Il tipo nat per Small20

Mario Correddu

October 12, 2020

nat per Small20

- vogliamo introdurre il tipo di dato nat dei numeri interi non negativi
- vogliamo anche che nat si comporti come sottotipo di int,e che quindi ne sfrutti la struttura
- l'utilizzo di nat rispetto a int, oltre ad essere d'aiuto perchè evidenzia che si sta lavorando con numeri naturali, genera degli ulteriori vincoli per garantire la correttezza del codice
- introduciamo anche l'espressione Cast che data una espressione di tipo int o nat restituisca una espressione con lo stesso valore e di tipo int o nat, a scelta.

Sintassi

- Sintassi Astratta
  Type ::= [int] | [nat] |...
  ...
  EXP ::= ... | [Cast] Type Exp
  ...

- Sintassi Concreta: Una CFG per Small20
  Simple ::= int | nat | bool
  ...
  ExpR ::= ExpR == ExpA2 | ExpR < ExpA2 | ExpR > ExpA2 | ExpA3
  ExpA3 := ... | cast(Simple , ExpA3)

Modifica del codice

```
type tye =
    Int
    | Nat
    ...
exp = Val of ide
    ...
    | Cast of tye * exp
```

```
let rec toStringTye = (function
    | Int -> "int"
    | Nat -> "nat"
    ...
```

```
toStringExp = (function
    ...
    | Cast(tye, exp) -> "cast( "  ^ (toStringTye tye)  ^ " ,"  ^ (toStringExp exp)  ^ ")"
    ...
```

```
let isSimple = ,function
    Int -> true
    | Nat -> true
    ...
```

```
let isDval = function
    DConstN (Int,(n:num)) -> true
    | DConstN (Nat, (n:num)) -> true
    ...
```

```
let toStringDval = function
    DConstN (Int,(n:num)) -> sprintf "(int, %d)" n
    | DConstN (Nat, (n: num)) -> sprintf "(nat, %d)" n
```

Comportamento

- regole su inferenza di tipi: in small20 ogni espressione che può essere trattata come nat viene dedotta come int di sottotipo nat e solo in caso l'espressione sia incompatibile con nat allora viene dedotta come int;
- nat eredita le operazioni da int modificando quando oppotuno il loro funzionamento.
- In particolare con le operazioni aritmetiche vogliamo che se i termini sono entrambi nat gli operatori restituiscano un valore di tipo nat altrimenti se lo è uno solo, viene operato un upcast da nat a int per i termini e l'operazione diventa di int.
- tuttavia con la sottrazione il primo caso non è possibile e viene quindi ridefinita per nat in modo che venga restituito un errore se il risulato è negativo

Semantica: Dclexp

$$Y_1 = \frac{<e, Y_\rho> \rightarrow_Y (t', Y_\rho) \quad t \in Simple \quad (t'=t \vee (t'=[nat] \wedge t=[int]))}{<const\ \ t\ \ I\ \ e,\ \ Y_\rho> \rightarrow_Y ([void],\ \ [I/t] \circ Y_\rho)}$$

$$Y_2 = \frac{<e, Y_\rho> \rightarrow_Y (t', Y_\rho) \quad t \in Simple \quad (t'=t \vee (t'=[nat] \wedge t=[int]))}{<var\ \ t\ \ I\ \ e,\ \ Y_\rho> \rightarrow_Y ([void],\ \ [I/[mut]\ t] \circ Y_\rho)}$$

$$E_2 = \frac{<e, Y_\rho> \rightarrow_Y (t', Y_\rho) \quad t' \neq t \quad (t' \neq [nat] \vee t \neq [int])}{<const\ \ t\ \ I\ \ e,\ \ Y_\rho> \rightarrow_Y ([terr],\ \ Y_\rho)}$$

$$E_4 = \frac{<e,\ \ Y_\rho> -> _Y (t',\ \ Y_\rho) \quad t' \neq t \quad (t' \neq [nat] \vee t \neq [int])}{<var\ \ t\ \ I\ \ e,\ \ Y_\rho> \rightarrow_Y ([terr],\ \ Y_\rho)}$$

