A few definitions from previous lectures...

- From lecture 8, recall the forward/bottom-up collecting semantics of arithmetic expressions:

\[
\text{Faexp} \in \text{Aexp} \mapsto \rho(\text{Env}[P]) \mapsto (\|_{\Omega}),
\]

\[
\text{Faexp}[A]R \text{ def } = \{v \mid \exists \rho \in R : \rho \vdash A \Rightarrow v\}
\]

(1)

- the collecting semantics of boolean expressions:

\[
\text{Cbexp} \in \text{Bexp} \mapsto \rho(\mathbb{R}) \mapsto \rho(\mathbb{R}),
\]

\[
\text{Cbexp}[B]R \text{ def } = \{\rho \in R \mid \rho \vdash B \Rightarrow \tt\}
\]

(2)

- From lecture 14, the forward reachability collecting semantics of commands:

\[
\text{Rcom} \in \text{Com} \mapsto \rho(\text{Env}[P]) \mapsto \langle \text{in}[P][C] \mapsto \rho(\text{Env}[P])
\]

\[
\text{Rcom}[C]R\ell \text{ def } = \{\rho \mid \exists \rho' \in R : \langle \text{at}[P][C], \rho' \rangle, \langle \ell, \rho \rangle \in \tau^\ast[C]\}
\]

(3)

- From lecture 16, the generic abstraction of value properties:

\[
\langle \rho(\|_{\Omega}), \leq \rangle \xleftarrow{\gamma} \langle L, \leq \rangle
\]

(4)

- the abstraction of environment properties:

\[
\langle \rho(\mathbb{V} \mapsto \|_{\Omega}), \leq \rangle \xleftarrow{\gamma} \langle \mathbb{V} \mapsto L, \leq \rangle
\]

(5)

where

\[
\alpha(R) \text{ def } = \lambda X \in \mathbb{V} : \alpha(\{\rho(X) \mid \rho \in R\}),
\]

\[
\gamma(\tau) \text{ def } = \{\rho \mid \forall X \in \mathbb{V} : \rho(X) \in \gamma(\tau(X))\}
\]

(6)

- the functional abstraction of monotonic predicate transformers:

\[
\langle \rho(\mathbb{V} \mapsto \|_{\Omega}) \mapsto \rho(\|_{\Omega}), \leq \rangle \xleftarrow{\gamma} \langle (\mathbb{V} \mapsto L) \mapsto \rho(\|_{\Omega}), \leq \rangle
\]

(7)

where
\[ \alpha'(\Phi) \overset{\text{def}}{=} \alpha \circ \Phi \circ \gamma, \quad (8) \]

\[ \gamma'(\varphi) \overset{\text{def}}{=} \gamma \circ \varphi \circ \alpha \]

- the nonrelational abstraction of the forward reachability collecting semantics of commands:

\[
\left\langle \varphi(\mathbb{R}) \xrightarrow{\quad \gamma[C] \quad} (\text{in}_p[C] \xrightarrow{\quad} \varphi(\mathbb{R})), \quad \frac{\gamma[C]}{\alpha[C]} \right\rangle
\]

where

\[ \alpha[C] \varphi \overset{\text{def}}{=} \lambda r.\lambda L.\alpha(\varphi(\gamma(r))(\ell)) \]

\[ \gamma[C] \psi \overset{\text{def}}{=} \lambda R.\lambda L.\gamma(\psi(\alpha(R))(\ell)) \]

- From lecture 16, various generic nonrelational abstract semantics, starting from the abstract semantics of arithmetic expressions:

\[ \text{Faexp}[A] \overset{\text{def}}{=} \alpha'(\text{Faexp}[A]) \quad (9) \]

- the generic nonrelational abstract semantics of boolean expressions:

\[ \text{Abexp} \in \text{Bexp} \mapsto (\text{V} \mapsto L) \xrightarrow{\quad \gamma \quad} (\text{V} \mapsto L) \quad (10) \]

\[ \text{Abexp}[B] \overset{\text{def}}{=} \alpha(\text{Cbexp}[B]) \quad (11) \]

and for the empty set:

\[ \text{Abexp}[B]|_{\gamma.\perp} \overset{\text{def}}{=} \text{Abexp}[B] \quad (12) \]

- From lecture 16, the initialization and simple sign abstraction:

\[ \alpha(P) \overset{\text{def}}{=} \begin{cases} \{ \text{BOT} \} & \text{if } P = \{ \Omega_A \} \quad (13) \\ \{ \text{NEG} \} & \text{if } P = \{ \text{max_int}, -1 \} \cup \{ \Omega_A \} \\ \{ \text{ZERO} \} & \text{if } P = \{ 0, \Omega_A \} \\ \{ \text{POS} \} & \text{if } P = \{ 1, \text{max_int} \} \cup \{ \Omega_A \} \quad (17) \\ \{ \text{ERR} \} & \text{if } P \subseteq \{ \Omega_A \} \quad \text{INIT} \\ \{ \text{TOP} \} & \text{if } P \subseteq \{ \Omega_1, \Omega_A \} \quad \text{TOP} \end{cases} \]

and concretization:

\[ \gamma(\text{BOT}) \overset{\text{def}}{=} \{ \Omega_A \} \]

\[ \gamma(\text{NEG}) \overset{\text{def}}{=} \{ \text{NEG} \} \]

\[ \gamma(\text{ZERO}) \overset{\text{def}}{=} \{ \text{ZERO} \} \]

\[ \gamma(\text{POS}) \overset{\text{def}}{=} \{ \text{POS} \} \]

\[ \gamma(\text{ERR}) \overset{\text{def}}{=} \{ \text{ERR} \} \]

\[ \gamma(\text{INIT}) \overset{\text{def}}{=} \text{IN} \]

\[ \gamma(\text{TOP}) \overset{\text{def}}{=} \text{TOP} \]

\[ \gamma(\text{ERR}) \overset{\text{def}}{=} \{ \text{ERR} \} \]

\[ \gamma(\text{INIT}) \overset{\text{def}}{=} \{ \text{INIT} \} \]

\[ \gamma(\text{TOP}) \overset{\text{def}}{=} \{ \text{TOP} \} \]

\[ \gamma(\text{ERR}) \overset{\text{def}}{=} \{ \text{ERR} \} \]

\[ \gamma(\text{INIT}) \overset{\text{def}}{=} \{ \text{INIT} \} \]

\[ \gamma(\text{TOP}) \overset{\text{def}}{=} \{ \text{TOP} \} \]
Motivating example

- The forward/top-down static analysis of arithmetic expressions brings no information on the values of the variables appearing in the arithmetic expressions when the expected result of such expressions is known, e.g. in tests

- Example (initialization and simple sign):

```bash
% cd Initialization-Simple-Sign
% ./a.out ../Examples/example13.sil
{ y:ERR; r:ERR }
0: y := ?;
y := ?;
1: if (y = 0) then 2: r := 0
    r := 0
3: else {((y < 0) | (0 < y))}
4: r := 0
5: fi
6: { y:INI; r:ZERO }
```

- Example 13 (left) where \( r = 0 \) at point 6: shows that in example 14 (right), \( r \) is not known to be 0 at line 3: whence that \( y \) is not known to be 0 at line 2: whence that \((y = 0)\) brings no abstract information on \( y \) at line 1:

- More generally, any information on the possible result of an arithmetic expression should bring information on the values of the variables involved in the arithmetic expressions for its result to satisfy the known information
Backward/bottom-up collecting semantics of arithmetic expressions

- The *backward/top-down collecting semantics* \( Baexp[A](R) \) of an arithmetic expression \( A \) defines the subset of possible environments \( R \) such that the arithmetic expression may evaluate, without producing a runtime error, to a value belonging to given set \( P \)

\[
\begin{align*}
Baexp & \in Aexp \mapsto \varphi(\mathbb{R}) \mapsto \varphi(\mathbb{I}_g) \mapsto \varphi(\mathbb{R}), \\
Baexp[A](R)P & \overset{\text{def}}{=} \{ \rho \in R \mid \exists i \in P \cap I : \rho \vdash A \Rightarrow i \}.
\end{align*}
\]

**Theorem.** \( \forall P \in \varphi(\mathbb{R}) : \lambda R \cdot Baexp[A](R)P \) is a lower closure operator

**Proof.**
1. Monotone. If \( R_1 \subseteq R_2 \) then \( \{ \rho \in R_1 \mid \exists i \in P \cap I : \rho \vdash A \Rightarrow i \} \subseteq \{ \rho \in R_2 \mid \exists i \in P \cap I : \rho \vdash A \Rightarrow i \} \) whence \( Baexp[A](\{R_1\})P \subseteq Baexp[A](\{R_2\})P \)
2. Reductive. \( \{ \rho \in R \mid \exists i \in P \cap I : \rho \vdash A \Rightarrow i \} \subseteq R \) and so \( Baexp[A](\{R\})P \subseteq R \)
3. Idempotent. \( \{ \rho \in \{\rho' \in R \mid \exists i \in P \cap I : \rho' \vdash A \Rightarrow i \} \mid \exists j \in P \cap I : \rho \vdash A \Rightarrow j \} = \{ \rho \in R \mid \exists i \in P \cap I : \rho \vdash A \Rightarrow j \} \) and so

\[ \begin{align*}
\end{align*}
\]

