Technische Universität München
Institut für Informatik
Prof. Tobias Nipkow, Ph.D.
Kevin Kappelmann, Lukas Stevens

WS 24/25
23.1.2024

Semantics of Programming Languages

Exercise Sheet 13

Exercise 13.1 Termination for sign analysis

Recall the abstract interpreter from the last sheet:

```
datatype sign = Pos | Zero | Neg | Any

instantiation sign :: order
begin
definition less_eq_sign where "x ≤ y = (y = Any ∨ x=y)"
definition less_sign where "x < y = (x ≤ y ∧ ¬ y ≤ (x::sign))"

instance by standard (auto simp: less_eq_sign_def less_sign_def)
end

instantiation sign :: semilattice_sup_top
begin
definition sup_sign where "x ⊔ y = (if x = y then x else Any)"
definition top_sign where "⊤ = Any"

instance by standard (auto simp: less_eq_sign_def sup_sign_def top_sign_def)
end

fun γ_sign :: "sign ⇒ val set" where
  "γ_sign Neg = {i. i < 0}"
| "γ_sign Pos  = {i. i > 0}"
| "γ_sign Zero  = {0}"
| "γ_sign Any = UNIV"

fun num_sign :: "val ⇒ sign" where
"num_sign i = (if i = 0 then Zero else if i > 0 then Pos else Neg)"

fun plus_sign :: "sign ⇒ sign ⇒ sign" where
  "plus_sign y Zero = y"
| "plus_sign Zero y  = y"
| "plus_sign x y = (if x = y then x else Any)"

global_interpretation Val_semilattice
  where γ = γ_sign and num' = num_sign and plus' = plus_sign
```

proof (*standard*, *goal_cases*)
  case (*4* _ *a1* _ *a2*) **thus** *?case*
    by (*induction a1 a2 rule*: *plus_sign.induct*) (*auto simp add*:*mod_add_eq*)
qed (*auto simp*: *less_eq_sign_def top_sign_def*)

global_interpretation *Abs_Int*
  where $\gamma = \gamma\_sign$ **and** $num' = num\_sign$ **and** $plus' = plus\_sign$
  defines $aval\_sign = aval'$ **and** $step\_sign = step'$ **and** $AI\_sign = AI$
  ..

Define a measure function on the abstract domain, which can be used to prove that the analysis always terminates. Define a function *m_sign* from the sign domain into the natural numbers such that

- $x < y \Longrightarrow m\_sign\ x > m\_sign\ y$
- $m\_sign\ x \leq h\_sign$

where *h_sign* is the height of the sign domain.

abbreviation *h_sign* :: *nat*
fun *m_sign* :: *"sign ⇒ nat"*

global_interpretation *Abs_Int_mono*
  where $\gamma = \gamma\_sign$ **and** $num' = num\_sign$ **and** $plus' = plus\_sign$
global_interpretation *Abs_Int_measure*
where $\gamma = \gamma\_sign$ **and** $num' = num\_sign$ **and** $plus' = plus\_sign$
and $m = m\_sign$ **and** $h = h\_sign$

Exercise 13.2 Inverse Analysis

Consider a similar analysis based on this abstract domain:

datatype *sign0 = None | Neg | Pos0 | Any*

fun γ_0 :: *"sign0 ⇒ val set"* **where**
"γ_0 None = {}" |
"γ_0 Neg = {i. i < 0}" |
"γ_0 Pos0 = {i. i ≥ 0}" |
"γ_0 Any = UNIV"

Define inverse analyses for "+" and "<" and prove the required correctness properties:

fun *inv_plus'* :: *"sign0 ⇒ sign0 ⇒ sign0 ⇒ sign0 * sign0"*
lemma
  "⟦ inv_plus' a a1 a2 = (a1',a2'); i1 ∈ γ_0 a1; i2 ∈ γ_0 a2; i1+i2 ∈ γ_0 a ⟧
  ⟹ i1 ∈ γ_0 a1' ∧ i2 ∈ γ_0 a2' "
fun *inv_less'* :: *"bool ⇒ sign0 ⇒ sign0 ⇒ sign0 * sign0"*
lemma
  "⟦ inv_less' bv a1 a2 = (a1',a2'); i1 ∈ γ_0 a1; i2 ∈ γ_0 a2; (i1<i2) = bv ⟧
  ⟹ i1 ∈ γ_0 a1' ∧ i2 ∈ γ_0 a2'"

Homework 13.1 Inverse Analysis for the Extended Reals

Submission until Wednesday, Jan 29, 23:59pm.

Extend the abstract interpreter from the last sheet with an inverse analysis. We skip the technical part, so you only need to define the inverse operations:

fun *inv_plus'* :: *"bounds ⇒ bounds ⇒ bounds ⇒ (bounds * bounds)"*
fun *inv_less'* :: *"bool option ⇒ bounds ⇒ bounds ⇒ (bounds * bounds)"*

They should be as precise as possible, i.e., the result minimal. For example:

value *"inv_plus' (B {NaN}) (B {∞⁺}) (B {∞⁻, Real}) = (B {∞⁺}, B {∞⁻})"*
value *"inv_plus' (B {∞⁺,∞⁻}) (B {∞⁺, NaN}) (B {∞⁻, Real, ∞⁺}) = (B {∞⁺}, B {∞⁺, Real})"*
value *"inv_less' (Some True) (B {∞⁺, ∞⁻}) (B {∞⁺}) = (B {∞⁻}, B {∞⁺})"*

Prove them sound!

lemma *inv_plus'_sound*:
  assumes *"inv_plus' (B A) (B A1) (B A2) = (B A1', B A2')"*
    and *"i1 ∈ γ_bounds (B A1)"*
    and *"i2 ∈ γ_bounds (B A2)"*
    and *"(case (i1,i2) of*
        (None, _) ⇒ None ∈ γ_bounds (B A)
      | (_, None) ⇒ None ∈ γ_bounds (B A)
      | (Some PInfty, Some MInfty) ⇒ None ∈ γ_bounds (B A)
      | (Some MInfty, Some PInfty) ⇒ None ∈ γ_bounds (B A)
      | (Some i1, Some i2) ⇒ Some (i1+i2) ∈ γ_bounds (B A))"
  shows *"i1 ∈ γ_bounds (B A1') ∧ i2 ∈ γ_bounds (B A2')"*

lemma *inv_less'_sound*:
  assumes *"inv_less' bv (B A1) (B A2) = (B A1',B A2')"*
    and *"i1 ∈ γ_bounds (B A1)"*
    and *"i2 ∈ γ_bounds (B A2)"*
    and *"(case (i1,i2) of*
        (None, _) ⇒ None = bv
      | (_, None) ⇒ None = bv
      | (Some i1, Some i2) ⇒ Some (i1<i2) = bv)"
  shows *"i1 ∈ γ_bounds (B A1') ∧ i2 ∈ γ_bounds (B A2')"*