$$D1 = \frac{\begin{array}{c}<e,\ \ (\rho, \mu)> \rightarrow <t_e,\ \ v,\ \ (\rho_e,\ \ \mu_e)> \quad t \in Simple \\ (t=t_e \vee (t_e=[nat] \wedge t=[int])) \quad [I/(t,\ \ v)] \circ \rho = \rho_I\end{array}}{<const\ \ t\ \ I\ \ e,\ \ (\rho,\ \ \mu)> \rightarrow ([void],\ \ (\rho_I,\ \ \mu_e))}$$

$$D_2 = \frac{\begin{array}{c}<e, (\rho,\ \ \mu)> \rightarrow (t_e,\ \ v\ \ (\rho,\ \ \mu_e)) \quad (t'=t \vee (t'=[nat] \wedge t=[int])) \\ \triangleright(\mu_e, 1)=(loc_t,\ \ \mu_a) \quad [I/([mut]\ t_e,\ \ loc_t)] \circ \rho = \rho_F \quad \mu_a[loc_t \leftarrow v] = \mu_F\end{array}}{<var\ \ t\ \ I\ \ e,\ \ (\rho,\mu)> \rightarrow ([void],\ \ (\rho_F,\ \ \mu_F)}$$

Per Simple= { [int], [nat], [bool]}

I nomi sono dati in modo che X_n sostituisca la definizione data in origine e $X_{n.k}$ sia una aggiunta legata a X_n

Semantica: Dclexp

```
let ysame t1 t2 = ((t1 = t2)||((t2==Nat)&&(t1==Int)));;


let rec dclSem dcl (rho,(Store(d,g)as mu)) =
      match dcl with
        | Const(ty,ide,exp) − >
          (match expSem exp (rho,mu) with
            |(te,v,(rho1,mu1)) when (isSimple te) && (ysame ty te)
              − > (let den = (function
                | (Nat, Ival n) when(n>=0) − > DConstN(Nat,n)
                ...
                |_ > raise(SystemError("dclSem:Const",ide))) in
                  let rho2 = bind rho1 ide (den(ty,v)) in (*modifica dovuta a ysame*)
                  let st2 = (rho2,mu1) in
                  (Void,st2))
        | Var(ty,ide,exp) − >
          (match expSem exp (rho,mu) with
            |(te,v,(rhoe,mue)) when (isSimple te) && ysame(ty, te)
              ...
```

Semantica: Sem_{exp} e Sem_{den}

$$Y_{5.1} = \frac{N \geq 0}{<[num]\ N,\ \sigma> \rightarrow_Y ([nat],\ Y_\rho)}$$

$$Y_5 = \frac{N < 0}{<[num]\ N,\ \sigma> \rightarrow_Y ([int],\ Y_\rho)}$$

$$Y_{11.1} = \frac{Y_\rho(I) = [mut]([arr]\ t\ N) \quad <e,\ Y_\rho> \rightarrow_Y ([nat],\ Y_\rho) \quad t \in Simple \quad InBoundDinamycCheck}{<I[\uparrow]\ e,\ Y_\rho> \rightarrow_{DY} ([mut]\ t,\ Y_\rho)}$$

$$Y_{12.1} = \frac{Y_\rho(I) = [mut]([arr]\ t\ N) \quad <e,\ Y_\rho> \rightarrow_Y ([nat],\ Y_\rho) \quad t \in Simple \quad InBoundDinamycCheck}{<I[\uparrow]\ e,\ Y_\rho> \rightarrow_Y (t,\ Y_\rho)}$$

$$E_{11} = \frac{Y_\rho(I) = [mut]([arr]\ t\ N) \quad <e,\ Y_\rho> \rightarrow_Y (t_e,\ Y_\rho) \quad t_e \notin T_N}{<I[\uparrow]\ e,\ Y_\rho> \rightarrow_{DY} ([terr],\ Y_\rho)}$$