Operational semantics of arithmetic expressions (recall)

Let us recall that:

\[
\begin{align*}
\rho \vdash n & \Rightarrow n & \text{decimal numbers}; \\
\rho \vdash \varphi & \Rightarrow \varphi & \text{variables}; \\
\rho \vdash ? & \Rightarrow ? & \text{random}; \\
\rho \vdash \varphi \vdash A & \Rightarrow v & \text{unary arithmetic operations}; \\
\rho \vdash \varphi \vdash uA & \Rightarrow \varphi u & \\
\rho \vdash \varphi \vdash A_1 \Rightarrow v_1 & \rho \vdash \varphi \vdash A_2 \Rightarrow v_2 & \text{binary arithmetic operations}; \\
\rho \vdash \varphi \vdash A_1 \vdash \varphi \vdash A_2 \Rightarrow \varphi v_1 \varphi v_2 & \\
\rho \vdash \varphi \vdash A_1 \vdash \varphi \vdash A_2 & \Rightarrow \varphi v_1 \varphi v_2 & \text{binary arithmetic operations}.
\end{align*}
\]

Structural definition of the backward/bottom-up collecting semantics of arithmetic expressions

\[
\begin{align*}
Baexp[n](R)P & = \{ n \in P \cap I \mid R : \emptyset \} \quad (25) \\
Baexp[\varphi](R)P & = \{ \rho \in R \mid \rho(\varphi) \in P \cap I \} \quad (26) \\
Baexp[?](R)P & = \{ P \cap I : \emptyset \subseteq R \} \quad (27) \\
Baexp[uA](R)P & = Baexp[A](R)u(\text{Faexp}[A](R),P) \quad (28)
\end{align*}
\]

where \( u(\varphi, Q, P) \overset{\text{def}}{=} \{ v \in Q \mid \varphi v \in P \cap I \} \)

\[
\begin{align*}
Baexp[A_1 \vdash A_2](R)P & = \quad (29)
\end{align*}
\]

let \( \langle P_1, P_2 \rangle = \varphi(\text{Faexp}[A_1](R), \text{Faexp}[A_2](R), P) \) in

\[
\begin{align*}
& \quad \text{Baexp}[A_1](R)P_1 \cap \text{Baexp}[A_2](R)P_2 \\
\end{align*}
\]

where \( \varphi(\varphi, P_1, P_2, P) \overset{\text{def}}{=} \{ (v_1, v_2) \in P_1 \times P_2 \mid v_1 \varphi v_2 \in P \cap I \} \)
Structural definition of the backward/bottom-up collecting semantics of arithmetic expressions

Before providing the proof, let us recall a few definitions from lectures 5 and 8:

\[
\begin{align*}
\Omega_i & \overset{\text{def}}{=} \Omega_i, \\
\Omega_i & \overset{\text{def}}{=} \Omega_i^i, \quad \text{if } i \in \mathbb{N}; \\
\Omega_i & \overset{\text{def}}{=} \Omega_i^{*}, \quad \text{if } i \notin \mathbb{N}.
\end{align*}
\]

(30)

\[
\begin{align*}
\Omega_i \cup \Omega_j & \overset{\text{def}}{=} \Omega_i, \\
i_1 \cup i_2 & \overset{\text{def}}{=} i_1 \cup i_2, \quad \text{if } b \in \{+,-,\ast\} \land i_1 \cup i_2 \in \mathbb{N}; \\
i_1 \cup i_2 & \overset{\text{def}}{=} i_1 \cup i_2, \quad \text{if } b \in \{/,\mod\} \land i_1 \cup i_2 \in \mathbb{N} \land i_1 \cup i_2 \in \mathbb{N}; \\
i_1 \cup i_2 & \overset{\text{def}}{=} \Omega_0, \quad \text{if } i_1 \cup i_2 \in \mathbb{N} \land i_1 \cup i_2 \notin \mathbb{N}.
\end{align*}
\]

(31)

**PROOF.**

- \(\text{Baexp}[n](R)P\)

\[
\begin{align*}
\{\rho \in R \mid \exists j \in \Pi \land \rho \vdash n \Rightarrow j\} & \overset{\text{def}}{=} \{\rho \in R \mid \exists j \in \Pi \land \rho \vdash n \Rightarrow j\} \quad \text{(by (19))} \\
\{\rho \in R \mid \rho \in \Pi \land \rho \vdash n \Rightarrow j\} & \overset{\text{def}}{=} \{\rho \in R \mid \rho \in \Pi \land \rho \vdash n \Rightarrow j\} \quad \text{(by (20))} \\
\{\rho \in R \mid \Pi \land \rho \vdash n \Rightarrow j\} & \overset{\text{def}}{=} \{\rho \in R \mid \Pi \land \rho \vdash n \Rightarrow j\} \quad \text{(by (21))} \\
\{\rho \in R \mid \Pi \land \rho \vdash n \Rightarrow j\} & \overset{\text{def}}{=} \{\rho \in R \mid \Pi \land \rho \vdash n \Rightarrow j\} \quad \text{(by (22))} \\
\{\rho \in R \mid \Pi = \emptyset \land \rho \vdash n \Rightarrow j\} & \overset{\text{def}}{=} \{\rho \in R \mid \Pi = \emptyset \land \rho \vdash n \Rightarrow j\} \quad \text{(by (23))}
\end{align*}
\]

- \(\text{Baexp}[x](R)P\)

\[
\begin{align*}
\{\rho \in R \mid \exists j \in \Pi \land \rho \vdash x \Rightarrow j\} & \overset{\text{def}}{=} \{\rho \in R \mid \exists j \in \Pi \land \rho \vdash x \Rightarrow j\} \quad \text{(by (19))} \\
\{\rho \in R \mid \rho \in \Pi \land \rho \vdash x \Rightarrow j\} & \overset{\text{def}}{=} \{\rho \in R \mid \rho \in \Pi \land \rho \vdash x \Rightarrow j\} \quad \text{(by (20))} \\
\{\rho \in R \mid \Pi \land \rho \vdash x \Rightarrow j\} & \overset{\text{def}}{=} \{\rho \in R \mid \Pi \land \rho \vdash x \Rightarrow j\} \quad \text{(by (21))} \\
\{\rho \in R \mid \Pi \land \rho \vdash x \Rightarrow j\} & \overset{\text{def}}{=} \{\rho \in R \mid \Pi \land \rho \vdash x \Rightarrow j\} \quad \text{(by (22))} \\
\{\rho \in R \mid \Pi = \emptyset \land \rho \vdash x \Rightarrow j\} & \overset{\text{def}}{=} \{\rho \in R \mid \Pi = \emptyset \land \rho \vdash x \Rightarrow j\} \quad \text{(by (23))}
\end{align*}
\]

- \(\text{Baexp}[?](R)P\)

\[
\begin{align*}
\{\rho \in R \mid \exists j \in \Pi \land \rho \vdash ? \Rightarrow j\} & \overset{\text{def}}{=} \{\rho \in R \mid \exists j \in \Pi \land \rho \vdash ? \Rightarrow j\} \quad \text{(by (19))} \\
\{\rho \in R \mid \rho \in \Pi \land \rho \vdash ? \Rightarrow j\} & \overset{\text{def}}{=} \{\rho \in R \mid \rho \in \Pi \land \rho \vdash ? \Rightarrow j\} \quad \text{(by (20))} \\
\{\rho \in R \mid \Pi \land \rho \vdash ? \Rightarrow j\} & \overset{\text{def}}{=} \{\rho \in R \mid \Pi \land \rho \vdash ? \Rightarrow j\} \quad \text{(by (21))} \\
\{\rho \in R \mid \Pi = \emptyset \land \rho \vdash ? \Rightarrow j\} & \overset{\text{def}}{=} \{\rho \in R \mid \Pi = \emptyset \land \rho \vdash ? \Rightarrow j\} \quad \text{(by (22))} \\
\{\rho \in R \mid \Pi = \emptyset \land \rho \vdash ? \Rightarrow j\} & \overset{\text{def}}{=} \{\rho \in R \mid \Pi = \emptyset \land \rho \vdash ? \Rightarrow j\} \quad \text{(by (23))}
\end{align*}
\]
Generic backward/bottom-up non-relational abstract semantics of arithmetic expressions

We now design the backward/bottom-up abstract semantics of arithmetic expressions

\[ \text{Baexp}^q \in \text{Aexp} \mapsto (\forall \mapsto L) \xrightarrow{m} L \xrightarrow{m} (\forall \mapsto L). \]

The objective is to get an overapproximation of the backward collecting semantics (19) such that

\[ \text{Baexp}^q[A] \xrightarrow{\alpha} \alpha^q(\text{Baexp}[A]). \] (33)

