**Script** **generated by TTT**

Title: Petter: Compiler Construction (25.06.2020) - 48: Structural Subtyping

Date: Fri Jun 26 10:18:13 CEST 2020

Duration: 31:49 min

Pages: 11

## Subtyping $\leq$

On the arithmetic basic types **`char`**, **`int`**, **`long`**, etc. there exists a rich *subtype* hierarchy

### Subtypes

$t_1 \leq t_2$, means that the values of type $t_1$

1. form a subset of the values of type $t_2$;
2. can be converted into a value of type $t_2$;
3. fulfill the requirements of type $t_2$;
4. are assignable to variables of type $t2$.

Example:
assign smaller type (fewer values) to larger type (more values)

$$
\begin{array}{ll}
t_1 & x; \\
t_2 & y; \\
y = x; &
\end{array}
$$



## Subtyping $\leq$

On the arithmetic basic types **`char`**, **`int`**, **`long`**, etc. there exists a rich *subtype* hierarchy

### Subtypes

$t_1 \leq t_2$, means that the values of type $t_1$

1. form a subset of the values of type $t_2$;
2. can be converted into a value of type $t_2$;
3. fulfill the requirements of type $t_2$;
4. are assignable to variables of type $t2$.

Example:
assign smaller type (fewer values) to larger type (more values)

$$
\begin{array}{ll}
t_1 & x; \\
t_2 & y; \\
y = x; & \\
t_1 \leq t_2 &
\end{array}
$$



## Subtyping $\leq$

On the arithmetic basic types **`char`**, **`int`**, **`long`**, etc. there exists a rich *subtype* hierarchy

### Subtypes

$t_1 \leq t_2$, means that the values of type $t_1$

1. form a subset of the values of type $t_2$;
2. can be converted into a value of type $t_2$;
3. fulfill the requirements of type $t_2$;
4. are assignable to variables of type $t2$.

Example:
assign smaller type (fewer values) to larger type (more values)

$$
\begin{array}{l}
\texttt{int}\ x; \\
\texttt{double}\ y; \\
y = x; \\
\texttt{int} \leq \texttt{double}
\end{array}
$$

## Example: Subtyping

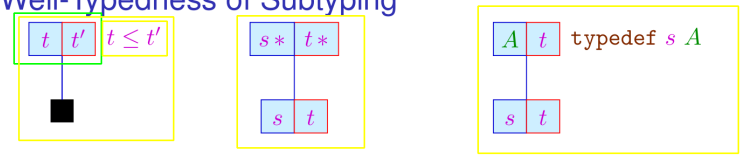
Extending the subtype relationship to more complex types, observe:

```
string extractInfo( struct { string info; } x) {
  return x.info;
}
```

- we want `extractInfo` to be applicable to all argument structures that return a `string` typed field for accessor `info`
- the idea of subtyping on values is related to subclasses
- we use deduction rules to describe when $t_1 \leq t_2$ should hold...