$$E_{13.1} = \frac{Y_\rho(I) = [mut]([arr]\ t\ N) \quad <e,\ Y_\rho> \rightarrow_Y ([nat],\ Y_\rho) \quad OutofBoundDinamycCheck}{<I[\uparrow]\ e,\ Y_\rho> \rightarrow_Y ([terr],\ Y_\rho)}$$

dove $T_N = \{[nat],\ [int]\}$

Semantica: Sem_{exp} e Sem_{den}

```
expSem exp (rho,(Store(d,g)as mu)) =
    match exp with
      N n when (n>=0) -> (Nat, Ival n ,(rho, mu))
      | N n when (n<0) -> (Int,Ival n,(rho,mu))
      ...
      | Val ide -> (
        try (let den = getEnv rho ide in
        match den with
          ...
          | DConstN(Nat, n) ->
            (Nat, Ival n, (rho, mu))
          ...
      |GetArrow(ide,exp) -> (
        try (let den = getEnv rho ide in
        match den with
          |DArray(Mut(Arr(Mut t,k)),Loc a) when (isSimple t) -> (
            match expSem exp (rho,mu) with
              ...
              |(Nat,Ival n,(rhoe,mue)) when (n>=0)&&(n<k)
                -> (let loct = Loc(a+n) in
                  try (let vt = mTOe(getStore mu loct) in
                    (t,vt,(rhoe,mue))
                  ) with
                    |(UndefinedLoc( _ ,loc2)) ->
                      (let msg = "Alloc. Problems in " ^(toStringLoc loct) ^" or in " ^(toStringLoc loc2) in
                        raise( SystemErrorE ( "expSem: " ^msg )))
                    | (SystemErrorM _ ) ->
                      (let msg = "Integrity Pms in Store State" in
                      raise(SystemErrorE("expSem: "^msg))))
              ...
              |(Nat,Ival n,(rhoe,mue)) -> raise(TypeErrorE("E13.1: expSem - " ^(toStringN n)))
              |(t ,v ,(rhoe,mue)) -> raise(TypeErrorE("E11: expSem - " ^(toStringTye t)))
```

Semantica: Sem_{exp} e Sem_{den}

```
dexpSem dexp (rho,(Store(d,g)as mu)) =
    match dexp with
      ...
      | GetArrow(ide,exp) -> (
        try (let den = getEnv rho ide in match den with
          |DArray(Mut(Arr(Mut t,k)),Loc a) when (isSimple t)-> (
            match expSem exp (rho,mu) with
              |(Nat,Ival n,(rhoe,mue)) when (n>=0)&&(n<k)
                -> (let loct = Loc(a+n) in (Mut t,loct,(rhoe,mue)))
              ...
              |(Nat,Ival n,(rhoe,mue))
              -> raise(TypeErrorE("E13.1: dexpSem - " ^(toStringN n)))
              |(t ,v ,(rhoe,mue)) ->
              raise(TypeErrorE("E11: expSem - " ^(toStringTye t)))
```

Semantica: Sem_{exp} (operazioni)

$$Y_{13.1} = \frac{<e_1,\ Y_\rho> \rightarrow_Y (t_1,\ Y_\rho) \quad <e_2,\ Y_\rho> \rightarrow_Y (t_2,\ Y_\rho) \quad Y_\rho(op)=[abs]\ t\ t_1'[x]t_2' \quad op \in O_2 \quad t_1'=t_1 \quad t_1=[int] \quad t_2'=[int] \quad t_2=[nat]}{<e1\ [op]\ e2,\ Y_\rho> \rightarrow_Y (t,\ Y_\rho)}$$

$$Y_{13.2} = \frac{<e_1,\ Y_\rho> \rightarrow_Y (t_1,\ Y_\rho) \quad <e_2,\ Y_\rho> \rightarrow_Y (t_2,\ Y_\rho) \quad Y_\rho(op)=[abs]\ t\ t_1'[x]t_2' \quad op \in O_2 \quad t_2'=t_2 \quad t_2=[int] \quad t_1'=[int] \quad t_1=[nat]}{<e1\ [op]\ e2,\ Y_\rho> \rightarrow_Y (t,\ Y_\rho)}$$