Structural definition of the generic backward/bottom-up non-relational abstract semantics of arithmetic expressions

\[
\begin{align*}
\text{Baexp}^q[A][\lambda y \cdot \bot] & \equiv \lambda y \cdot \bot & \text{if } \gamma(\bot) = 0 \\
\text{Baexp}[[n][r]] & \equiv \langle n \rangle (p) \equiv r : \lambda y \cdot \bot \\
\text{Baexp}[[x][r]] & \equiv r[x := r(x) \cap p \cap ?] \\
\text{Baexp}[[?]][r] & \equiv \langle ?(p) \equiv r : \lambda y \cdot \bot \rangle \\
\text{Baexp}[[u A]][r] & \equiv \text{Baexp}^q[A][\langle u \rangle (\text{Faexp}^q[A][r], p)) \\
\text{Baexp}[[A_1 \cdot A_2]][r] & \equiv \langle \langle p_1, p_2 \rangle = b'(\text{Faexp}^q[A_1][r], \text{Faexp}^q[A_2][r], p) \rangle \text{ in } \text{Baexp}^q[A_1][\langle r \rangle][p_1] \cap \text{Baexp}^q[A_2][\langle r \rangle][p_2]
\end{align*}
\]
Parameterized by the following backward abstract operations on $L$:

$$
n'(p) \triangleq (n \in \gamma(p) \land I) \tag{36}
$$

$$
?^*(p) \triangleq (\gamma(p) \land \mathbb{I} \neq \emptyset) \tag{37}
$$

$$
u(q, p) \equiv \alpha_{\{i \in \gamma(q) \mid \mathbb{u} \in \gamma(p) \land I\}}\tag{38}
$$

$$
b'^*(q_1, q_2, p) \equiv \alpha_{2^\{i_1, i_2 \in \gamma^*(q_1, q_2) \mid i_1, i_2 \in \gamma(p) \land I\}}\tag{39}
$$

Given any $r \in \mathbb{V} \Rightarrow L$, $r \neq \lambda y. \bot$ or $\gamma(\bot) \neq \emptyset$ and $p \in L$, we proceed by structural induction on the arithmetic expression $A$.

1. When $A = n \in \text{Nat}$ is a number, we have

$$
a^*(\text{Baexp}[n](r)p) = \alpha(\text{Baexp}[n](\gamma(r))\gamma(p))$$

2. When $A = x \in \mathbb{V}$ is a variable, we have

$$
a^*(\text{Baexp}[x](r)p) = \alpha(\text{Baexp}[x](\gamma(r))\gamma(p))$$

3. When $A = n \in \gamma(p) \land I \not\supseteq r : \lambda y. \bot$ we define $n^*(p) \triangleq (n \in \gamma(p) \land I)$.

$$
\alpha^*(\text{Baexp}[n](r)p) \trianglerighteq \alpha(\text{Baexp}[n](\gamma(r))\gamma(p))$$

$$
n^*(p) \not\trianglerighteq \lambda y. \bot.$$

Calculation design of the generic backward/bottom-up non-relational abstract semantics of arithmetic expressions

PROOF. We derive $\text{Baexp}^*[A]$ by calculus, as follows

$$
\alpha^*(\text{Baexp}[A])
$$

$$= \ \{\text{def. (32) of } \alpha^*\}$$

$$\lambda r \in \mathbb{V} \Rightarrow L \cdot \lambda p \in L \cdot \alpha(\text{Baexp}[A](\gamma(r))\gamma(p))$$

$$= \ \{\text{def. (19) of } \text{Baexp}[A]\}$$

$$\lambda r \in \mathbb{V} \Rightarrow L \cdot \lambda p \in L \cdot \alpha(\{p \in \gamma(r) \mid \exists p \in \gamma(p) \land I : p \vdash A \Rightarrow t\})$$.

If $r$ is the infimum $\lambda y. \bot$ where the infimum $\bot$ of $L$ is such that $\gamma(\bot) = \emptyset$, then $\gamma(r) = \emptyset$ whence

$$\alpha^*(\text{Baexp}[A])(\lambda y. \bot)p$$

2. When $A = x \in \mathbb{V}$ is a variable, we have

$$\alpha^*(\text{Baexp}[x](r)p) = \alpha(\text{Baexp}[x](\gamma(r))\gamma(p))$$

3. When $A = n \in \gamma(p) \land I \not\supseteq r : \lambda y. \bot$ we define $n^*(p) \triangleq (n \in \gamma(p) \land I)$.

$$\alpha^*(\text{Baexp}[n](r)p) \trianglerighteq \alpha(\text{Baexp}[n](\gamma(r))\gamma(p))$$

$$n^*(p) \not\trianglerighteq \lambda y. \bot.$$
\( \{ \text{def. (6) of } \gamma \} \)
\[ \alpha(\{ \rho \mid \forall Y \neq X : \rho(Y) \in \gamma(\tau(Y)) \land \rho(X) \in \gamma(\tau(X)) \land \gamma(p) \land \gamma \circ \alpha(I) \}) \]
\( \{ \text{def. (6) of } \gamma \} \)
\[ \alpha(\{ \rho \mid \forall Y \neq X : \rho(Y) \in \gamma(\tau(Y)) \land \rho(X) \in \gamma(\tau(X)) \land \gamma(p) \land \gamma \circ \alpha(I) \}) \]
\( \{ \text{def. environment assignment} \} \)
\[ \alpha(\{ \rho \mid \forall Y \neq X : \rho(Y) \in \gamma(\tau(Y)) \land \rho(X) \in \gamma(\tau(X)) \land \gamma(p) \land \gamma \circ \alpha(I) \}) \]
\( \{ \text{set notation} \} \)
\[ \alpha(\gamma(\tau(X) := r(X) \land p \land \alpha(I))) \]
\( \{ \text{by definition } Baexp'[?] \} \)
\[ \alpha(Baexp'[?](r)p) \]
\[ r[X := r(X) \land p \land ?] \]
\( \{ \text{by defining } Baexp'[?] \} \)
\[ \alpha(Baexp'[?](r)p) \]
\[ \{ \text{by def. (32) of } \alpha' \} \]
\[ \alpha(Baexp'[?](r)p) \]
\( \{ \text{by def. (32) of } \alpha' \} \)
\[ \alpha(Baexp'[?](r)p) \]
\[ \{ \text{by def. (28) of } Baexp[u A'] \} \]
\[ \alpha(Baexp'[?](r)p) \]
\[ \{ \text{by def. (32) of } \alpha' \} \]
\[ \alpha(Baexp'[?](r)p) \]
\( \{ \text{by def. (32) of } \alpha' \} \)
\[ \alpha(Baexp'[?](r)p) \]
\[ \{ \text{by def. (32) of } \alpha' \} \]
\[ \alpha(Baexp'[?](r)p) \]
\[ \{ \text{by def. (32) of } \alpha' \} \]
\[ \alpha(Baexp'[?](r)p) \]
\( \{ \text{by def. (32) of } \alpha' \} \)
\[ \alpha(Baexp'[?](r)p) \]
\[ \{ \text{by def. (32) of } \alpha' \} \]
\[ \alpha(Baexp'[?](r)p) \]
When $A = A_1 \cdot A_2$ is a binary operation, we have

$$\alpha'(\text{Baexp}[A_1 \cdot A_2](\tau)p)$$

$$= \alpha(\text{Baexp}[A_1 \cdot A_2]((\gamma(\tau))(\gamma(p))))$$

$$= \langle \text{def. (32) of } \alpha' \rangle$$

Let $(p_1, p_2) = b'(\text{Baexp}[A_1](\tau), \text{Baexp}[A_2](\tau), \gamma(p))$ in $\text{Baexp}[A_1](\tau)p_1 \cap \text{Baexp}[A_2](\tau)p_2$

$$= \langle \text{defining } \text{Baexp}[A_1 \cdot A_2](\tau)p \rangle$$

Let $(p_1, p_2) = b'(\text{Baexp}[A_1], \text{Baexp}[A_2], p)$ in $\text{Baexp}[A_1](\tau)p_1 \cap \text{Baexp}[A_2](\tau)p_2$

$$\text{Baexp}[A_1 \cdot A_2](\tau)p$$

---

Implementation of the structural definition of the generic backward/bottom-up non-relational abstract semantics of arithmetic expressions:

1. (* baexp.mli *)
2. open Abstract_Syntax
3. open Avaluas
4. open Aenv
5. (* backward evaluation of arithmetic operations *)
6. val b_aexp : aexp -> Aenv.t -> Avaluas.t -> Aenv.t
Primitive backward/bottom-up non-relational abstract operations for initialization and simple sign analysis

We must now define the primitive operations $n^d$, $?^d$, $u^d$, $b^d$ for the abstract lattice

$$
\gamma(\text{BOT}) \triangleq \{ \Omega_3 \}, \quad \gamma(\text{INI}) \triangleq \emptyset \cup \{ \Omega_3 \}, \\
\gamma(\text{NEG}) \triangleq \lfloor \min \text{ _int}, -1 \rfloor \cup \{ \Omega_3 \}, \quad \gamma(\text{ERR}) \triangleq \{ \Omega_3, \Omega_4 \}, \\
\gamma(\text{ZERO}) \triangleq \{ 0, \Omega_4 \}, \quad \gamma(\text{TOP}) \triangleq \emptyset
$$

Implementation of the primitive backward/bottom-up non-relational abstract arithmetic operations for initialization and simple sign analysis

1. In the abstract interpretation (35) of variables, we have

$$
?^d = \text{INI}
$$

by definition (17) of $\alpha$. 