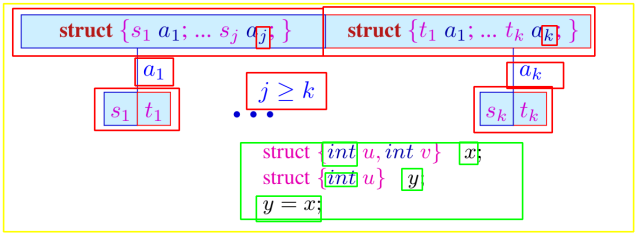


## Rules for Well-Typedness of Subtyping



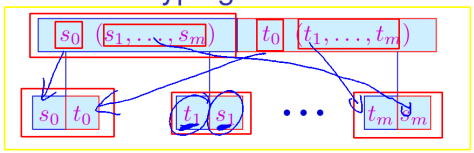

Sub

Sup

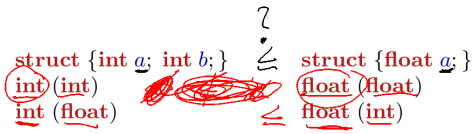





## Rules and Examples for Subtyping

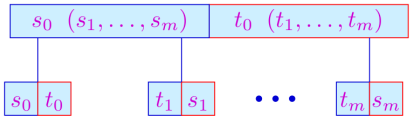



### Examples:






## Rules and Examples for Subtyping




### Examples:

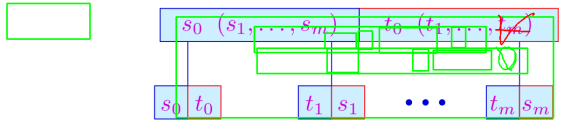
| | |
|---|---|
| struct {int $a$; int $b$;} | struct {float $a$;} |
| int (int) | float (float) |
| int (float) | float (int) |

**Definition**

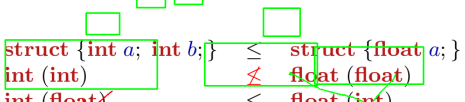
Given two function types in subtype relation $s_0(s_1, \ldots s_n) \leq t_0(t_1, \ldots t_n)$ then we have

- co-variance of the return type $s_0 \leq t_0$ and
- contra-variance of the arguments $s_i \geq t_i$ für $1 < i \leq n$

# Rules and Examples for Subtyping




Examples:




**Definition**

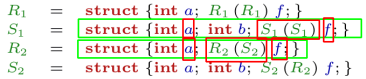
Given two function types in subtype relation $s_0(s_1, \ldots s_n) \leq t_0(t_1, \ldots t_n)$ then we have

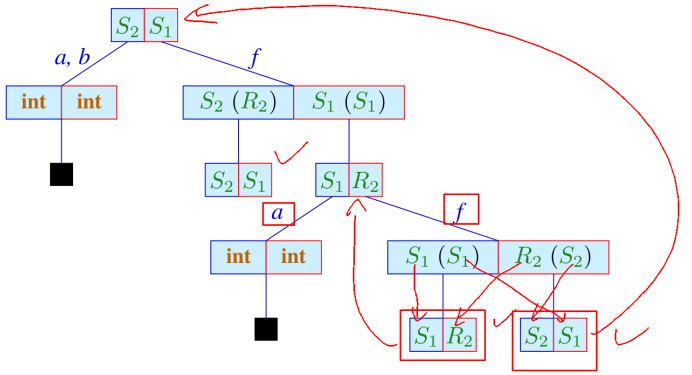
- co-variance of the return type $s_0 \leq t_0$ and
- contra-variance of the arguments $s_i \geq t_i$ für $1 < i \leq n$



# Subtypes: Application of Rules (II)

Check if $S_2 \leq S_1$:

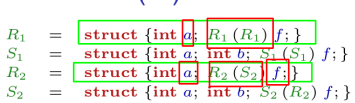


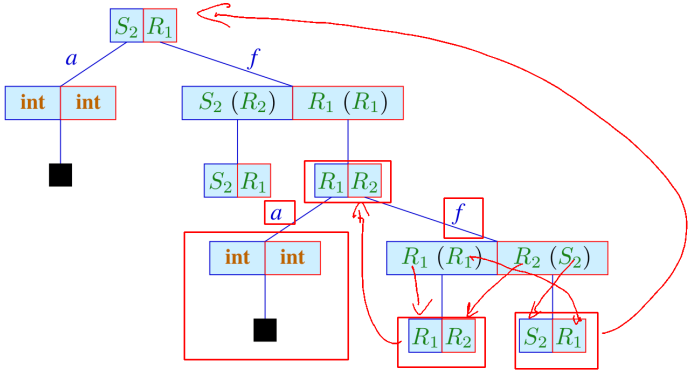






# Subtypes: Application of Rules (III)

Check if $S_2 \leq R_1$:






# Discussion

- for presentational purposes, proof trees are often abbreviated by omitting deductions within the tree
- structural sub-types are very powerful and can be quite intricate to understand
- Java generalizes structs to objects/classes where a sub-class $A$ inheriting form base class $O$ is a subtype $A \leq O$
- subtype relations between classes must be explicitly declared