

GraalVM Stamp Theory

January 8, 2022

Abstract

The GraalVM compiler uses stamps to track type and range information during program analysis. Type information is recorded by using distinct subclasses of the abstract base class **Stamp**, i.e. **IntegerStamp** is used to represent an integer type. Each subclass introduces facilities for tracking range information. Every subclass of the **Stamp** class forms a lattice, together with an arbitrary top and bottom element each sublattice forms a lattice of all stamps. This Isabelle/HOL theory models stamps as instantiations of a lattice.

Contents

1	Stamps: Type and Range Information	3
1.1	Void Stamp	3
1.2	Stamp Lattice	4
1.2.1	Stamp Order	4
1.2.2	Stamp Join	5
1.2.3	Stamp Meet	6
1.2.4	Stamp Bounds	6
1.3	Java Stamp Methods	8
1.4	Mapping to Values	8
1.5	Generic Integer Stamp	9

1 Stamps: Type and Range Information

```
theory StampLattice
imports
  Values
  HOL.Lattices
begin
```

1.1 Void Stamp

The VoidStamp represents a type with no associated values. The VoidStamp lattice is therefore a simple single element lattice.

```
datatype void =
  VoidStamp
```

```
instantiation void :: order
begin
```

```
definition less-eq-void :: void  $\Rightarrow$  void  $\Rightarrow$  bool where
  less-eq-void a b = True
```

```
definition less-void :: void  $\Rightarrow$  void  $\Rightarrow$  bool where
  less-void a b = False
```

```
instance
  <proof>
```

```
end
```

```
instantiation void :: semilattice-inf
begin
```

```
definition inf-void :: void  $\Rightarrow$  void  $\Rightarrow$  void where
  inf-void a b = VoidStamp
```

```
instance
  <proof>
```

```
end
```

```
instantiation void :: semilattice-sup
begin
```

```
definition sup-void :: void  $\Rightarrow$  void  $\Rightarrow$  void where
  sup-void a b = VoidStamp
```

```
instance
  <proof>
```

end

instantiation *void* :: *bounded-lattice*
begin

definition *bot-void* :: *void* **where**
bot-void = *VoidStamp*

definition *top-void* :: *void* **where**
top-void = *VoidStamp*

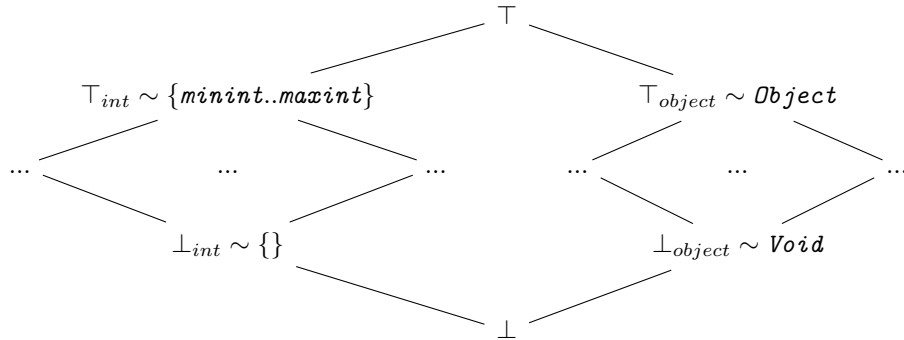
instance
<proof>

end

Definition of the stamp type

datatype *stamp* =
intstamp int64 int64 — Type: Integer; Range: Lower Bound & Upper Bound

1.2 Stamp Lattice



1.2.1 Stamp Order

Defines an ordering on the stamp type.

