Abstract Interpretation
Project Report

—

Abstract Debugging of the Three Counter Machine using Forward and Backward Analysis over the Interval Domain

Pierre Chatelain

–

Department of Computer Science, University of Aarhus
Aabogade 34, 8200 Århus N, Denmark

June 10, 2011

1 Introduction

Abstract debugging, as introduced in [1], consists in the static debugging of programs using the tools of abstract interpretation. One can with this technique predict certain run-time errors and their source at compile-time.

The goal of this project is to apply abstract debugging to Plotkin's three counter machine [2], using the domain of intervals over the natural numbers as abstraction.

We will look for several classes of bugs (termination, reachability, invalid argument), and try to give a precise informations about their nature and the input values for wich they can occur.

2 Definitions

2.1 The three counter machine

One can find various definitions of the three counter machine in the litterature. We will use here Plotkin's three counter machine [2], defined as follows by:

- three variables $var \in Var = \{x, y, z\}$;
- a program counter $pc \in PC = \mathbb{N}^*$;
- a set of instruction $Inst =$ inc var | dec var | zero var pc' else pc'' | stop
  where $var \in Var$ and $(pc', pc'') \in PC^2$;

- the set of configurations $States = PC \times \mathbb{N} \times \mathbb{N} \times \mathbb{N}$

where $\mathbb{N}$ is the set of positive integers and $\mathbb{N}^* = \mathbb{N} \setminus \{0\}$.

A three counter machine program is then $P \in Inst^{PC}$ of instructions, such that for $pc \in PC$, $P_{pc} \in Inst$, and $P_{PC} = \text{stop}$.

This program defines a *transition system*, with the initial states $S_0 = \{\langle 1, x_0, 0, 0\rangle \mid i \in \mathbb{N}\}$, the final states $S_f = \{\langle pc, 0, y_f, 0\rangle \mid P_{pc} = \text{stop} \wedge y_f \in \mathbb{N}\}$, and the following transition relation:

$$T(\langle pc, xv, yv, zv\rangle) \rightarrow \begin{cases} \langle pc+1, xv+1, yv, zv\rangle & \text{if } P_{pc} = \text{inc } x \\ \langle pc+1, xv, yv+1, zv\rangle & \text{if } P_{pc} = \text{inc } y \\ \langle pc+1, xv, yv, zv+1\rangle & \text{if } P_{pc} = \text{inc } z \\ \langle pc+1, xv-1, yv, zv\rangle & \text{if } P_{pc} = \text{dec } x \wedge xv > 0 \\ \langle pc+1, xv, yv-1, zv\rangle & \text{if } P_{pc} = \text{dec } y \wedge yv > 0 \\ \langle pc+1, xv, yv, zv-1\rangle & \text{if } P_{pc} = \text{dec } z \wedge zv > 0 \\ \langle pc', xv, yv, zv\rangle & \text{if } P_{pc} = \text{zero } x\ pc' \text{ else } pc'' \wedge xv = 0 \\ \langle pc'', xv, yv, zv\rangle & \text{if } P_{pc} = \text{zero } x\ pc' \text{ else } pc'' \wedge xv \neq 0 \\ \langle pc', xv, yv, zv\rangle & \text{if } P_{pc} = \text{zero } y\ pc' \text{ else } pc'' \wedge yv = 0 \\ \langle pc'', xv, yv, zv\rangle & \text{if } P_{pc} = \text{zero } y\ pc' \text{ else } pc'' \wedge yv \neq 0 \\ \langle pc', xv, yv, zv\rangle & \text{if } P_{pc} = \text{zero } z\ pc' \text{ else } pc'' \wedge zv = 0 \\ \langle pc'', xv, yv, zv\rangle & \text{if } P_{pc} = \text{zero } z\ pc' \text{ else } pc''x \wedge zv \neq 0 \end{cases} \tag{1}$$

An execution of the program, given an input value x_0, is the sequence of states obtained when starting from $\langle 1, x_0, 0, 0\rangle$ and applying successively the transition relation. This sequence is unique as the transition system is deterministic. We say that the program *terminates* if this execution is finite and ends in a state $\langle pc, x_f, y_f, z_f\rangle \in S_f$ with $P_{pc} = \text{stop}$. If the program terminates and the last state is $\langle pc, 0, y_f, 0\rangle \in S_f$, then the result is y_f.