7 (* baexp.ml *)
8 open Abstract_Syntax
9 (* Backward abstract interpretation of arithmetic operations *)
10 let rec b_aexp' a r p =
11    match a with
12    | (NAT i) -> if (Avalues.b_NAT i p) then r else (Aenv.bot ())
13    | (VAR v) ->
14       (Aenv.set r v (Avalues.meet (Avalues.meet (Aenv.get r v) p) (Avalues.f_RANDOM ()
15    | RANDOM -> if (Avalues.b_RANDOM p) then r else (Aenv.bot ())
16    | (UMINUS a1) -> (b_aexp' a1 r (Avalues.b_UMINUS (Aaexp.a_aexp a1 r) p))
17    | (UPLUS a1) -> (b_aexp' a1 r (Avalues.b_UPLUS (Aaexp.a_aexp a1 r) p))
18    | (PLUS (a1, a2)) ->
19       let (p1, p2) = (Avalues.b_PLUS (Aaexp.a_aexp a1 r) (Aaexp.a_aexp a2 r) p)
20       in (Aenv.meet (b_aexp' a1 r p1) (b_aexp' a2 r p2))
21    | (MINUS (a1, a2)) ->
22       let (p1, p2) = (Avalues.b_MINUS (Aaexp.a_aexp a1 r) (Aaexp.a_aexp a2 r) p)
23       in (Aenv.meet (b_aexp' a1 r p1) (b_aexp' a2 r p2))
24    | (TIMES (a1, a2)) ->
25       let (p1, p2) = (Avalues.b_TIMES (Aaexp.a_aexp a1 r) (Aaexp.a_aexp a2 r) p)
26       in (Aenv.meet (b_aexp' a1 r p1) (b_aexp' a2 r p2))
27    | (DIV (a1, a2)) ->
28       let (p1, p2) = (Avalues.b_DIV (Aaexp.a_aexp a1 r) (Aaexp.a_aexp a2 r) p)
29       in (Aenv.meet (b_aexp' a1 r p1) (b_aexp' a2 r p2))
30    | (MOD (a1, a2)) ->
31       let (p1, p2) = (Avalues.b_MOD (Aaexp.a_aexp a1 r) (Aaexp.a_aexp a2 r) p)
32       in (Aenv.meet (b_aexp' a1 r p1) (b_aexp' a2 r p2))
33    | (DIV (a1, a2)) ->
34      if (Aenv.is_bot r) & (Avalues.isbotempty ()) then (Aenv.bot ()) else b_aexp' a r p
2 From the definition (36) of $n^\circ$ and (18) of $\gamma$, we directly get by case analysis

<table>
<thead>
<tr>
<th>$n(p)$</th>
<th>BOT</th>
<th>NEG</th>
<th>ZERO</th>
<th>POS</th>
<th>INIT</th>
<th>ERR</th>
<th>TOP</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n \in \text{[min_int}, -1]$</td>
<td>tt</td>
<td>tt</td>
<td>tt</td>
<td>tt</td>
<td>tt</td>
<td>tt</td>
<td>tt</td>
</tr>
<tr>
<td>$n = 0$</td>
<td>tt</td>
<td>tt</td>
<td>tt</td>
<td>tt</td>
<td>tt</td>
<td>tt</td>
<td>tt</td>
</tr>
<tr>
<td>$n \in \text{[1, max_int}]$</td>
<td>tt</td>
<td>tt</td>
<td>tt</td>
<td>tt</td>
<td>tt</td>
<td>tt</td>
<td>tt</td>
</tr>
<tr>
<td>$n &lt; \text{min_int} \lor n &gt; \text{max_int}$</td>
<td>tt</td>
<td>tt</td>
<td>tt</td>
<td>tt</td>
<td>tt</td>
<td>tt</td>
<td>tt</td>
</tr>
</tbody>
</table>

3 From the definition (37) of $?^\circ$ and (18) of $\gamma$, we directly get by case analysis

<table>
<thead>
<tr>
<th>$p$</th>
<th>BOT</th>
<th>NEG</th>
<th>ZERO</th>
<th>POS</th>
<th>INIT</th>
<th>ERR</th>
<th>TOP</th>
</tr>
</thead>
<tbody>
<tr>
<td>$?^\circ(p)$</td>
<td>tt</td>
<td>tt</td>
<td>tt</td>
<td>tt</td>
<td>tt</td>
<td>tt</td>
<td>tt</td>
</tr>
</tbody>
</table>

4 For the backward unary arithmetic operations (38), we have

<table>
<thead>
<tr>
<th>$p$</th>
<th>BOT</th>
<th>NEG</th>
<th>ZERO</th>
<th>POS</th>
<th>INIT</th>
<th>ERR</th>
<th>TOP</th>
</tr>
</thead>
<tbody>
<tr>
<td>$+^\circ(q, p)$</td>
<td>BOT</td>
<td>$q \uplus \text{NEG}$</td>
<td>$q \uplus \text{ZERO}$</td>
<td>$q \uplus \text{POS}$</td>
<td>$q \uplus \text{INIT}$</td>
<td>$BOT$</td>
<td>$q \uplus \text{INIT}$</td>
</tr>
<tr>
<td>$-(q, p)$</td>
<td>BOT</td>
<td>$q \uplus \text{POS}$</td>
<td>$q \uplus \text{NEG}$</td>
<td>$q \uplus \text{INIT}$</td>
<td>$BOT$</td>
<td>$q \uplus \text{INIT}$</td>
<td></td>
</tr>
</tbody>
</table>

5 If $p = \text{BOT}$ or $p = \text{ERR}$ then by (18),

$$\forall i \in \gamma(p) \cap I = \{\Omega_1, \Omega_3\} \cap [\text{min_int}, \text{max_int}] = \emptyset$$

is false so that $u^\circ(q, p) = \alpha(0) = \text{BOT}$.

6 If $p = \text{POS}$ then by (18), $\neg i \in \gamma(p) \cap I = [1, \text{max_int}]$ if and only if $\neg i \in [\text{min_int} + 1, \text{max_int}] - I$ so that $\neg (q, p) = \alpha(\gamma(q) \cap [\text{min_int} + 1, \text{max_int}])$ by (18). But $\gamma$ preserves meets whence this is equal to $\alpha(\gamma(q \cap \text{NEG})) \subseteq q \cap \text{NEG}$ since $\alpha \circ \gamma$ is reductive.

7 If $p = \text{INI}$ or $p = \text{TOP}$ then by (18), $\neg i \in \gamma(p) \cap I = [\text{min_int}, \text{max_int}]$ if and only if $\neg i \in [\text{min_int} + 1, \text{max_int}]$ so that $\neg (q, p) = \alpha(\gamma(q) \cap [\text{min_int} + 1, \text{max_int}]) \subseteq \alpha(\gamma(q \cap \text{INI}))$ by (18). But $\gamma$ preserves meets whence this is equal to $\alpha(\gamma(q \cap \text{INI})) \subseteq q \cap \text{INI}$ since $\alpha \circ \gamma$ is reductive.

8 For the backward binary arithmetic operations (39), we have

$$/\circ(q_1, q_2, p) \overset{\text{def}}{=} \text{mod}^{\circ}(q_1, q_2, p) \overset{\text{def}}{=} (q_1 \in \{\text{BOT, NEG, ERR}\} \lor q_2 \in \{\text{BOT, NEG, ZERO, ERR}\} \lor p \in \{\text{BOT, NEG, ERR}\} \lor \{\text{BOT, BOT}\} \lor (p = \text{POS}) ? \text{smash}(\{q_1 \cap \text{POS}, q_2 \cap \text{POS}\}) \circ (q_1 \cap \text{INI}, q_2 \cap \text{POS}))$$

$$\text{smash}(x, y) \overset{\text{def}}{=} (x = \text{BOT} \lor y = \text{BOT} ? \{\text{BOT, BOT}\} ? (x, y)).$$

Proof. Let us consider a few typical cases.

8.1 If $p = \text{BOT}$ or $p = \text{ERR}$ then by (18),

$$\forall i \in \gamma(p) \cap I = \{\Omega_1, \Omega_3\} \cap [\text{min_int}, \text{max_int}] = \emptyset$$

is false so that $u^\circ(q, p) = \alpha(0) = \text{BOT}$.

6. If $p = \text{POS}$ then by (18), $\neg i \in \gamma(p) \cap I = [1, \text{max_int}]$ if and only if $\neg i \in [\text{min_int} + 1, \text{max_int}] - I$ so that $\neg (q, p) = \alpha(\gamma(q) \cap [\text{min_int} + 1, \text{max_int}])$ by (18). But $\gamma$ preserves meets whence this is equal to $\alpha(\gamma(q \cap \text{NEG})) \subseteq q \cap \text{NEG}$ since $\alpha \circ \gamma$ is reductive.

7. If $p = \text{INI}$ or $p = \text{TOP}$ then by (18), $\neg i \in \gamma(p) \cap I = [\text{min_int}, \text{max_int}]$ if and only if $\neg i \in [\text{min_int} + 1, \text{max_int}]$ so that $\neg (q, p) = \alpha(\gamma(q) \cap [\text{min_int} + 1, \text{max_int}]) \subseteq \alpha(\gamma(q \cap \text{INI}))$ by (18). But $\gamma$ preserves meets whence this is equal to $\alpha(\gamma(q \cap \text{INI})) \subseteq q \cap \text{INI}$ since $\alpha \circ \gamma$ is reductive.