One stamp is less than another if the valid values for the stamp are a strict subset of the other stamp.

instantiation *stamp* :: *order*
begin

fun *less-eq-stamp* :: *stamp* ⇒ *stamp* ⇒ *bool* **where**
less-eq-stamp (*intstamp l1 u1*) (*intstamp l2 u2*) = (*{l1..u1}* ⊆ *{l2..u2}*)

fun *less-stamp* :: *stamp* ⇒ *stamp* ⇒ *bool* **where**
less-stamp (*intstamp l1 u1*) (*intstamp l2 u2*) = (*{l1..u1}* ⊂ *{l2..u2}*)

```

lemma less-le-not-le:
  fixes  $x\ y :: \text{stamp}$ 
  shows  $(x < y) = (x \leq y \wedge \neg y \leq x)$ 
   $\langle \text{proof} \rangle$ 

```

```

lemma order-refl:
  fixes  $x :: \text{stamp}$ 
  shows  $x \leq x$ 
   $\langle \text{proof} \rangle$ 

```

```

lemma order-trans:
  fixes  $x\ y\ z :: \text{stamp}$ 
  shows  $x \leq y \implies y \leq z \implies x \leq z$ 
   $\langle \text{proof} \rangle$ 

```

```

lemma antisym:
  fixes  $x\ y :: \text{stamp}$ 
  shows  $x \leq y \implies y \leq x \implies x = y$ 
   $\langle \text{proof} \rangle$ 

```

```

instance
   $\langle \text{proof} \rangle$ 
end

```

1.2.2 Stamp Join

Defines the *join* operation for stamps.

For any two stamps, the *join* is defined as the intersection of the valid values for the stamp.

```

instantiation stamp :: semilattice-inf
begin

```

```

notation inf (infix  $\sqcap$  65)

```

```

fun inf-stamp :: stamp  $\Rightarrow$  stamp  $\Rightarrow$  stamp where
  inf-stamp (intstamp  $l1\ u1$ ) (intstamp  $l2\ u2$ ) = intstamp ( $\max\ l1\ l2$ ) ( $\min\ u1\ u2$ )

```

```

lemma inf-le1:
  fixes  $x\ y :: \text{stamp}$ 
  shows  $(x \sqcap y) \leq x$ 
   $\langle \text{proof} \rangle$ 

```

```

lemma inf-le2:
  fixes  $x\ y :: \text{stamp}$ 
  shows  $(x \sqcap y) \leq y$ 
   $\langle \text{proof} \rangle$ 

```

```

lemma inf-greatest:
  fixes  $x\ y\ z :: \text{stamp}$ 
  shows  $x \leq y \implies x \leq z \implies x \leq (y \sqcap z)$ 
   $\langle \text{proof} \rangle$ 

```

```

instance
   $\langle \text{proof} \rangle$ 
end

```

1.2.3 Stamp Meet

Defines the *meet* operation for stamps.

For any two stamps, the *meet* is defined as the union of the valid values for the stamp.

```

instantiation  $\text{stamp} :: \text{semilattice-sup}$ 
begin

```

```

notation sup (infix  $\sqcup$  65)

```

```

fun sup-stamp ::  $\text{stamp} \Rightarrow \text{stamp} \Rightarrow \text{stamp}$  where
  sup-stamp (intstamp  $l1\ u1$ ) (intstamp  $l2\ u2$ ) = intstamp ( $\min\ l1\ l2$ ) ( $\max\ u1\ u2$ )

```

```

lemma sup-ge1:
  fixes  $x\ y :: \text{stamp}$ 
  shows  $x \leq x \sqcup y$ 
   $\langle \text{proof} \rangle$ 

```

```

lemma sup-ge2:
  fixes  $x\ y :: \text{stamp}$ 
  shows  $y \leq x \sqcup y$ 
   $\langle \text{proof} \rangle$ 

```

```

lemma sup-least:
  fixes  $x\ y\ z :: \text{stamp}$ 
  shows  $y \leq x \implies z \leq x \implies ((y \sqcup z) \leq x)$ 
   $\langle \text{proof} \rangle$ 

```

```

instance
   $\langle \text{proof} \rangle$ 
end

```

1.2.4 Stamp Bounds

Defines the top and bottom elements of the stamp lattice.

This poses an interesting question as our stamp type is a union of the various *Stamp* subclasses, e.g. *IntegerStamp*, *ObjectStamp*, etc.

