Compiler Construction

Exam 2020 review

Goal

Modify the C++ language to introduce a simplified variant of the **when** keyword avaible in kotlin

```
when (x) {
  0 -> "0"
  1 -> "1"
  else -> "e"
}
```

```
when {
  x > 0 -> 42
  x == 0 -> 1337
  x < 0 -> 51
}
```

Question 1.

Do we need to modify the scanner?

- ☐ Yes, add ”->”
- ☐ Yes, add ”>”
- ☐ No
- ☐ Yes, add ”else”
- ☐ Yes, add ”when”
- ☐ Yes, add ”-”

Question 1.

Do we need to modify the scanner?

- [x] Yes, add ”->”
- [ ] Yes, add ”>”
- [ ] No
- [ ] Yes, add ”else”
- [x] Yes, add ”when”
- [ ] Yes, add ”-”

Question 2.

Introducing this construct may raise conflicts?

- ☐ reduce/reduce on ”else”
- ☐ shift/reduce on ”-”
- ☐ shift/reduce on ”->”
- ☐ shift/reduce on ”else”
- ☐ reduce/reduce on ”->”
- ☐ shift/reduce on ”->”
- ☐ shift/reduce on ”when”
- ☐ reduce/reduce on ”when”

Question 2.

Introducing this construct may raise conflicts?

- ☐ reduce/reduce on ”else”
- ☐ shift/reduce on ”-”
- ☐ shift/reduce on ”->”
- ☐ shift/reduce on ”else”
- ☑ reduce/reduce on ”->”
- ☑ shift/reduce on ”->”
- ☐ shift/reduce on ”when”
- ☐ reduce/reduce on ”when”

Question 3. What is the EBNF for this construct?

- ```
  exp ::= ...
    | "when" exp? {
        exp+ -> exp+
      }
    | "else"
  ```
- ```
  stm ::= ...
    | "when" exp? {
        exp+ -> stm+
      }
    | "else"
  ```
- ```
  exp ::= ...
    | "when" exp? {
        entry+
      }
  entry ::=
      exp -> stm
    | ["else" -> exp]?
  ```
- ```
  exp ::= ...
    | "when" exp? {
        entry+
        ["else" -> exp]?
      }
  entry ::=
      exp -> exp
  ```
- ```
  exp ::= ...
    | "when" exp? {
        entry+
        ["else" -> stm]?
      }
  entry ::=
      exp -> stm
  ```
- same as the bottom-left but with ”stm” in the right part of ”else”
  ```

Question 3. What is the EBNF for this construct?

☐
```
exp ::= ...
  | "when" exp? {
      exp+ -> exp+
    }
  | "else"
```

☐
```
stm ::= ...
  | "when" exp? {
      exp+ -> stm+
    }
  | "else"
```

☐
```
exp ::= ...
  | "when" exp? {
      entry+
    }
entry ::=
     exp -> stm
  |  ["else" -> exp]?
```

☑
```
exp ::= ...
  | "when" exp? {
      entry+
      ["else" -> exp]?
    }
entry ::=
     exp -> exp
```

☐
```
exp ::= ...
  | "when" exp? {
      entry+
      ["else" -> stm]?
    }
entry ::=
     exp -> stm
```

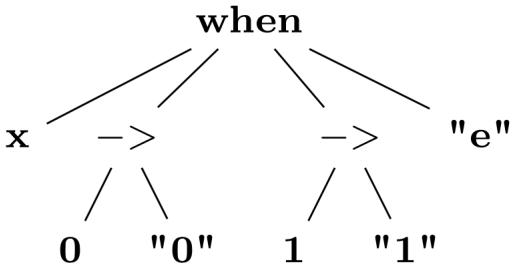
☐ same as the bottom-left but with ”stm” in the right part of ”else”

Question 4. Draw the AST for this snippet

```
when (x) {
  0 -> "0"
  1 -> "1"
  else -> "e"
}
```

Question 4. Draw the AST for this snippet

```
when (x) {
  0 -> "0"
  1 -> "1"
  else -> "e"
}
```



Question 5.

What should the binder do with this new construct

- ☐ Bind left part of ”->” to its right part
- ☐ Binds ”breaks” to ”while”/”for”
- ☐ Bind ”else” with ”when”
- ☐ Nothing
- ☐ Bind variables uses to their declarations

Question 5.

What should the binder do with this new construct

- ☐ Bind left part of ”->” to its right part
- ☒ Binds ”breaks” to ”while”/”for”
- ☐ Bind ”else” with ”when”
- ☐ Nothing
- ☑ Bind variables uses to their declarations

Question 5. What is the maximum level of the symbol table (including the global table) of this example?

```
when  {
  x == 0 -> when {
    y == 1  -> 2
    else -> 3
  }
  else -> when {
    y == 1 -> 2
    else -> 3
  }
}
```

- ☐ 0
- ☐ 1
- ☐ 2
- ☐ 3
- ☐ 4

Question 5. What is the maximum level of the symbol table (including the global table) of this example?