$$Y_{13.3} = \frac{<e_1,\ Y_\rho> \rightarrow_Y (t_1,\ Y_\rho) \quad <e_2,\ Y_\rho> \rightarrow_Y (t_2,\ Y_\rho) \quad Y_\rho(op)=[abs]\ t\ t_1'[x]t_2' \quad op \in O_2 - \{-\} \quad t_2'=t_1' \quad t_1'=[int] \quad t_1=t_2 \quad t_1=[nat] \quad t=[int]}{<e1\ [op]\ e2,\ Y_\rho> \rightarrow_Y ([nat],\ Y_\rho)}$$

$$Y_{13.4} = \frac{<e_1,\ Y_\rho> \rightarrow_Y (t_1,\ Y_\rho) \quad <e_2,\ Y_\rho> \rightarrow_Y (t_2,\ Y_\rho) \quad Y_\rho(op)=[abs]\ t\ t_1'[x]t_2' \quad op \in O_2 - \{-\} \quad t_2'=t_1' \quad t_1'=[int] \quad t_1=t_2 \quad t_1=[nat] \quad t \neq [int]}{<e1\ [op]\ e2,\ Y_\rho> \rightarrow_Y (t,\ Y_\rho)}$$

$$E_{15} = \frac{<e_2,\ Y_\rho> \rightarrow_Y (t_2,\ Y_\rho) \quad Y_\rho(op)=[abs]\ t\ t_1'[x]t_2' \quad op \in O_2 \quad t_2' \neq t_2 \quad (t_2 \neq [nat] \vee t_2' \neq [int])}{<e1\ [op]\ e2,\ Y_\rho> \rightarrow_Y ([terr],\ Y_\rho)}$$

$$E_{14} = \frac{<e_1,\ Y_\rho> \rightarrow_Y (t_1,\ Y_\rho) \quad Y_\rho(op)=[abs]\ t\ t_1'[x]t_2' \quad op \in O_2 \quad t_1' \neq t_1 \quad (t_1 \neq [nat] \vee t_1' \neq [int])}{<e1\ [op]\ e2,\ Y_\rho> \rightarrow_Y ([terr],\ Y_\rho)}$$

$$E_{14.5} = \frac{<e_1,\ \sigma> \rightarrow \lfloor t_1,\ v_1,\ \sigma_1 \rfloor \quad <e_2,\ \sigma> \rightarrow \lfloor t_2,\ v_2,\ \sigma_2 \rfloor \quad Y_\rho(op)=[abs]\ t\ t_1'[x]t_2' \quad op=- \quad t_1=t_2 \quad t_1=[nat] \quad \overline{op}(v1,v2)=v \quad v<0}{<e1\ [op]\ e2,\ \sigma> \rightarrow \lfloor [terr],\ \sigma_2 \rfloor}$$

Semantica: Sem_{exp} (operazioni)

$$X_{9.1} = \frac{<e_1,\ \sigma> \rightarrow \lfloor t_1,\ v_1,\ \sigma_1 \rfloor \quad <e_2,\ \sigma> \rightarrow \lfloor t_2,\ v_2,\ \sigma_2 \rfloor \quad Y_\rho(op)=[abs]\ t\ t_1'[x]t_2' \quad op \in O_2 \quad t_1'=t_1 \quad t_1=[int] \quad t_2'=[int] \quad t_2=[nat] \quad \overline{op}(v1,v2)=v}{<e1\ [op]\ e2,\ \sigma> \rightarrow \lfloor t,\ v,\ \sigma_2 \rfloor}$$