Each subclass should preferably have its own unique top and bottom ele-

ment, i.e. An *IntegerStamp* would have the top element of the full range of integers allowed by the bit width and a bottom of a range with no integers. While the *ObjectStamp* should have *Object* as the top and *Void* as the bottom element.

instantiation *stamp* :: *bounded-lattice*
begin

notation *bot* (\perp 50)
notation *top* (\top 50)

definition *width-min* :: *nat* \Rightarrow *int64* **where**
width-min *bits* = $-(2^{\wedge}(\text{bits}-1))$

definition *width-max* :: *nat* \Rightarrow *int64* **where**
width-max *bits* = $(2^{\wedge}(\text{bits}-1)) - 1$

value (*sint* (*width-min* 64), *sint* (*width-max* 64))
value *max-word*::*int64*

lemma
assumes *x* = *width-min* 64
assumes *y* = *width-max* 64
shows *sint* *x* < *sint* *y*
 \langle *proof* \rangle

Note that this definition is valid for unsigned integers only.

The bottom and top element for signed integers would be (- 9223372036854775808, 9223372036854775807).

For unsigned we have (0, 18446744073709551615).

For Java we are likely to be more concerned with signed integers. To use the appropriate bottom and top for signed integers we would need to change our definition of *less_eq* from *l1..u1* <= *l2..u2* to *sint* *l1*..*sint* *u1* <= *sint* *l2*..*sint* *u2*

We may still find an unsigned integer stamp useful. I plan to investigate the Java code to see if this is useful and then apply the changes to switch to signed integers.

definition *bot-stamp* = *intstamp* *max-word* 0
definition *top-stamp* = *intstamp* 0 *max-word*

lemma *bot-least*:
fixes *a* :: *stamp*
shows (\perp) \leq *a*
 \langle *proof* \rangle

lemma *top-greatest*:
fixes *a* :: *stamp*

shows $a \leq (\top)$
 $\langle proof \rangle$

instance
 $\langle proof \rangle$
end

1.3 Java Stamp Methods

The following are methods from the Java Stamp class, they are the methods primarily used for optimizations.

definition *is-unrestricted* :: stamp \Rightarrow bool **where**
is-unrestricted $s = (\top = s)$

fun *is-empty* :: stamp \Rightarrow bool **where**
is-empty $s = (\perp = s)$

fun *as-constant* :: stamp \Rightarrow Value option **where**
as-constant (intstamp $l\ u$) = (if (card { $l..u$ }) = 1
then Some (IntVal64 (SOME x . $x \in \{l..u\}$))
else None)

definition *always-distinct* :: stamp \Rightarrow stamp \Rightarrow bool **where**
always-distinct stamp1 stamp2 = $(\perp = (stamp1 \sqcap stamp2))$

definition *never-distinct* :: stamp \Rightarrow stamp \Rightarrow bool **where**
never-distinct stamp1 stamp2 =
 $(as-constant\ stamp1 = as-constant\ stamp2 \wedge as-constant\ stamp1 \neq None)$

1.4 Mapping to Values

fun *valid-value* :: stamp \Rightarrow Value \Rightarrow bool **where**
valid-value (intstamp $l\ u$) (IntVal64 v) = $(v \in \{l..u\})$ |
valid-value (intstamp $l\ u$) - = False

The *valid-value* function is used to map a stamp instance to the values that are allowed by the stamp.

It would be nice if there was a slightly more integrated way to perform this mapping as it requires some infrastructure to prove some fairly simple properties.

lemma *bottom-range-empty*:
 $\neg(valid-value\ (\perp)\ v)$
 $\langle proof \rangle$

lemma *join-values*:
assumes *joined* = $x\text{-stamp} \sqcap y\text{-stamp}$
shows $valid-value\ joined\ x \longleftrightarrow (valid-value\ x\text{-stamp}\ x \wedge valid-value\ y\text{-stamp}\ x)$
 $\langle proof \rangle$


```

lemma disjoint-empty:
  fixes x-stamp y-stamp :: stamp
  assumes  $\perp = x\text{-stamp} \sqcap y\text{-stamp}$ 
  shows  $\neg(\text{valid-value } x\text{-stamp } x \wedge \text{valid-value } y\text{-stamp } x)$ 
   $\langle \text{proof} \rangle$ 