```
when  {
  x == 0 -> when {
    y == 1  -> 2
    else -> 3
  }
  else -> when {
    y == 1 -> 2
    else -> 3
  }
}
```

- ☐ 0
- ☑ 1
- ☐ 2
- ☐ 3
- ☐ 4

Question 7.

What is the purpose of the Visitor pattern?

- ☐ Simulate multi-methods
- ☐ Walk an AST
- ☐ Have template at runtime
- ☐ Modify the order of dynamic dispatch

Question 7.

What is the purpose of the Visitor pattern?

- [x] Simulate multi-methods
- [ ] Walk an AST
- [ ] Have template at runtime
- [ ] Modify the order of dynamic dispatch

Question 8.

What is the inference rule for :

`when ( σ ) { α -> β else -> ϵ }`

- $\dfrac{\Gamma \vdash \sigma : \mathsf{T1} \wedge \Gamma \vdash \alpha : \mathsf{T2} \wedge \Gamma \vdash \beta : \mathsf{T3} \wedge \Gamma \vdash \epsilon : \mathsf{T4}}{\Gamma \vdash \mathsf{when(..)\{...\}}\text{: } \mathsf{T3} \vee \mathsf{T4}}$
- $\dfrac{\Gamma \vdash \sigma : \mathsf{T1} \wedge \Gamma \vdash \alpha : \mathsf{T1} \wedge \Gamma \vdash \beta : \mathsf{T2} \wedge \Gamma \vdash \epsilon : \mathsf{T2}}{\Gamma \vdash \mathsf{when(..)\{...\}}\text{: } \mathsf{T2}}$
- $\dfrac{\Gamma \vdash \sigma : \mathsf{T1} \wedge \Gamma \vdash \alpha : \mathsf{T1} \wedge \Gamma \vdash \beta : \mathsf{T2} \wedge \Gamma \vdash \epsilon : \mathsf{T3}}{\Gamma \vdash \mathsf{when(..)\{...\}}\text{: } \mathsf{T2} \vee \mathsf{T3}}$
- $\dfrac{\Gamma \vdash \sigma : \mathsf{T1} \wedge \Gamma \vdash \alpha : \mathsf{T2} \wedge \Gamma \vdash \beta : \mathsf{T3} \wedge \Gamma \vdash \epsilon : \mathsf{T4}}{\Gamma \vdash \mathsf{when(..)\{...\}}\text{: } \mathsf{LUB(T3,T4)}}$
- $\dfrac{\Gamma \vdash \sigma : \mathsf{T1} \wedge \Gamma \vdash \alpha : \mathsf{T1} \wedge \Gamma \vdash \beta : \mathsf{T2} \wedge \Gamma \vdash \epsilon : \mathsf{T3}}{\Gamma \vdash \mathsf{when(..)\{...\}}\text{: } \mathsf{LUB(T2,T3)}}$

Question 8.

What is the inference rule for :

`when ( σ ) { α -> β else -> ϵ }`

- ☐ $$\frac{\Gamma \vdash \sigma : \mathsf{T1} \wedge \Gamma \vdash \alpha : \mathsf{T2} \wedge \Gamma \vdash \beta : \mathsf{T3} \wedge \Gamma \vdash \epsilon : \mathsf{T4}}{\Gamma \vdash \mathsf{when(..)\{...\}} : \mathsf{T3} \vee \mathsf{T4}}$$
- ☐ $$\frac{\Gamma \vdash \sigma : \mathsf{T1} \wedge \Gamma \vdash \alpha : \mathsf{T1} \wedge \Gamma \vdash \beta : \mathsf{T2} \wedge \Gamma \vdash \epsilon : \mathsf{T2}}{\Gamma \vdash \mathsf{when(..)\{...\}} : \mathsf{T2}}$$
- ☐ $$\frac{\Gamma \vdash \sigma : \mathsf{T1} \wedge \Gamma \vdash \alpha : \mathsf{T1} \wedge \Gamma \vdash \beta : \mathsf{T2} \wedge \Gamma \vdash \epsilon : \mathsf{T3}}{\Gamma \vdash \mathsf{when(..)\{...\}} : \mathsf{T2} \vee \mathsf{T3}}$$
- ☐ $$\frac{\Gamma \vdash \sigma : \mathsf{T1} \wedge \Gamma \vdash \alpha : \mathsf{T2} \wedge \Gamma \vdash \beta : \mathsf{T3} \wedge \Gamma \vdash \epsilon : \mathsf{T4}}{\Gamma \vdash \mathsf{when(..)\{...\}} : \mathsf{LUB(T3,T4)}}$$
- ☑ $$\frac{\Gamma \vdash \sigma : \mathsf{T1} \wedge \Gamma \vdash \alpha : \mathsf{T1} \wedge \Gamma \vdash \beta : \mathsf{T2} \wedge \Gamma \vdash \epsilon : \mathsf{T3}}{\Gamma \vdash \mathsf{when(..)\{...\}} : \mathsf{LUB(T2,T3)}}$$

Question 9. Unsugar the previous code into C++

```
if (σ==α)
      β;
else
      ϵ;
```

```
switch (σ) {
   case α:  β; break;
   default:  ϵ; break;
}
```

```
[&](){
  if (σ==α) β;
  else if (σ!=α) ϵ;
}
```

```
[&](auto y){
  if (y==α) return β;
  else  return ϵ;
}(σ)
```

Question 9. Unsugar the previous code into C++

☐
```
if (σ==α)
      β;
else
      ϵ;
```

☐
```
switch (σ) {
   case α:  β; break;
   default:  ϵ; break;
}
```

☐
```
[&](){
  if (σ==α) β;
  else if (σ!=α) ϵ;
}
```

☑
```
[&](auto y){
  if (y==α) return β;
  else  return ϵ;
}(σ)
```

Summary

Complete example that covers front-end!

Few more questions about generalization