$$X_{9.2} = \frac{<e_1,\ \sigma> \rightarrow \lfloor t_1,\ v_1,\ \sigma_1 \rfloor \quad <e_2,\ \sigma> \rightarrow \lfloor t_2,\ v_2,\ \sigma_2 \rfloor \quad Y_\rho(op)=[abs]\ t\ t_1'[x]t_2' \quad op \in O_2 \quad t_2'=t_2 \quad t_2=[int] \quad t_1'=[int] \quad t_1=[nat] \quad \overline{op}(v1,v2)=v}{<e1\ [op]\ e2,\ \sigma> \rightarrow \lfloor t,\ v,\ \sigma_2 \rfloor}$$

$$X_{9.3} = \frac{<e_1,\ \sigma> \rightarrow \lfloor t_1,\ v_1,\ \sigma_1 \rfloor \quad <e_2,\ \sigma> \rightarrow \lfloor t_2,\ v_2,\ \sigma_2 \rfloor \quad Y_\rho(op)=[abs]\ t\ t_1'[x]t_2' \quad op \in O_2 - \{-\} \quad t_2'=t_1' \quad t_1'=[int] \quad t_1=t_2 \quad t_1=[nat] \quad t=[int] \quad \overline{op}(v1,v2)=v}{<e1\ [op]\ e2,\ \sigma> \rightarrow \lfloor [nat],\ v,\ \sigma_2 \rfloor}$$

$$X_{9.31} = \frac{<e_1,\ \sigma> \rightarrow \lfloor t_1,\ v_1,\ \sigma_1 \rfloor \quad <e_2,\ \sigma> \rightarrow \lfloor t_2,\ v_2,\ \sigma_2 \rfloor \quad Y_\rho(op)=[abs]\ t\ t_1'[x]t_2' \quad op=- \quad t_1=t_2 \quad t_1=[nat] \quad \overline{op}(v1,v2)=v \quad v \geq 0}{<e1\ [op]\ e2,\ \sigma> \rightarrow \lfloor [nat],\ v,\ \sigma_2 \rfloor}$$

$$X_{9.4} = \frac{<e_1,\ \sigma> \rightarrow \lfloor t_1,\ v_1,\ \sigma_1 \rfloor \quad <e_2,\ \sigma> \rightarrow \lfloor t_2,\ v_2,\ \sigma_2 \rfloor \quad Y_\rho(op)=[abs]\ t\ t_1'[x]t_2' \quad op \in O_2 - \{-\} \quad t_2'=t_1' \quad t_1'=[int] \quad t_1=t_2 \quad t_1=[nat] \quad t \neq [int] \quad \overline{op}(v1,v2)=v}{<e1\ [op]\ e2,\ \sigma> \rightarrow \lfloor t,\ v,\ \sigma_2 \rfloor}$$

$$Y_{14} = \frac{<e,\ \sigma> \rightarrow_Y (t_e,\ Y_\rho) \quad Y_\rho(op)=[abs]\ t\ t' \quad (t'=t_e \vee (t=[int] \wedge t_e=[nat])) \quad op \in O_1}{<[op]\ e,\ Y_\rho> \rightarrow_Y (t,\ Y_\rho)}$$

$$X_{10} = \frac{<e,\ \sigma> \rightarrow \lfloor t_e,\ v_e, \sigma_e \rfloor \quad Y_\rho(op)=[abs]\ t\ t' \quad (t'=t_e \vee (t=[int] \wedge t_e=[nat])) \quad op \in O_1 \quad \overline{op}(v_e)=v}{<[op]\ e,\ \sigma> \rightarrow \lfloor t,\ v,\ \sigma_e \rfloor}$$

$$E_{16} = \frac{<e,\ \sigma> \rightarrow_Y (t_e,\ Y_\rho) \quad Y_\rho(op)=[abs]\ t\ t' \quad op \in O_1 \quad t' \neq t_e \quad (t=[int] \vee t_e=[nat]))}{<[op]\ e,\ Y_\rho> \rightarrow_Y (t,\ Y_\rho)}$$