```

experiment begin

A possible equivalent alternative to the definition of `less_eq`

```

fun less-eq-alt :: 'a::ord  $\times$  'a  $\Rightarrow$  'a  $\times$  'a  $\Rightarrow$  bool where
  less-eq-alt (l1, u1) (l2, u2) = ( $(\neg l1 \leq u1) \vee l2 \leq l1 \wedge u1 \leq u2$ )

```

Proof equivalence

```

lemma
  fixes l1 l2 u1 u2 :: int
  assumes  $l1 \leq u1 \wedge l2 \leq u2$ 
  shows  $\{l1..u1\} \subseteq \{l2..u2\} = ((l1 \geq l2) \wedge (u1 \leq u2))$ 
   $\langle \text{proof} \rangle$ 

```

```

lemma
  fixes l1 l2 u1 u2 :: int
  shows  $\{l1..u1\} \subseteq \{l2..u2\} = \text{less-eq-alt } (l1, u1) (l2, u2)$ 
   $\langle \text{proof} \rangle$ 
end

```

1.5 Generic Integer Stamp

Experimental definition of integer stamps generically, restricting the datatype to only allow valid ranges and the bottom integer element (`max_int..min_int`).

```

lemma
  assumes  $(x::int) > 0$ 
  shows  $(2 \wedge x)/2 = (2 \wedge (x - 1))$ 
   $\langle \text{proof} \rangle$ 

```

```

definition max-signed-int :: 'a::len word where
  max-signed-int =  $(2 \wedge (\text{LENGTH}('a) - 1)) - 1$ 

```

```

definition min-signed-int :: 'a::len word where
  min-signed-int =  $-(2 \wedge (\text{LENGTH}('a) - 1))$ 

```

```

definition int-bottom :: 'a::len word  $\times$  'a word where
  int-bottom = (max-signed-int, min-signed-int)

```

```

definition int-top :: 'a::len word  $\times$  'a word where
  int-top = (min-signed-int, max-signed-int)

```

```

lemma
  fixes  $x :: 'a::len\ word$ 
  shows  $sint\ x \leq sint\ (((2 \wedge (LENGTH('a) - 1)) - 1)::'a\ word)$ 
   $\langle proof \rangle$ 

value  $sint\ (0::1\ word)$ 
value  $sint\ (1::1\ word)$ 
value  $sint\ (((2 \wedge 0) - 1)::1\ word)$ 

value  $sint\ (((2 \wedge 31) - 1)::32\ word)$ 

lemma max-signed:
  fixes  $a :: 'a::len\ word$ 
  shows  $sint\ a \leq sint\ (max-signed-int::'a\ word)$ 
   $\langle proof \rangle$ 

lemma min-signed:
  fixes  $a :: 'a::len\ word$ 
  shows  $sint\ a \geq sint\ (min-signed-int::'a\ word)$ 
   $\langle proof \rangle$ 

value  $max-signed-int :: 32\ word$ 
value  $int-bottom::(32\ word \times 32\ word)$ 
value  $sint\ (2147483647::32\ word)$ 
value  $sint\ (2147483648::32\ word)$ 

typedef (overloaded) ( $'a::len$ ) intstamp =
  { $bounds :: ('a\ word, 'a\ word)\ prod . ((fst\ bounds) \leq s\ (snd\ bounds) \vee bounds =$ 
   $int-bottom)$ }
   $\langle proof \rangle$ 

setup-lifting type-definition-intstamp

lift-definition lower :: ( $'a::len$ ) intstamp  $\Rightarrow 'a\ word$ 
  is  $prod.fst \circ Rep-intstamp\ \langle proof \rangle$ 

lift-definition upper :: ( $'a::len$ ) intstamp  $\Rightarrow 'a\ word$ 
  is  $prod.snd \circ Rep-intstamp\ \langle proof \rangle$ 

lift-definition lower-int :: ( $'a::len$ ) intstamp  $\Rightarrow int$ 
  is  $sint \circ prod.fst\ \langle proof \rangle$ 

lift-definition upper-int :: ( $'a::len$ ) intstamp  $\Rightarrow int$ 
  is  $sint \circ prod.snd\ \langle proof \rangle$ 