2.2 The Abstract Interval Domain

We define the abstract interval domain in the same way as in Miné's thesis [3], but considering only intervals over $\mathbb{N}$:

$$Interval = \{\bot\} \cup \{[l, u] \mid l \in \mathbb{N} \wedge u \in \mathbb{N} \cup \{+\infty\} \wedge l \leq u\} \tag{2}$$

This set is partially ordered using the following order relation:

$$\forall I \in Interval \qquad \bot \sqsubseteq I \tag{3}$$

$$\forall l, u, l', u' \in \mathbb{N} \mid l \geq l' \wedge u \leq u' \quad [l, u] \sqsubseteq [l', u'] \tag{4}$$

The union $\sqcup$ and disjonction $\sqcap$ are then defined as:

$$\forall I \in Interval \qquad I \sqcup \bot = \bot \sqcup I = I \tag{5}$$

$$\forall l, u, l', u' \in \mathbb{N} \qquad [l, u] \sqcup [l', u'] = [\min(l, l'), \max(u, u')] \tag{6}$$

$$\forall I \in Interval \qquad I \sqcap \bot = \bot \sqcap I = \bot \tag{7}$$

$$\forall l, u, l', u' \in \mathbb{N} \quad [l, u] \sqcap [l', u'] \begin{cases} [\max(l, l'), \min(u, u')] & \text{if } \max(l, l') \leq \min(u, u') \\ \bot & \text{otherwise} \end{cases} \tag{8}$$

As the interval domain contains infinite strictly increasing chains, we will also need to define widening operator:

$$\forall I \in \mathit{Interval} \qquad I \nabla \bot = \bot \nabla I = I \tag{9}$$

$$\forall l, u, l', u' \in \mathbb{N} \quad [l,u] \nabla [l',u'] = \left[\min(l,l'), \begin{cases} +\infty & \text{if } u' > u \\ u & \text{if } u' \leq u \end{cases} \right] \tag{10}$$

3 Abstracting the three counter machine

3.1 Construction of the Galois connection

In order to analyse the three counter machine programs, we will abstract the reachable states collecting semantics with the interval domain, using the following Galois connection:

$$\langle \wp(\mathbb{N}); \subseteq \rangle \underset{\alpha_{int}}{\overset{\gamma_{int}}{\leftrightarrows}} \langle \mathit{Interval}; \sqsubseteq \rangle \tag{11}$$

where

$$\begin{aligned} \gamma_{int}(\bot) &= \emptyset & (12)\\ \gamma_{int}([a,b]) &= \{n \in \mathbb{N} \mid a \leq n \leq b\} & (13)\\ \alpha_{int}(\emptyset) &= \bot & (14)\\ \alpha_{int}(S) &= [\min S, \max S] & (15) \end{aligned}$$

As the machine has three registers, we need to abstract $\wp(\mathbb{N} \times \mathbb{N} \times \mathbb{N})$. Therefore we compose the two following Galois connections:

$$\begin{aligned} \langle \wp(\mathbb{N} \times \mathbb{N} \times \mathbb{N}); \subseteq \rangle &\underset{\alpha_1}{\overset{\gamma_1}{\leftrightarrows}} \langle \wp(\mathbb{N}) \times \wp(\mathbb{N}) \times \wp(\mathbb{N}); \dot{\subseteq} \rangle & (16)\\ \langle \wp(\mathbb{N}) \times \wp(\mathbb{N}) \times \wp(\mathbb{N}); \dot{\subseteq} \rangle &\underset{\alpha_2}{\overset{\gamma_2}{\leftrightarrows}} \langle \mathit{Interval} \times \mathit{Interval} \times \mathit{Interval}; \dot{\sqsubseteq} \rangle & (17) \end{aligned}$$