Semantica: Sem_{exp} (operazioni)

```
| _ -> try ( let app = optMap exp in(
  match app with
    | (optId,2,Int,e1,e2) -> (
      let (t1,v1,sg1) = expSem e1 (rho,mu) in
        let (t2,v2,sg2) = expSem e2 sg1 in
          match (t1, t2) with
            | (Nat, Nat) -> (let r = dispatcher v1 v2 (optId,2,[Int;Int]) in
              (match optId with
                |"minus" -> ( match r with
                  |Val n when (n<0) -> raise(TypeErrorE "E14.5: ")
                  |_ -> (Nat,r,sg2))
                |_ -> (Nat,r,sg2)))
            | (Int, Nat) -> ( let r = dispatcher v1 v2 (optId,2,[t1;Int]) in (Int,r,sg2))
            | (Nat, Int) -> ( let r = dispatcher v1 v2 (optId,2,[Int;t2]) in (Int,r,sg2))
            | _ -> ( let r = dispatcher v1 v2 (optId,2,[t1;t2]) in (Int,r,sg2)))
    | (optId,2,Bool,e1,e2) -> (
      let (t1,v1,sg1) = expSem e1 (rho,mu) in
        let (t2,v2,sg2) = expSem e2 sg1 in
          ( match (t1,t2) with
            | (Nat, Nat) -> let r = dispatcher v1 v2 (optId,2,[Int;Int]) in (Bool,r,sg2)
            | (Nat, Int) -> let r = dispatcher v1 v2 (optId,2,[Int;Int]) in (Bool,r,sg2)
            | (Int, Nat) -> let r = dispatcher v1 v2 (optId,2,[Int;Int]) in (Bool,r,sg2)
            |_ -> let r = dispatcher v1 v2 (optId,2,[t1;t2]) in (Bool,r,sg2)))
    | (optId,1,tr,e1, _ ) -> (
      let (t1,v1,sg1) = expSem e1 (rho,mu) in
        match (t1,tr ) with
          | (Nat, Int) -> ( let r = dispatcher v1 v1 (optId,1,[Int]) in (tr,r,sg1))
          | _ -> ( let r = dispatcher v1 v1 (optId,1,[t1]) in (tr,r,sg1)))
```

Semantica: Sem_{exp}

$$Y_{15} = \frac{<e_1,\ Y_\rho> \rightarrow_Y (t_1,\ Y_\rho) \quad <e_r,\ Y_\rho> \rightarrow_Y (t_r,\ Y_\rho) \quad t_1{=}[Mut]\ t \quad (t{=}t_r \vee (t{=}[int] \wedge t_r{=}[nat]))}{<e1\ [=]\ e_r,\ Y_\rho> \rightarrow_Y (t,\ Y_\rho)}$$

$$E_{19} = \frac{<e_1,\ Y_\rho> \rightarrow_Y (t_1,\ Y_\rho) \quad <e_r,\ Y_\rho> \rightarrow_Y (t_r,\ Y_\rho) \quad t_1{=}[Mut]\ t \quad t{\neq}t_r \quad (t{\neq}[int] \vee t_r{\neq}[nat])}{<e1\ [=]\ e_r,\ Y_\rho> \rightarrow_Y ([terr],\ Y_\rho)}$$

$$X_{11} = \frac{<e_r,\ \sigma> \rightarrow \lfloor t_r,\ v_r,\ \sigma_r \rfloor \quad <el,\ \sigma_r> \rightarrow_{DEN} \lfloor t_l,\ loc_t,\ \sigma_l \rfloor \quad t_l{=}[mut]\ t \quad (t{=}t_r \vee (t{=}[int] \wedge t_r{=}[nat])) \quad t{\in}Simple \quad \sigma_l{=}(\rho_l,\mu_l) \quad \mu_l[loc_t \leftarrow v_r]{=}\mu_F}{<e_l[=]e_r,\sigma> \rightarrow \lfloor t, v_r, (\rho_l, \mu_F) \rfloor}$$

```
| Upd(el,er) − >
  ( let (tr,vr,sgr) = expSem er (rho,mu) in
    match (dexpSem el sgr) with
    | (Mut t,loct,(rhol,mul)) when (ysame t tr)
    ...
```