```

lift-definition *range* :: ('a::len) intstamp \Rightarrow int set

is $\lambda (l, u). \{sint\ l..sint\ u\}$ $\langle proof \rangle$

lift-definition *bounds* :: ('a::len) intstamp \Rightarrow ('a word \times 'a word)

is *Rep-intstamp* $\langle proof \rangle$

lift-definition *is-bottom* :: ('a::len) intstamp \Rightarrow bool

is $\lambda x. x = int-bottom$ $\langle proof \rangle$

lift-definition *from-bounds* :: ('a::len word \times 'a word) \Rightarrow 'a intstamp

is *Abs-intstamp* $\langle proof \rangle$

instantiation *intstamp* :: (len) order

begin

definition *less-eq-intstamp* :: 'a intstamp \Rightarrow 'a intstamp \Rightarrow bool **where**

less-eq-intstamp *s1 s2* = (*range s1* \subseteq *range s2*)

definition *less-intstamp* :: 'a intstamp \Rightarrow 'a intstamp \Rightarrow bool **where**

less-intstamp *s1 s2* = (*range s1* \subset *range s2*)

value *int-bottom*::(1 word \times 1 word)

value *sint* (0::1 word)

value *sint* (1::1 word)

value *int-bottom*::(2 word \times 2 word)

value *sint* (1::2 word)

value *sint* (2::2 word)

value *sint* ((2 \wedge (LENGTH(32) - 1) - 1)::32 word) > *sint* ((- (2 \wedge (LENGTH(32) - 1)))::32 word)

lemma *bottom-is-bottom*:

assumes *is-bottom s*

shows $s \leq a$

$\langle proof \rangle$

lemma *bounds-has-value*:

fixes $x\ y :: int$

assumes $x < y$

shows *card* {*x..y*} > 0

$\langle proof \rangle$

lemma *bounds-has-no-value*:

fixes $x\ y :: int$

assumes $x < y$

shows *card* {*y..x*} = 0

```

    <proof>

lemma bottom-unique:
  fixes  $a\ s :: 'a\ \text{intstamp}$ 
  assumes is-bottom  $s$ 
  shows  $a \leq s \longleftrightarrow \text{is-bottom } a$ 
  <proof>

lemma bottom-antisym:
  assumes is-bottom  $x$ 
  shows  $x \leq y \implies y \leq x \implies x = y$ 
  <proof>

lemma int-antisym:
  fixes  $x\ y :: 'a\ \text{intstamp}$ 
  shows  $x \leq y \implies y \leq x \implies x = y$ 
  <proof>

instance
  <proof>
end

value take-bit  $\text{LENGTH}(63)\ 20::\text{int}$ 
value take-bit  $\text{LENGTH}(63)\ ((-20)::\text{int})$ 
value bit  $(20::\text{int64})\ (63::\text{nat})$ 
value bit  $((-20)::\text{int64})\ (63::\text{nat})$ 

value  $((-20)::\text{int64}) < (20::\text{int64})$ 

value take-bit  $\text{LENGTH}(63)\ ((-20)::\text{int})$ 

lift-definition  $\text{smax} :: 'a::\text{len word} \Rightarrow 'a\ \text{word} \Rightarrow 'a\ \text{word}$ 
  is  $\lambda\ a\ b.\ (\text{if } (\text{sint } a) \leq (\text{sint } b)\ \text{then } b\ \text{else } a)$  <proof>

lift-definition  $\text{smin} :: 'a::\text{len word} \Rightarrow 'a\ \text{word} \Rightarrow 'a\ \text{word}$ 
  is  $\lambda\ a\ b.\ (\text{if } (\text{sint } a) \leq (\text{sint } b)\ \text{then } a\ \text{else } b)$  <proof>

instantiation intstamp ::  $(\text{len})\ \text{semilattice-inf}$ 
begin

notation inf (infix  $\sqcap$  65)

definition join-bounds ::  $'a\ \text{intstamp} \Rightarrow 'a\ \text{intstamp} \Rightarrow ('a\ \text{word} \times 'a\ \text{word})$  where
  join-bounds  $s1\ s2 = (\text{smax } (\text{lower } s1)\ (\text{lower } s2), \text{smin } (\text{upper } s1)\ (\text{upper } s2))$ 

definition join-or-bottom ::  $'a\ \text{intstamp} \Rightarrow 'a\ \text{intstamp} \Rightarrow ('a\ \text{word} \times 'a\ \text{word})$ 
where