where $\dot{\subseteq}$ and $\dot{\sqsubseteq}$ are the respective pointwise versions of $\dot{\subseteq}$ and $\dot{\sqsubseteq}$, and the abstractions and concretization functions are defined as follows:

$$\begin{aligned} \alpha_1(T) &= \langle \pi_1(T), \pi_2(T), \pi_3(T) \rangle & (18)\\ \gamma_1(\langle X, Y, Z \rangle) &= X \times Y \times Z & (19)\\ \alpha_2(\langle X, Y, Z \rangle) &= \langle \alpha_{int}(X), \alpha_{int}(Y), \alpha_{int}(Z) \rangle & (20)\\ \gamma_2(\langle X^\#, Y^\#, Z^\# \rangle) &= \langle \gamma_{int}(X^\#), \gamma_{int}(Y^\#), \gamma_{int}(Z^\#) \rangle & (21)\\ & & (22) \end{aligned}$$

This composition gives a new Galois connection:

$$\langle \wp(\mathbb{N} \times \mathbb{N} \times \mathbb{N}); \subseteq \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle \mathit{Interval} \times \mathit{Interval} \times \mathit{Interval}; \dot{\sqsubseteq} \rangle \tag{23}$$

which we use in a final step to abstract $PC \longrightarrow \wp(\mathbb{N} \times \mathbb{N} \times \mathbb{N})$:

$$\langle PC \longrightarrow \wp(\mathbb{N} \times \mathbb{N} \times \mathbb{N}); \subseteq \rangle \underset{\dot{\alpha}}{\overset{\dot{\gamma}}{\leftrightarrows}} \langle PC \longrightarrow \mathit{Interval} \times \mathit{Interval} \times \mathit{Interval}; \dot{\sqsubseteq} \rangle \tag{24}$$

where $\dot{\alpha}$ and $\dot{\gamma}$ are the respective pointwise versions of α and γ.

3.2 Abstract operations

The execution of the three counter machine involves the following operations on $\wp(\mathbb{N})$:

$$\begin{aligned} &: && \wp(\mathbb{N}) \to \wp(\mathbb{N}) \\ \text{=0} &= && \lambda S.\{s|s \in S \wedge s = 0\} \\ \text{<>0} &= && \lambda S.\{s|s \in S \wedge s \neq 0\} \\ \text{+1} &= && \lambda S.\{s+1|s \in S\} \\ \text{-1} &= && \lambda S.\{s-1|s \in S \wedge s > 0\} \end{aligned} \tag{25}$$

In order to derive the abstract transition function, we need to abstract this four operators. We get optimal abstractions by composing this operators with α_{int} and γ_{int}:

- Abstraction of =0:

$$\begin{aligned} \text{=0}^{int} &= \lambda a.\alpha_{int} \circ \text{=0} \circ \gamma_{int}(a) \\ &= \lambda a.\alpha_{int} \circ \lambda S.\{s|s \in S \wedge s = 0\} \circ \gamma_{int}(a) \\ &= \lambda a.\alpha_{int}(\{s|s \in \gamma_{int}(a) \wedge s = 0\}) \\ &= \lambda a.\begin{cases} \alpha_{int}(\{s|s \in \emptyset \wedge s = 0\}) & \text{if } a = \bot \\ \alpha_{int}(\{s|l \leq s \leq u \wedge s = 0\}) & \text{if } a = [l,u] \end{cases} \\ &= \lambda a.\begin{cases} \alpha_{int}(\emptyset) & \text{if } a = \bot \vee (a = [l,u] \wedge l > 0) \\ \alpha_{int}(\{0\}) & \text{if } a = [0,u] \end{cases} \\ &= \lambda a.\begin{cases} \bot & \text{if } a = \bot \vee (a = [l,u] \wedge l > 0) \\ [0,0] & \text{if } a = [0,u] \end{cases} \end{aligned}$$