8. For the backward binary arithmetic operations (39), we have
If \( p \in \{ \text{BOT, NEG, ERR} \} \) then \( i_1 \cdot b \cdot i_2 \not\in \gamma(p) \cap I \subseteq [\min, \max, -1] \) in contradiction with (31) showing that \( i_1 \cdot b \cdot i_2 \) is not negative. Again \( b(q_1, q_2, p) = \alpha^2(\theta) = (\text{BOT, BOT}) \) according to the componentwise definition of \( \alpha^2 \) and (17).

Otherwise to have \( i_1 \cdot b \cdot i_2 \in I \), we must have \( i_1 \in [0, \max, \text{int}] \) and \( i_2 \in [1, \max, \text{int}] \) whence necessarily \( i_1 \in \gamma(\text{INI}) \) and \( i_2 \in \gamma(\text{POS}) \) so that

\[
\alpha^2(\gamma(q_1 \cap \text{INI}, q_2 \cap \text{POS})) \subseteq (q_1 \cap \text{INI}, q_2 \cap \text{POS}) \equiv b(q_1, q_2, p).
\]

Moreover the quotient is strictly positive only if the dividend is not zero. □

**9** With the same reasoning, for addition \( +^\gamma \), we have

\[
+^\gamma(q_1, q_2, p) = \begin{cases} 
(\text{BOT, BOT}) & \text{if } q_1 \in \{ \text{BOT, ERR} \} \lor q_2 \in \{ \text{BOT, ERR} \} \lor p \in \{ \text{BOT, ERR} \} \\
(q_1 \cap \text{INI}, q_2 \cap \text{INI}) & \text{if } p \in \{ \text{INI, TOP} \} 
\end{cases}
\]

**10** The backward ternary subtraction operation \( -^\gamma \) is defined as

\[
-^\gamma(q_1, q_2, p) \overset{\text{def}}{=} \text{let } (r_1, r_2) = -^\gamma(q_1, -^\gamma(q_2), p) \text{ in } (r_1, -^\gamma(r_2)).
\]

**11** The handling of the backward ternary multiplication operation \((^\gamma)_\ast\) is similar
Implementation of the primitive backward/bottom-up non-relational abstract arithmetic operations for initialization and simple sign analysis

```
(* avalues.mli *)
open Values
(* abstraction of sets of machine integers by initialization *)
(* and simple sign *)
type t = BOT | NEG | ZERO | POS | INI | ERR | TOP

(* backward abstract interpretation of arithmetic expressions *)
exception Error_f_NAT of string
let remove_zeros i =
  let l = (String.length i) in
  if l = 0 then raise (Error_f_NAT "empty integer")
```

70  else if l = 1 then i
71  else if (String.get i 0) = '0' then String.sub i 1 (l - 1)
72  else i
73 let b_NAT i p =
74  let i' = (remove_zeros i) in
75  if i' = "0" then
76    match p with
77    | BOT -> false
78    | NEG -> false
79    | ZERO -> true
80    | POS -> false
81    | INI -> true
82    | ERR -> false
83    | TOP -> true
84  else
85    match p with
86    | BOT -> false
87    | NEG -> false
88    | ZERO -> false
89    | POS -> true
90  else
91  let b_RANDOM p =
92  match p with
93  | BOT -> false
94  | NEG -> false
95  | ZERO -> false
96  | POS -> true
97  | INI -> true
98  | ERR -> false
99  let b_UNMINUS q p =
100  match p with
101  | BOT -> q
102  | NEG -> meet q POS
103  | ZERO -> meet q ZERO
104  | POS -> meet q NEG
105  | INI -> meet q INI
106  | ERR -> q
107  let b_UPPLUS q p =
108  match p with
109  | BOT -> q
Revisiting the non-relational abstract interpretation of boolean expressions

The abstract interpretation of boolean expressions can be revised as follows, using the backward abstract interpretation of arithmetic expressions:

\[
\begin{align*}
\text{Abexp}[B]|_{\bot} & \overset{\text{def}}{=} \lambda y . \bot \quad \text{if } \gamma(\bot) = 0 \\
\text{Abexp}[\text{true}]r & \overset{\text{def}}{=} r \\
\text{Abexp}[\text{false}]r & \overset{\text{def}}{=} \lambda y . \bot \\
\text{Abexp}[A_1 \land A_2]r & \overset{\text{def}}{=} \\
\text{let } \langle p_1, p_2 \rangle = \tilde{c}(\text{Faexp}^p[A_1]r, \text{Faexp}^p[A_2]r) \text{ in } \\
\text{Baexp}[A_1](r)p_1 \land \text{Baexp}[A_2](r)p_2
\end{align*}
\]

Improving the non-relational analysis of boolean expressions using the backward analysis of its arithmetic subexpressions

\[
\begin{align*}
\text{Abexp}[B_1 \land B_2]r & \overset{\text{def}}{=} \text{Abexp}[B_1]r \land \text{Abexp}[B_2]r \\
\text{Abexp}[B_1 | B_2]r & \overset{\text{def}}{=} \text{Abexp}[B_1]r \lor \text{Abexp}[B_2]r
\end{align*}
\]

parameterized by the following abstract comparison operations \(\tilde{c}, c \in \{<, =\}\) on \(L\):

\[
\tilde{c}(p_1, p_2) \equiv^2 \alpha^2\{\{i_1, i_2\} | i_1 \in \gamma(p_1) \cap I \\
\wedge i_2 \in \gamma(p_2) \cap I \wedge i_1 \neq i_2 = \mathbf{tt}\}
\]
Calculational design of the revisited non-relational abstract interpretation of boolean expressions

PROOF. All cases have already been handled, except when \( B = A_1 \land A_2 \) is an arithmetic comparison. Let us recall that

\[
\rho \vdash A_1 \Rightarrow \nu_1, \rho \vdash A_2 \Rightarrow \nu_2, \rho \vdash A_1 \land A_2 \Rightarrow \nu_1 \land \nu_2 \quad (41)
\]

and

\[
\Omega_i \subseteq \nu_i \overset{\text{def.}}{=} \Omega_n, \quad i \subseteq \Omega_i \overset{\text{def.}}{=} \Omega_n, \quad i_1 \cup i_2 \overset{\text{def.}}{=} i_1 \cup i_2.
\]

We have

\[
\text{let } (p_1, p_2) = (\text{Faexp}[A_1]\text{r}, \text{Faexp}[A_2]\text{r}) \text{ in } \\
\alpha'(\{\rho \in \gamma(\tau) \mid \exists i_1, i_2 \in \Omega_1, \rho \vdash A_i \Rightarrow \nu_1 \land \rho \vdash A_2 \Rightarrow \nu_2 \land \nu_1 \land \nu_2 = \texttt{t} \})
\]

\[
= \{\text{def. (42) of } \subseteq \text{ implying } \nu_1, \nu_2 \not\in \Omega = \{\Omega_1, \Omega_2\} \}
\]

\[
\text{let } (p_1, p_2) = (\text{Faexp}[A_1]\text{r}, \text{Faexp}[A_2]\text{r}) \text{ in } \\
\alpha'(\{\rho \in \gamma(\tau) \mid \exists i_1, i_2 \in \Omega_1, \rho \vdash A_i \Rightarrow \nu_1 \land \rho \vdash A_2 \Rightarrow \nu_2 \land \nu_1 \land \nu_2 = \texttt{t} \})
\]

\[
= \{\text{set theory} \}
\]

\[
\text{let } (p_1, p_2) = (\text{Faexp}[A_1]\text{r}, \text{Faexp}[A_2]\text{r}) \text{ in } \\
\alpha'(\{\rho \in \gamma(\tau) \mid \exists i_1, i_2 \in \Omega_1, \rho \vdash A_i \Rightarrow \nu_1 \land \rho \vdash A_2 \Rightarrow \nu_2 \land \nu_1 \land \nu_2 = \texttt{t} \})
\]

\[
= \{\text{def. (41) of } \rho \vdash A_1 \land A_2 \Rightarrow \rho \}
\]

\[
= \{\text{let } \alpha \text{ is extensive }\}
\]

\[
= \{\text{set theory and } \gamma \circ \alpha \text{ is extensive}\}
\]

\[
= \{\text{set theory and } (9)\}
\]

\[
= \{\text{set theory and } (9)\}
\]

\[
= \{\text{let notation}\}
\]

\[
\text{let } (p_1, p_2) = (\text{Faexp}[A_1]\text{r}, \text{Faexp}[A_2]\text{r}) \text{ in } \\
\alpha'(\{\rho \in \gamma(\tau) \mid \exists i_1, i_2 \in \Omega_1, \rho \vdash A_i \Rightarrow \nu_1 \land \rho \vdash A_2 \Rightarrow \nu_2 \land \nu_1 \land \nu_2 = \texttt{t} \})
\]

\[
= \{\text{def. (42) of } \subseteq \text{ implying } \nu_1, \nu_2 \not\in \Omega = \{\Omega_1, \Omega_2\} \}
\]