```

join-or-bottom $s1\ s2 = (\text{let } \text{bound} = (\text{join-bounds } s1\ s2) \text{ in}$
 if $\text{sint } (\text{fst } \text{bound}) \geq \text{sint } (\text{snd } \text{bound})$ *then* int-bottom *else* bound)

definition *inf-intstamp* $:: 'a\ \text{intstamp} \Rightarrow 'a\ \text{intstamp} \Rightarrow 'a\ \text{intstamp}$ **where**
 inf-intstamp $s1\ s2 = \text{from-bounds } (\text{join-or-bottom } s1\ s2)$

lemma *always-valid*:

fixes $s1\ s2 :: 'a\ \text{intstamp}$

shows $\text{Rep-intstamp } (\text{from-bounds } (\text{join-or-bottom } s1\ s2)) = \text{join-or-bottom } s1\ s2$

$\langle \text{proof} \rangle$

lemma *invalid-join*:

fixes $s1\ s2 :: 'a\ \text{intstamp}$

assumes $\text{bound} = \text{join-bounds } s1\ s2$

assumes $\text{sint } (\text{fst } \text{bound}) \geq \text{sint } (\text{snd } \text{bound})$

shows $\text{from-bounds } \text{int-bottom} = s1 \sqcap s2$

$\langle \text{proof} \rangle$

lemma *unfold-bounds*:

$\text{bounds } x = (\text{lower } x, \text{upper } x)$

$\langle \text{proof} \rangle$

lemma *int-inf-le1*:

fixes $x\ y :: 'a\ \text{intstamp}$

shows $(x \sqcap y) \leq x$

$\langle \text{proof} \rangle$

lemma *int-inf-le2*:

fixes $x\ y :: 'a\ \text{intstamp}$

shows $(x \sqcap y) \leq y$

$\langle \text{proof} \rangle$

lemma

assumes $x \leq y$

assumes $\text{is-bottom } y$

shows $\text{is-bottom } x$

$\langle \text{proof} \rangle$

lemma *int-inf-greatest*:

fixes $x\ y :: 'a\ \text{intstamp}$

shows $x \leq y \Longrightarrow x \leq z \Longrightarrow x \leq y \sqcap z$

$\langle \text{proof} \rangle$

instance

$\langle \text{proof} \rangle$

end

```

instantiation intstamp :: (len) semilattice-sup
begin

notation sup (infix  $\sqcup$  65)

instance  $\langle \textit{proof} \rangle$ 

end

instantiation intstamp :: (len) bounded-lattice
begin

notation bot ( $\perp$  50)
notation top ( $\top$  50)

definition bot-intstamp = int-bottom
definition top-intstamp = int-top

instance  $\langle \textit{proof} \rangle$ 

end

value sint (0::1 word)
value sint (1::1 word)

datatype Stamp =
  BottomStamp |
  TopStamp |
  VoidStamp |

  Int8Stamp 8 intstamp |
  Int16Stamp 16 intstamp |
  Int32Stamp 32 intstamp |
  Int64Stamp 64 intstamp

instantiation Stamp :: order
begin

fun less-eq-Stamp :: Stamp  $\Rightarrow$  Stamp  $\Rightarrow$  bool where
  less-eq-Stamp BottomStamp - = True |
  less-eq-Stamp - TopStamp = True |
  less-eq-Stamp VoidStamp VoidStamp = True |
  less-eq-Stamp (Int8Stamp v1) (Int8Stamp v2) = (v1  $\leq$  v2) |
  less-eq-Stamp (Int16Stamp v1) (Int16Stamp v2) = (v1  $\leq$  v2) |
  less-eq-Stamp (Int32Stamp v1) (Int32Stamp v2) = (v1  $\leq$  v2) |
  less-eq-Stamp (Int64Stamp v1) (Int64Stamp v2) = (v1  $\leq$  v2) |
  less-eq-Stamp - - = False

fun less-Stamp :: Stamp  $\Rightarrow$  Stamp  $\Rightarrow$  bool where