Semantica: Cast

$$Y_{27} = \frac{t \in T_N \quad <e,\ Y_\rho> \rightarrow_Y (t_e,\ Y_\rho) \quad t_e \in T_N}{<[cast]\ t\ e,\ Y_\rho> \rightarrow_Y (t, Y_\rho)}$$

$$E_{37} = \frac{t \notin T_N}{<[cast]\ t\ e,\ Y_\rho> \rightarrow_Y ([terr],\ Y_\rho)}$$

$$E_{38} = \frac{t \in T_N \quad <e,\ Y_\rho> \rightarrow_Y (t_e,\ Y_\rho) \quad t_e \notin T_N}{<[cast]\ t\ e,\ Y_\rho> \rightarrow_Y ([terr],\ Y_\rho)}$$

$$E_{39} = \frac{t=[nat] \quad <e,\ \sigma> \rightarrow \lfloor t_e,\ v_e,\ \sigma_e \rfloor \quad (t_e = t \vee t_e = [int]) \quad v_e < 0}{<[cast]\ t\ e,\ \sigma> \rightarrow \lfloor [terr],\ \sigma_e \rfloor}$$

$$X_{12} = \frac{t=[nat] \quad <e,\ \sigma> \rightarrow \lfloor t_e,\ v_e,\ \sigma_e \rfloor \quad (t_e = t \vee t_e = [int]) \quad v_e \geq 0}{<[cast]\ t\ e,\ \sigma> \rightarrow \lfloor t,\ v_e,\ \sigma_e \rfloor}$$

$$X_{13} = \frac{t=[int] \quad <e,\ \sigma> \rightarrow \lfloor t_e,\ v_e,\ \sigma_e \rfloor \quad (t_e = t \vee t_e = [nat])}{<[cast]\ t\ e,\ \sigma> \rightarrow \lfloor t,\ v_e,\ \sigma_e \rfloor}$$

```
| Cast(t, exp) -> (
  match t with
    | Nat -> (
    match expSem exp (rho, mu) with
      | (Int, Ival n,ste) when (n >=0) -> (Nat,Ival n,ste)
      | (Nat,Ival n,ste) -> (Nat,Ival n,ste)
      | (Int, Ival n,ste) when (n<0)-> (
        raise(TypeErrorE( "E39: expSem - " ^(toStringExp exp) ^" is negative")))
      | _ -> raise(TypeErrorE("E38: expSem - "  ^ (toStringExp exp))))
    | Int -> ( match expSem exp (rho, mu) with
      | (Int, Ival n,ste) -> (Int,Ival n,ste)
      | (Nat,Ival n,ste) -> (Int,Ival n,ste)
      | _ -> raise(TypeErrorE("E38: expSem - "  ^ (toStringExp exp))))
    | _ -> raise(TypeErrorE("E37: expSem - " ^(toStringExp exp))))
```

Verifica del codice ed esempi

```
let e1 = Plus(Val "x",Val "x");;
let e2 = Minus ( Cast(Int,Val "x"), Val "s" );;
let body = Seq [
      StmD(Const(Nat, "s" ,N 8));
      StmD(Var(Nat,"x", Val "s" ));
      StmD(Var(Int, "y" , Val "s" ));
      StmD(Array(Int,"A", 4));
      StmC(Cmd(Upd(GetArrow("A", Val "x"), N 3)));
      StmC(Cmd(Upd(Val "x",e1)));
      StmC(Cmd(Upd(Val "y" , e2)));
      ];;
let ex1 = Prog("EX1",body);;
printProg ex1;;
run ex1;;
```

```
Program EX1{
  final nat s = 2;
  var nat x = s;
  var int y = s;
  int array A[4];
  A[x] = 3;
  x = (x + x);
  y = (cast( int ,x) - s); }
- : unit = ()
====== Traccia del Programma EX1 ======
================================
Stack: [A/(int[4],L2); y/(int,L1); x/(nat,L0); s/(nat, 2)]
Store: [L0<-4,L1<-2,L2<-Undef,L3<-Undef,L4<-3,L5<-Undef]
================================
========== Traccia: Fine =========
- : unit = ()
```