\[
\text{let } (p_1, p_2) = (\text{Faexp}[A_1]\text{r}, \text{Faexp}[A_2]\text{r}) \text{ in } \\
\alpha'(\{\rho \in \gamma(\tau) \mid \exists i_1, i_2 \in \Omega_1, \rho \vdash A_i \Rightarrow \nu_1 \land \rho \vdash A_2 \Rightarrow \nu_2 \land \nu_1 \land \nu_2 = \texttt{t} \})
\]

\[
= \{\text{set theory} \}
\]

\[
\text{let } (p_1, p_2) = (\text{Faexp}[A_1]\text{r}, \text{Faexp}[A_2]\text{r}) \text{ in } \\
\alpha'(\{\rho \in \gamma(\tau) \mid \exists i_1, i_2 \in \Omega_1, \rho \vdash A_i \Rightarrow \nu_1 \land \rho \vdash A_2 \Rightarrow \nu_2 \land \nu_1 \land \nu_2 = \texttt{t} \})
\]

\[
= \{\text{def. (19) of } \text{Baexp}\}
\]

\[
\text{let } (p_1, p_2) = (\text{Faexp}[A_1]\text{r}, \text{Faexp}[A_2]\text{r}) \text{ in } \\
\alpha'(\{\rho \in \gamma(\tau) \mid \exists i_1, i_2 \in \Omega_1, \rho \vdash A_i \Rightarrow \nu_1 \land \rho \vdash A_2 \Rightarrow \nu_2 \land \nu_1 \land \nu_2 = \texttt{t} \})
\]

\[
= \{\text{def. (32) of } \alpha'\}
\]
Implementation of the revisited non-relational abstract interpretation of boolean expressions

Abstract arithmetic comparison operations for the initialization and simple sign analysis

- Generic abstract boolean equality.
  The calculational design of the abstract equality operation \( \equiv \) does not depend upon the specific choice of \( L \).
  \[
p_1 \equiv p_2 \overset{\text{def}}{=} \text{let } p = p_1 \sqcap p_2 \sqcap ?^c \text{ in } \langle p, p \rangle.
\]

**Proof.**

\[
\alpha^2(\{(i_1, i_2) \mid i_1 \in \gamma(p_1) \sqcap I \land i_2 \in \gamma(p_2) \sqcap I \land i_1 \equiv i_2 = \text{tt} \})
\]

\( \overset{\text{def. (42) of } \equiv}{=} \)

\[
\alpha^2(\{(i, i) \mid i \in \gamma(p_1) \sqcap \gamma(p_2) \sqcap I \})
\]
If \( p_i \in \{ \text{BOT}, \text{ERR} \} \) where \( i = 1 \) or \( i = 2 \) then \( \gamma(p_i) \subseteq \mathbb{E} = \{ \Omega_1, \Omega_2 \} \) so that \( \gamma(p_i) \cap \mathbb{I} = \emptyset \) and we get:

\[
\alpha^2(\{(i_1, i_2) \mid i_1 \in \gamma(p_1) \cap \mathbb{I} \land i_2 \in \gamma(p_2) \cap \mathbb{I} \land i_2 = \text{tt} \}) = \alpha^0(0)
\]

\( \alpha^2(\{i_1, 0 \mid i_1 \in [1, \text{max int}] \land i_1 \leq 0) \}
\]

For \( \{ \text{POS}, \text{ZERO} \} \), we have

\[
\alpha^2(\{i_1, i_2 \mid i_1 \in \gamma(\text{POS}) \cap \mathbb{I} \land i_2 \in \gamma(\text{ZERO}) \cap \mathbb{I} \land i_2 = \text{tt} \})
\]

\( \alpha^2(\{i_1, 0 \mid i_1 \in [1, \text{max int}] \land i_1 \leq 0) \}
\]

- For \( \{ \text{TOP}, \text{TOP} \} \), we have

\[
\alpha^2(\{i_1, i_2 \mid i_1 \in \gamma(\text{TOP}) \cap \mathbb{I} \land i_2 \in \gamma(\text{TOP}) \cap \mathbb{I} \land i_2 = \text{tt} \})
\]

\( \alpha^2(\{i_1, 0 \mid i_1 \in [1, \text{max int}] \land i_1 \leq 0) \}
\]

\( \{\text{set theory}\}
\]

\[
\alpha^2(0)
\]

\( \{\text{componentwise definition of } \alpha^2 \text{ and } (17) \text{ of } \alpha\}
\]

\( \langle \text{BOT}, \text{BOT} \rangle
\]

\( \{\text{def. (43) of } \leq\}
\]

\( \leq (\text{POS}, \text{ZERO}).
\]
Implementation of the abstract arithmetic comparison operations for the initialization and simple sign analysis

191 (* avalues.ml *)
192 (* abstraction of sets of machine integers by initialization *)
193 (* and simple sign *)
194 type t
195 ...
196 (* abstract interpretation of boolean expressions *)
197 val a_EQ : t \rightarrow t \rightarrow t \times t
198 val a_LT : t \rightarrow t \rightarrow t \times t

199 (* avalues.mli *)
200 (* abstraction of sets of machine integers by initialization *)
201 (* and simple sign *)
202 type t
203 ...
204 val a_EQ : t \rightarrow t \rightarrow t \times t
205 val a_LT : t \rightarrow t \rightarrow t \times t
Back to the motivating example

Local decreasing iterations

Motivating example

```bash
% cd Initialization-Simple-Sign-FB
% ./a.out ../Examples/example13.sil
0: { y:ERR; r:ERR }
1: { y:ERR; r:ERR }
if (y = 0) then
  2: r := 0
else {((y < 0) | (0 < y))}
  4: r := 0
fi
6: { y:INI; r:ZERO }

% cd Initialization-Simple-Sign-FB
% ./a.out ../Examples/example14.sil
0: { y:ERR; r:ERR }
1: { y:ERR; r:ERR }
if (y = 0) then
  2: r := y
else {((y < 0) | (0 < y))}
  4: r := 0
fi
6: { y:INI; r:ZERO }

% cd Initialization-Simple-Sign-FB
% ./a.out ../Examples/example15.sil
0: { x:ERR; y:ERR; z:ERR; r:ERR }
1: { x:ERR; y:ERR; z:ERR; r:ERR }
if ((x = y) & (y = z)) then
  4: r := z
else {((x < y) | (y < x)) | ((y < z) | (z < y)))}
  6: r := 0
fi
6: { x:ZERO; y:INI; z:INI; r:INI }
```
- In this example, the test \((x = y)\) yields the information that \(y = 0\). Independently, the test \((y = z)\) brings no new information, on \(y\) and \(z\). The conjunction is
\[
\{ x: \text{ZERO}; \ y: \text{ZERO}; \ z: \text{INI}; \ r: \text{TOP} \}
\]
- The same analysis, starting from this valid information yields \(z = 0\).
- More generally, the analysis of the tests should be iterated until no new information can be brought in.
- The idea is formalized by noticing that the analysis of tests is a lower closure operator.

A theorem on the composition of lower closure operators and Galois connections

**Theorem.** If \(\langle P, \leq \rangle \xrightarrow{\gamma} \langle Q, \sqsubseteq \rangle\) and \(\rho\) is a lower closure operator on \(P\) (monotone, idempotent and reductive) then \(\alpha \circ \rho \circ \gamma\) is a lower closure operator on \(Q\).

**Proof.** Since \(\alpha \circ \rho \circ \gamma\) is the composition of monotone operators, it is monotone.
- Since \(\rho\) is reductive i.e. \(\forall x \in P : \rho(x) \leq x\), we have in particular \(\forall y \in Q : \rho(\gamma(y)) \leq \gamma(y)\) whence by monotony \(\forall y \in Q : \alpha(\rho(\gamma(y))) \sqsubseteq \alpha(\gamma(y)) \sqsubseteq y\) since \(\alpha \circ \rho \circ \gamma\) is reductive in a Galois connection. By transitivity, \(\forall y \in Q : \alpha \circ \rho \circ \gamma(y) \sqsubseteq y\) proving \(\alpha \circ \rho \circ \gamma\) to be reductive.

Abstraction of lower closure operators

- This theorem shows that whenever we have a lower closure operator in the concrete (e.g. the analysis of boolean expressions) then its abstraction \(\alpha \circ \rho \circ \gamma\) is also a lower closure operator in the abstract.
- Therefore, the abstract interpretation \(\overline{\rho}\) of the operator \(\alpha \circ \rho \circ \gamma\) can always be chosen to be a lower closure operator.
- In this case, if \(\overline{\rho}\) is not the best abstraction of \(\rho\), that is \(\alpha \circ \rho \circ \gamma \sqsubseteq \overline{\rho}\), \(\overline{\rho}\) can be improved by local decreasing iterations [1].