```

```

less-Stamp BottomStamp BottomStamp = False |
less-Stamp BottomStamp - = True |
less-Stamp TopStamp TopStamp = False |
less-Stamp - TopStamp = True |
less-Stamp VoidStamp VoidStamp = False |
less-Stamp (Int8Stamp v1) (Int8Stamp v2) = (v1 < v2) |
less-Stamp (Int16Stamp v1) (Int16Stamp v2) = (v1 < v2) |
less-Stamp (Int32Stamp v1) (Int32Stamp v2) = (v1 < v2) |
less-Stamp (Int64Stamp v1) (Int64Stamp v2) = (v1 < v2) |
less-Stamp - - = False

```

```

instance
  ⟨proof⟩
end

```

```

instantiation Stamp :: semilattice-inf
begin

```

```

notation inf (infix  $\sqcap$  65)

```

```

fun inf-Stamp :: Stamp  $\Rightarrow$  Stamp  $\Rightarrow$  Stamp where
  inf-Stamp BottomStamp - = BottomStamp |
  inf-Stamp - BottomStamp = BottomStamp |
  inf-Stamp TopStamp - = TopStamp |
  inf-Stamp - TopStamp = TopStamp |
  inf-Stamp VoidStamp VoidStamp = VoidStamp |
  inf-Stamp (Int8Stamp v1) (Int8Stamp v2) = Int8Stamp (v1  $\sqcap$  v2) |
  inf-Stamp (Int16Stamp v1) (Int16Stamp v2) = Int16Stamp (v1  $\sqcap$  v2) |
  inf-Stamp (Int32Stamp v1) (Int32Stamp v2) = Int32Stamp (v1  $\sqcap$  v2) |
  inf-Stamp (Int64Stamp v1) (Int64Stamp v2) = Int64Stamp (v1  $\sqcap$  v2)

```

```

instance
  ⟨proof⟩
end

```

```

instantiation Stamp :: semilattice-sup
begin

```

```

notation sup (infix  $\sqcup$  65)

```

```

fun sup-Stamp :: Stamp  $\Rightarrow$  Stamp  $\Rightarrow$  Stamp where
  sup-Stamp BottomStamp - = BottomStamp |
  sup-Stamp - BottomStamp = BottomStamp |
  sup-Stamp TopStamp - = TopStamp |
  sup-Stamp - TopStamp = TopStamp |
  sup-Stamp VoidStamp VoidStamp = VoidStamp |
  sup-Stamp (Int8Stamp v1) (Int8Stamp v2) = Int8Stamp (v1  $\sqcup$  v2) |
  sup-Stamp (Int16Stamp v1) (Int16Stamp v2) = Int16Stamp (v1  $\sqcup$  v2) |

```

```

    sup-Stamp (Int32Stamp v1) (Int32Stamp v2) = Int32Stamp (v1  $\sqcup$  v2) |
    sup-Stamp (Int64Stamp v1) (Int64Stamp v2) = Int64Stamp (v1  $\sqcup$  v2)

instance
  <proof>
end

instantiation Stamp :: bounded-lattice
begin

notation bot ( $\perp$  50)
notation top ( $\top$  50)

definition top-Stamp :: Stamp where
  top-Stamp = TopStamp
definition bot-Stamp :: Stamp where
  bot-Stamp = BottomStamp

instance
  <proof>
end

lemma [code]: Rep-intstamp (from-bounds (l, u)) = (l, u)
  <proof>

code-datatype Abs-intstamp

end

```