

Objective Caml version 3.06

(\* Die Deklarationen aus "subtyping\_rules.ml" werden als bekannt vorausgesetzt \*)

```
# let f1 o = o#a;;
val f1 : < a : 'a; .. > -> 'a = <fun>

# let f2 o = o#a + 1;;
val f2 : < a : int; .. > -> int = <fun>

# let f3 o = if o#a then o else o;;
val f3 : (< a : bool; .. > as 'a) -> 'a = <fun>

# let g1 o = o#x#a;;
val g1 : < x : < a : 'a; .. >; .. > -> 'a = <fun>

# let g2 o = o#x#a + o#y;;
val g2 : < x : < a : int; .. >; y : int; .. > -> int = <fun>

# let g3 o = if o#x#a > 0 && o#y < 0 then (o, o#x) else (o,o#x);;
val g3 : (< x : < a : int; .. > as 'b; y : int; .. > as 'a) -> 'a * 'b =
  <fun>

# g1 o_t;;
- : int = 0

# g1 o_s;;
- : int = 0

# g2 o_t;;
Characters 3-6:
  g2 o_t;;
  ^^^
This expression has type t = < x : < a : int > > but is here used with type
  < x : < a : int; .. >; y : int; .. >
  Only the second object type has a method y

# g2 o_s;;
- : int = 2

# g3 o_t;;
Characters 3-6:
  g3 o_t;;
  ^^^
This expression has type t = < x : < a : int > > but is here used with type
  < x : < a : int; .. >; y : int; .. >
  Only the second object type has a method y

# g3 o_s;;
- : s * < a : int; b : int > = (<obj>, <obj>)

# let g1 (o: < x : < a : 'a; .. >; .. >) = o#x#a;;
val g1 : < x : < a : 'a; .. >; .. > -> 'a = <fun>

# let g2 (o: < x : < a : int; .. >; y : int; .. >) = o#x#a + o#y;;
val g2 : < x : < a : int; .. >; y : int; .. > -> int = <fun>

# let g2 (o: < x : < a : 'a; .. >; .. >) = o#x#a + o#y;;
val g2 : < x : < a : int; .. >; y : int; .. > -> int = <fun>

# let g2 (o: < x : < a : 'a; .. > >) = o#x#a + o#y;;
Characters 45-46:
  let g2 (o: < x : < a : 'a; .. > >) = o#x#a + o#y;;
  ^
This expression has type < x : < a : int; .. > >
  It has no method y

# let g3 (o: < x : < a : int; .. > as 'b; y : int; .. > as 'a) =
  if o#x#a > 0 && o#y < 0 then (o, o#x) else (o,o#x);;
  val g3 : (< x : < a : int; .. > as 'b; y : int; .. > as 'a) -> 'a * 'b =
  <fun>

# let g3 (o: < x : < a : int; .. >; y : int; .. >) =
  if o#x#a > 0 && o#y < 0 then (o, o#x) else (o,o#x);;
  val g3 : (< x : < a : int; .. > as 'b; y : int; .. > as 'a) -> 'a * 'b =
  <fun>

# let g1 (o: c_t) = o#x#a;;
```

```

val g1 : c_t -> int = <fun>

# g1 o_t;;
- : int = 0

# g1 o_s;;
Characters 3-6:
  g1 o_s;;
  ^^^
This expression has type s = < x : < a : int; b : int >; y : int >
but is here used with type c_t = < x : c_a >
Only the first object type has a method y

# let g1 (o: #c_t) = o#x#a;;
val g1 : #c_t -> int = <fun>

# g1 o_t;;
- : int = 0

# g1 o_s;;
Characters 3-6:
  g1 o_s;;
  ^^^
This expression has type s = < x : < a : int; b : int >; y : int >
but is here used with type < x : c_a; y : int >
Type < a : int; b : int > is not compatible with type c_a = < a : int >
Only the first object type has a method b

# let g1 (o: <x: #c_a; ..>) = o#x#a;;
val g1 : < x : #c_a; .. > -> int = <fun>

# g1 o_t;;
- : int = 0

# g1 o_s;;
- : int = 0

#

```