References

Local decreasing iterations

**Theorem.** If $(M, \leq)$ is poset, $f \in M \mapsto M$ is monotone and idempotent, $(M, \leq) \leftrightarrow (L, \sqsubseteq)$ is a Galois connection, $(L, \sqsubseteq, \sqcap)$ is a dual dcpo, $g \in L \mapsto L$ is monotone and reductive and $\alpha \circ f \circ \gamma \sqsubseteq g$ then the lower closure operator $g^* \overset{\text{def.}}{=} \lambda x \cdot \text{gfp}_x g$ is a better abstract interpretation of $f$ than $g$

\[ \alpha \circ f \circ \gamma \sqsubseteq g^* \overset{\text{def.}}{=} g. \]

**Proof.** For all $x \in L$, $g(x) \sqsubseteq x$, so that by monotony the sequence $g^n(x) \overset{\text{def.}}{=} x$, $g^{\delta+1}(x) \overset{\text{def.}}{=} g(g^\delta(x))$ for all successor ordinals $\delta + 1$ and $g^\lambda \overset{\text{def.}}{=} \bigcap_{\delta < \lambda} g^\delta(x)$ for all limit ordinals $\lambda$ is a well-defined decreasing chain in the dual dcpo $(L, \sqsubseteq, \sqcap)$ whence ultimately stationary. It converges to $g^\epsilon$ where $\epsilon$ is the order of $g$, which is the greatest fixpoint $g^\epsilon = \text{gfp}_x g$ of $g$ which is $\sqsubseteq$-less than $x$. It follows that $g^\epsilon \overset{\text{def.}}{=} \text{gfp}_x xg$ is the greatest lower closure operator $\sqsubseteq$-less than $g$. In particular $g^\epsilon \sqsubseteq g$.

We have $\alpha \circ f \circ \gamma(x) \sqsubseteq g^\delta(x) = g(x) \sqsubseteq x = g^\epsilon(x)$. If $\alpha \circ f \circ \gamma(x) \sqsubseteq g^\delta(x)$ then $\alpha \circ f \circ \gamma(x)$

\[ \subseteq \{ \text{f idempotent} \} \]

\[ \alpha \circ f \circ \gamma \overset{\text{def.}}{=} \alpha \circ f \circ \gamma(x) \]

\[ \subseteq \{ \text{by induction hypothesis, } \gamma, f \text{ and } \alpha \text{ are monotone} \} \]

The forward/bottom-up collecting semantics of boolean expressions is a lower closure operator

Recall the *collecting semantics* $\text{Cexp}[B]R$ of a boolean expression $B$ from course 8:

\[ \text{Cexp}[B]R \overset{\text{def.}}{=} \{ \rho \in R \mid \rho \vdash B \Rightarrow \text{tt} \} \] (44)

**Theorem.** $\text{Cexp}[B]$ is a lower closure operator.
The forward/top-down nonrelational abstract semantics of arithmetic expressions is monotone

Recall the forward/top-down nonrelational abstract semantics of arithmetic expressions

\[
\begin{align*}
\text{Faexp}^{+}[A](\lambda Y \cdot L) & \overset{\text{def}}{=} \bot & \text{if } \gamma(\bot) = \emptyset \\
\text{Faexp}^{+}[n]r & \overset{\text{def}}{=} n^\circ \\
\text{Faexp}^{+}[x]r & \overset{\text{def}}{=} r(x) \\
\text{Faexp}^{+}[?]r & \overset{\text{def}}{=} \gamma^\circ \\
\text{Faexp}^{+}[u A']r & \overset{\text{def}}{=} u^\circ (\text{Faexp}^{+}[A']r) \\
\text{Faexp}^{+}[A_1 \mathbin{b} A_2]r & \overset{\text{def}}{=} b^\circ (\text{Faexp}^{+}[A_1]r, \text{Faexp}^{+}[A_2]r)
\end{align*}
\]

\[
(45)
\]

\textbf{Theorem.} If \(u^\circ \in L \xrightarrow{m} L\) and \(b^\circ \in L \times L \xrightarrow{m} L\) then \(\text{Faexp}^{+}[A]\) is monotone.

\textbf{Proof.} - In the case \(\bot \subseteq r\), we have \(\text{Faexp}^{+}[A] \overset{\text{def}}{=} \bot \subseteq \text{Faexp}^{+}[A] r\)

\(\Box\)
The forward/top-down nonrelational abstract semantics of boolean expressions is monotone and reductive
We have defined
\[ \text{Abexp}[B] \equiv \alpha(\text{Cexp}[B]) \] such that
\[ \begin{align*}
\text{Abexp}[B] & \equiv \bot & \text{if } \gamma(\bot) = \emptyset \\
\text{Abexp}[\text{true}] & \equiv r \\
\text{Abexp}[\text{false}] & \equiv \top \\
\text{Abexp}[A_1 \land A_2] & \equiv \text{Abexp}[A_1] \land \text{Abexp}[A_2] \\
\text{Abexp}[B_1 \lor B_2] & \equiv \text{Abexp}[B_1] \lor \text{Abexp}[B_2] \\
\end{align*} \] (51)

parameterized by the following abstract comparison operations
\[ \mathcal{C}, c \in \{ <, = \} \] on \( L \)
\[ \begin{align*}
\mathcal{C}(p_1, p_2) & \equiv (\hat{B}(p_1, p_2) \equiv r \equiv \top) \\
\hat{B}(p_1, p_2) & \equiv \exists v_1 \in \gamma(p_1) : \exists v_2 \in \gamma(p_2) \cap I : v_1 \leq v_2 = \text{tt} \\
\end{align*} \] (52)

THEOREM. \text{Abexp}[B] is reductive and if \( \hat{B} \in \langle L, \sqsubseteq \rangle \times \langle L, \subseteq \rangle \xrightarrow{\equiv} \langle B, \equiv \rangle \) is monotonic then \text{Abexp}[B] is monotonic.

PROOF. – \text{Abexp}[B] is reductive. The proof is by structural induction on \( B \).
(a) For the infimum \( \bot \equiv \bot \equiv \text{Abexp}[B] \equiv \bot \)
(b) By reflexivity \( r \equiv r \equiv \text{Abexp}[\text{true}] \equiv r \)
(c) For the infimum \( r \equiv \top \equiv \text{Abexp}[\text{false}] \equiv r \)

(d) If \( \hat{B}(p_1, p_2) \) holds then \( r \sqsubseteq r \equiv \mathcal{C}(p_1, p_2) \). Otherwise \( \neg(\hat{B}(p_1, p_2)) \) holds and so \( r \sqsubseteq \bot \equiv \mathcal{C}(p_1, p_2) \). So \( \mathcal{C}(p_1, p_2) \) is reductive for all \( p_1 \) and \( p_2 \) and so in particular \( \text{Abexp}[A_1 \land A_2] \equiv \mathcal{C}(\text{Faexp}[A_1]r, \text{Faexp}[A_2]r) \) is reductive
(e) By induction hypothesis, \( \text{Abexp}[B_1] \equiv \rceil r \) and \( \text{Abexp}[B_2] \equiv r \) so \( \text{Abexp}[B_1 \lor B_2] \equiv \text{Abexp}[B_1] \lor \text{Abexp}[B_2] \equiv r \lor r = r \) by def. glb
(f) Similarly, \( \text{Abexp}[B_1 \land B_2] \equiv \text{Abexp}[B_1] \land \text{Abexp}[B_2] \equiv r \land r = r \) by ind. hyp. and def. lub.

- \text{Abexp}[B] is monotone. The proof is by structural induction on \( B \).
(a) if \( \bot \sqsubseteq r \) then \( \text{Abexp}[B] \equiv \bot \equiv \bot \equiv \text{Abexp}[B] \)
(b) \text{Abexp}[\text{true}] is the identity, which is monotone
(c) \text{Abexp}[\text{false}] is a constant function, which is monotone
(d) If \( r_1 \sqsubseteq r_2 \) then \( \text{Faexp}[A_1]r_1 \sqsubseteq \text{Faexp}[A_1]r_2 \) since \( \text{Faexp}[A_1]r, i = 1, 2 \) has been shown to be monotone. It follows by monotony of \( \hat{B} \) that if \( r_1 \sqsubseteq r_2 \) then \( \hat{B}(\text{Faexp}[A_1]r_1, \text{Faexp}[A_1]r_2) \equiv \hat{B}(\text{Faexp}[A_1]r_1, \text{Faexp}[A_1]r_2) \)
and so, by cases:
Reductive iterations for boolean expressions

Reductive iteration has a direct application to the analysis of boolean expressions. The abstract interpretation \( \text{Abexp}[B] \) (12) of boolean expressions \( B \) can always be replaced by its reductive iteration \( \text{Abexp}^*[B] \) which is sound (11) and always more precise.

