perform it on every element of the array.

\begin{verbatim}
\textit{do\_all} (\textit{do\_this} : \texttt{PROCEDURE[ANY, TUPLE[G]]})
local
  \texttt{i : INTEGER}
do
from
  \texttt{i := 1}
until
  \texttt{i > count}
loop
  \texttt{do\_this\_call ([at (i)])}
i := i + 1
end
end
\end{verbatim}
For-all quantifiers over lists

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

\[
\text{local}
\]

\[
i : \text{INTEGER}
\]

\[
do
\]

\[
\text{Result} := \text{True}
\]

\[
\text{from}
\]

\[
i := 1
\]

\[
\text{until}
\]

\[
i > \text{count} \text{ or not Result}
\]

\[
\text{loop}
\]

\[
\text{Result} := \text{pred.item ([at (i)])}
\]

\[
i := i + 1
\]

\[
\text{end}
\]

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

\[
\begin{align*}
f (\text{proc} : \text{PROCEDURE} \ [\text{ANY, TUPLE} \ [\text{INTEGER}]]) \\
\quad \text{do} \\
\quad \quad \text{proc.call} ([5]) \\
\quad \text{end}
\end{align*}
\]

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

Yes! Oh no... that procedure needs at least TWO arguments!