Technische Universität München
Institut für Informatik
Prof. Tobias Nipkow, Ph.D.
Andrei Popescu, Peter Lammich

WS 2012/13
15. 1. 2013

Semantics of Programming Languages

Exercise Sheet 12

In this exercise, you will prove some properties of Galois connections—these provide a general framework for the abstract interpretation concretization functions, such as the for parity analysis from the lectrures.

We start by defining a slightly enriched version of the parity domain from the lectures.

datatype *parity* = *Even* | *Odd* | *Either* | *Emp*

Recall that *Even*, *Odd* and *Either* are aimed at representing information about sets of integers: consisting of only even numbers, only odd numbers, and either odd or even numbers, respectively. We have added a fourth element, *Emp*, covering the case of a set both consisting of only even numbers and consisting of only odd number—the only such set is the empty set.

Moreover, recall that we model the notion of containing more information as an order relation, here denoted $\leq$. Thus, e.g., we need to set $Even \leq Either$.

Your first task is to define $\leq$ on the elements of *parity* and then register *parity* as a partial order.

instantiation *parity* :: *order*

$\leq$ is more than a partial order: it is lattice, even a bounded one, and even a complete one. Climb the type-class hierarchy with *parity* by defining the necessary operators and proving the necessary facts.

instantiation *parity* :: *lattice*

instantiation *parity* :: *bounded_lattice*

instantiation *parity* :: *complete_lattice*

Extend the concretization function from the lectures by mapping *Emp* to the empty set:

fun γ_*parity* :: *"parity* $\Rightarrow$ *int set"* **where**

Think of how one came up the domain *parity*: the four values of *parity* represent degrees of knowledge about the parity of the integers in given sets. This intuition can be modelled by a so-called abstraction function:

definition α_*parity* :: *"int set* $\Rightarrow$ *parity"* **where**

Sometimes the desired concretization function is definable from an abstraction function—this is the case here:

lemma *γ_parity_α_parity*: *"γ_parity a = ⋃ {S . α_parity S ≤ a}"*

Intuitively, we read *α_parity S ≤ x* as *x approximates S*. (Indeed, according to the intuitive reading of ≤, *α_parity S* contains less information than *x*.) Then *γ_parity x* can be taken to be the largest set approximated by *x*.

Dually, *α_parity* can be obtained from *γ_parity*, defining *α_parity S* to be the best approximation of *S* w.r.t. *γ_parity*:

lemma *α_parity_γ_parity*: *"α_parity S = Inf {a . S ⊆ γ_parity a}"*

The above properties can be obtained more abstractly, using the concept of Galois connection:

definition *galois* ::
"('a::complete_lattice ⇒ 'c::complete_lattice) ⇒ ('c ⇒ 'a) ⇒ bool"
where
"galois α γ ≡ ∀ c a. α c ≤ a ⟷ c ≤ γ a"

Galois connections postulate the ideal relationship between a concretization and an abstraction: *x* approximates the abstraction of *s* iff the concretization of *x* approximates *s*.

Show that *α_parity* and *γ_parity* form a Galois connection:

lemma *"galois α_parity γ_parity"*

Prove the following consequences of the Galois connection property, including monotonicity of its components and the very facts we have proved about *α_parity* and *γ_parity*:

lemma *γα_infl*: **assumes** *"galois α γ"* **shows** *"c ≤ γ(α c)"*

lemma *αγ_defl*: **assumes** *"galois α γ"* **shows** *"α(γ a) ≤ a"*

lemma *γ_mono*: **assumes** *"galois α γ"* **shows** *"mono γ"*

lemma *α_mono*: **assumes** *"galois α γ"* **shows** *"mono α"*

lemma *γ_α*: **assumes** *"galois α γ"* **shows** *"γ a = Sup {c . α c ≤ a}"*

lemma *α_γ*: **assumes** *"galois α γ"* **shows** *"α c = Inf {a . c ≤ γ a}"*