By the local decreasing iterations Th. (page 85), we have

\[ \alpha(\text{Cbexp}[B]) \subseteq \text{Abexp}[B]^* \subseteq \text{Abexp}[B]. \]

\[ \text{PROOF. - } \text{Cbexp}[B] \text{ is a lower closure operator} \]
\[ \text{- Abexp}[B] \text{ is monotone and reductive (but not idempotent as shown for the motivating example)} \]
\[ \text{- So Abexp}[B]^* = \lambda x \cdot \text{gfp} x \text{ is a better abstraction of Cbexp}[B] than Abexp}[B] \]

Implementation of the reductive iterations for abstract interpretation of boolean expressions

\[ \text{let x' = (f x) in} \]
\[ \text{if (c x' x) then x'} \]
\[ \text{else lfp x' c f} \]
\[ (* \text{gfp x c f : iterative computation of the c-greatest fixpoint of *}) \]
\[ (* f, c-less than or equal to the postprefixpoint x (f(x) <= x) *) \]
\[ \text{let gfp x c f =} \]
\[ \text{let c_1 a b = c b a in} \]
\[ \text{ifp x c_1 f} \]
\[ (* \text{abexp.ml *}) \]
\[ \text{open Abstract_Syntax} \]
\[ \text{open Fixpoint} \]
\[ (* \text{abstract interpretation of boolean operations with iterative *}) \]
\[ (* \text{reduction *}) \]
\[ \text{let rec a_bexp' b r =} \]
\[ \text{match b with} \]
\[ \text{| TRUE -> r} \]
\[ \text{| FALSE -> (Aenv.bot ())} \]
\[ \text{| EQ (a1, a2) ->} \]

\[ \text{let (p1,p2) = (Avalues.a_EQ (Aaexp.a_aexp a1 r) (Aaexp.a_aexp a2 r))} \]
\[ \text{in (Aenv.meet (Baexp.b_aexp a1 r p1) (Baexp.b_aexp a2 r p2))} \]
\[ \text{let (p1,p2) = (Avalues.a_LT (Aaexp.a_aexp a1 r) (Aaexp.a_aexp a2 r))} \]
\[ \text{in (Aenv.meet (Baexp.b_aexp a1 r p1) (Baexp.b_aexp a2 r p2))} \]
\[ \text{| AND (b1, b2) -> (Aenv.meet (a_bexp' b1 r) (a_bexp' b2 r))} \]
\[ \text{| OR (b1, b2) -> (Aenv.join (a_bexp' b1 r) (a_bexp' b2 r))} \]
\[ \text{let a_bexp b r =} \]
\[ \text{if (Aenv.is_bot r) & (Avalues.isbotempty ()) then (Aenv.bot ())} \]
\[ \text{else gfp r Aenv.leq (a_bexp' b)} \]
Motivating example ... revisited

% cd Initialization-Simple-Sign-FB-LDI-B
% ./a.out ../Examples/example15.sil
0: \{ x:ERR; y:ERR; z:ERR; r:ERR \}
x := 0; y := ?; z := ?;
if ((x = y) & (y = z)) then
  4: r := z
  5: 
else \{(((x < y) | (y < x)) | ((y < z) | (z < y)))\}
  6: r := 0
  7: 
fi
8: \{ x:ZERO; y:INI; z:INI; r:ZERO \}

Motivating example

% cd Initialization-Simple-Sign-FB-LDI-B
% ./a.out ../Examples/example18.sil
\{ x:ERR; y:ERR \}
0: x := ?;
1: y := (1 / x)
2: \{ x:INI; y:INI \}

Although the program execution is blocked at line 1:
when $x \leq 0$ (the division requires its second argument
to be strictly positive), this fact is not taken into account
ar line 2: since the abstract assignment
\[
A\text{com}[X := A]R(\text{after}_P[X := A]) = R[X := \text{Faexp}[A](R) \cap ?]
\]
does not restricts the values of variables (other than $X$)
in environment $R$ to those for which the expression $A$ is
well-defined. (its values belonging to $I$, which excludes
those errors for which the execution stops).
Revisiting the forward/bottom-up nonrelational abstract interpretation of assignments using the backward analysis of arithmetic expressions.

By restricting the values of the variables to those for which the expression $A$ is well-defined, (its values belonging to $\mathbb{I}$, which excludes those errors for which the execution stops), we get

**Theorem.**

\[
\alpha[X := A](\text{Rcom}[X := A]) \supseteq \text{Acom}[X := A](\text{Baexp}[A](R) \ldots)(\text{after}_p[X := A])
\]

**Proof.** By (13), we have $\alpha[X := A](\text{Rcom}[X := A]) \supseteq \text{Acom}[X := A]$ whence by def. (9) of $\alpha[x := A]|\varphi \triangleq \lambda r \cdot \lambda \ell \cdot \alpha(\varphi(\gamma(\ell)))(\ell)$, we have for all $r$ and $\ell$, $\alpha(\text{Rcom}[x := A](\gamma(\ell)))(\ell) \supseteq \text{Acom}[x := A](\ell)$. In particular for $r = \text{Baexp}[A](r')$, we get $\forall \ell' \cdot \text{Rcom}[x := A](\gamma(\text{Baexp}[A](r')))(\ell) \subseteq \text{Acom}[x := A](\ell')$.

By (33), we have $\alpha'(\text{Baexp}[A]) \supseteq \text{Baexp}'[A]$ so that by def. (32) of $\alpha'(\Phi) \triangleq \lambda r \cdot \lambda p \cdot \alpha(\Phi(\gamma(p)))(p)$, we have $\forall p : \alpha'(\text{Baexp}[A](\gamma(p)))(p) \subseteq \text{Baexp}'[A](p)$ whence for $p = ?$ such that $\gamma(?)$ def $\mathbb{I}$ we get $\forall r : \alpha'(\text{Baexp}[A](\gamma(?) )) \subseteq \text{Baexp}'[A](r')$. It follows, by the Galois connection property (4), that $\forall r' : \text{Baexp}[A](\gamma(?) ) \subseteq (\text{Baexp}'[A](r'))$.

In the abstract, we have:

\[
\alpha[C](\text{Rcom}[C]) \supseteq \text{Acom}[C]
\]

It follows by monotony of $\alpha$ and $\text{Rcom}[x := A]$ that $\alpha(\text{Rcom}[x := A](\text{Baexp}[A](r' ))(\ell)) \supseteq \alpha'(\text{Rcom}[x := A](\gamma(\text{Baexp}[A](r') )))(\ell)$ and so we conclude

\[
\alpha[X := A](\text{Rcom}[X := A]) \supseteq \text{Acom}[X := A](\text{Baexp}[A](r' ))(\ell)
\]

\[
\alpha[X := A](\text{Rcom}[X := A]) \supseteq \text{Acom}[X := A](\text{Baexp}[A](r' ))(\ell)
\]

\[
\alpha[X := A](\text{Rcom}[X := A]) \supseteq \text{Acom}[X := A](\text{Baexp}[A](r' ))(\ell)
\]

\[
\alpha[X := A](\text{Rcom}[X := A]) \supseteq \text{Acom}[X := A](\text{Baexp}[A](r' ))(\ell)
\]

\[
\alpha[X := A](\text{Rcom}[X := A]) \supseteq \text{Acom}[X := A](\text{Baexp}[A](r' ))(\ell)
\]

\[
\alpha[X := A](\text{Rcom}[X := A]) \supseteq \text{Acom}[X := A](\text{Baexp}[A](r' ))(\ell)
\]

\[
\alpha[X := A](\text{Rcom}[X := A]) \supseteq \text{Acom}[X := A](\text{Baexp}[A](r' ))(\ell)
\]

\[
\alpha[X := A](\text{Rcom}[X := A]) \supseteq \text{Acom}[X := A](\text{Baexp}[A](r' ))(\ell)
\]
Implementation of the revisited forward/bottom-up nonrelational abstract interpretation of assignments using the backward analysis of arithmetic expressions

(* acom.ml *)
open Abstract_Syntax
open Labels
open Aenv
open Aaexp
open Abexp
open Fixpoint
open Baexp
(* forward abstract semantics of commands *)

exception Error of string
let rec acom c r l =
match c with
| (SKIP (l', l'')) ->
  if (l = l') then r
  else if (l = l'') then r
  else (raise (Error "SKIP incoherence"))
| (ASSIGN (l', x, a, l'')) ->
  if (l = l') then r
  else if (l = l'') then r
  else (raise (Error "ASSIGN incoherence"))
| (SEQ (l', s, l'')) ->
  (acomseq s r l)
| (IF (l', b, nb, t, f, l'')) ->
  if (l = l') then r
  else if (incom l t) then
    (acom t (a_bexp b r) l)
  else if (incom l f) then
    (acom f (a_bexp nb r) l)
  else if (l = l'') then
    (join (acom t (a_bexp b r) (after t))
    (acom f (a_bexp nb r) (after f)))
  else (raise (Error "IF incoherence"))
| (WHILE (l', b, nb, c', l'')) ->
  let f x = join r (acom c' (a_bexp b x) (after c'))
  in let i = 1fp (bot ()). leq f in
  if (l = l') then i
  else if (incom l c') then (acom c' (a_bexp b i) l)
  else if (l = l'') then (a_bexp nb i)
  else (raise (Error "WHILE incoherence"))
and acomseq s r l = match s with
| [] -> raise (Error "empty SEQ incoherence")
| [c] -> if (incom l c) then (acom c r l)
| h::t -> if (incom l h) then (acom h r l)
  else (acomseq t (acom h r (after h)) l)
Motivating example ... revisited

```bash
% cd Initialization-Simple-Sign-FB-LDI-BA
% ./a.out ../Examples/example18.sil
{ x:ERR; y:ERR }
0:
  x := ?;
1:
  y := (1 / x)
2:
{ x:POS; y:INI }
```

Personal project: homework 2

- Change file `values.ml` of the non-relational initialization and simple sign abstract interpreters with backward analysis of expressions discussed in this lecture to implement the finitary analysis that you have chosen.

Bibliography


THE END