Verifica del codice ed esempi

```
let e3 = Neg(Val "x");;
let body = Seq [
      StmD(VarN(Nat,"x"));
      StmD(Var(Int, "y" , N 4));
      StmC(Cmd(Upd(Val "x", Cast(Nat, Val "y" ))));
      StmC(Cmd(Upd(Val "y" , e3)));
      ];;
let ex2 = Prog("EX2",body);;
printProg ex2;;
run ex2;;
```

```
Program EX2{
   var nat x;
   var int y = 4;
   x = cast( nat ,y);
   y = -x;
}
- : unit = ()
====== Traccia del Programma EX2 ======
================================
Stack: [y/(int,L1); x/(nat,L0)]
Store: [L0<-4,L1<-4]
================================
========== Traccia: Fine =========
- : unit = ()
```

Verifica del codice ed esempi

```
let e4 = Minus (Val "s" , Val "s" );;
let body = Seq [
      StmD(Const(Nat, "s" ,N 8));
      StmD(VarN(Int, "y" ));
      StmC(Cmd(Upd(Val "y" ,e4)));
      ];;
let ex3 = Prog("EX3",body);;
printProg ex3;;
run ex4;;
```

```
Program EX3{
  final nat s = 8;
  var int y;
  y = (s - s);
}
- : unit = ()
====== Traccia del Programma EX3 ======
============================
Stack: [y/(int,L0); s/(nat, 8)]
Store: [L0 < - 0]
============================
========== Traccia: Fine =========
```

Verifica del codice ed esempi

```
let body = Seq [
      StmD(Var(Int, "y" , N 8));
      StmD(Var(Nat,"x", Val "y" ));
      ];;
let ex4 = Prog("EX4",body);;
printProg ex4;;
run ex4;;
```

```
Program EX4{
  var int y = 8;
  var nat x = y;
}
- : unit = ()
Exception: TypeErrorS "E36: E34/35: E34/35: E32: E4:
dclSemvar nat x = y".
```

```
let e5 = Minus (Val "s", Val "q" );;
let body = Seq [
      StmD(Const(Nat,"s",N 8));
      StmD(Const(Nat,"q",N 9));
      StmD(VarN(Int,"y"));
      StmC(Cmd(Upd(Val "y",e5)));
      ];;
let ex5 = Prog("EX5",body);;
printProg ex5;;
run ex5;;
```

```
Program EX5 {
  final nat s = 8;
  final nat q = 9;
  var int y;
  y = (s - q);
  }
- : unit = ()
Exception: TypeErrorE "E14.5: (s - q)".
```

Verifica del codice ed esempi

```
let body = Seq [
      StmD(Var(Bool,"b",B False));
      StmC(Cmd(Upd(Val "b",Cast(Bool, Val "b")))); ];;
let ex6 = Prog("EX6",body);;
printProg ex6;;
run ex5;;
```

```
Program EX6{
  var bool b = false;
  b = cast( bool ,b);
}
- : unit = ()
Exception: TypeErrorE "E37: expSem - b".
```

```
let body = Seq [
      StmD(VarN(Int, "y" ));
      StmD(Var(Bool,"b",B False));
      StmC(Cmd(Upd(Val "x",Cast(Nat,Val "b")))); ];;
let ex7 = Prog("EX7",body);;
printProg ex7;;
run ex7;;
```

```
Program EX7{
  var int y;
  var bool b = false;
  x = cast( nat ,b); }
- : unit = ()
Exception: TypeErrorE "E38: expSem - b".
```

```
let body = Seq [
      StmD(VarN(Int, "y" ));
      StmD(Var(Int,"z",N (-2)));
      StmC(Cmd(Upd(Val "y",Cast(Nat,Val "z")))); ];;
let ex8 = Prog("EX8",body);;
printProg ex8;;
run ex8;;
```

```
Program EX8 {
  var int y;
  var int z = -2;
  y = cast( nat ,z); }
- : unit = ()
Exception: TypeErrorE "E39: expSem - z ".
```