- Abstraction of <>0:

$$\begin{aligned} \text{<>0}^{int} &= \lambda a.\alpha_{int} \circ \text{<>0} \circ \gamma_{int}(a) \\ &= \lambda a.\alpha_{int} \circ \lambda S.\{s|s \in S \wedge s \neq 0\} \circ \gamma_{int}(a) \\ &= \lambda a.\alpha_{int}(\{s|s \in \gamma_{int}(a) \wedge s \neq 0\}) \\ &= \lambda a.\begin{cases} \alpha_{int}(\{s|s \in \emptyset \wedge s \neq 0\}) & \text{if } a = \bot \\ \alpha_{int}(\{s|l \leq s \leq u \wedge s \neq 0\}) & \text{if } a = [l,u] \end{cases} \\ &= \lambda a.\begin{cases} \alpha_{int}(\emptyset) & \text{if } a = \bot \vee a = [0,0] \\ \alpha_{int}(\{s|1 \leq s \leq u\}) & \text{if } a = [0,u] \\ \alpha_{int}(\{s|l \leq s \leq u\}) & \text{if } a = [l,u] \wedge l > 0 \end{cases} \\ &= \lambda a.\begin{cases} \bot & \text{if } a = \bot \vee a = [0,0] \\ [1,u] & \text{if } a = [0,u] \\ [l,u] & \text{if } a = [l,u] \wedge l > 0 \end{cases} \end{aligned}$$

- Abstraction of +1:

$$\begin{array}{rcl}
+1^{int} & = & \lambda a.\alpha_{int} \circ +1 \circ \gamma_{int}(a) \\
& = & \lambda a.\alpha_{int} \circ \lambda S.\{s+1 | s \in S\} \circ \gamma_{int}(a) \\
& = & \lambda a.\alpha_{int}(\{s+1 | s \in \gamma_{int}(a)\}) \\
& = & \lambda a.\left\{ \begin{array}{ll} \alpha_{int}(\{s+1|s \in \emptyset) & \text{if } a = \bot \\ \alpha_{int}(\{s+1|l \le s \le u) & \text{if } a = [l,u] \end{array} \right. \\
& = & \lambda a.\left\{ \begin{array}{ll} \alpha_{int}(\emptyset) & \text{if } a = \bot \\ \alpha_{int}(\{s|l+1 \le s \le u+1\}) & \text{if } a = [l,u] \wedge u < +\infty \\ \alpha_{int}(\{s|l+1 \le s\}) & \text{if } a = [l,+\infty] \end{array} \right. \\
& = & \lambda a.\left\{ \begin{array}{ll} \bot & \text{if } a = \bot \\ [l+1,u+1] & \text{if } a = [l,u] \wedge u < +\infty \\ [l+1,+\infty] & \text{if } a = [l,+\infty] \end{array} \right.
\end{array}$$

- Abstraction of -1:

$$\begin{array}{rcl}
-1^{int} & = & \lambda a.\alpha_{int} \circ -1 \circ \gamma_{int}(a) \\
& = & \lambda a.\alpha_{int} \circ \lambda S.\{s-1 | s \in S \wedge s > 0\} \circ \gamma_{int}(a) \\
& = & \lambda a.\alpha_{int}(\{s-1 | s \in \gamma_{int}(a) \wedge s > 0\}) \\
& = & \lambda a.\left\{ \begin{array}{ll} \alpha_{int}(\{s-1|s \in \emptyset \wedge s > 0) & \text{if } a = \bot \\ \alpha_{int}(\{s-1|l \le s \le u \wedge s > 0) & \text{if } a = [l,u] \end{array} \right. \\
& = & \lambda a.\left\{ \begin{array}{ll} \alpha_{int}(\emptyset) & \text{if } a = \bot \vee a = [0,0] \\ \alpha_{int}(\{s|0 \le s \le u-1\}) & \text{if } a = [0,u] \wedge u > 0 \\ \alpha_{int}(\{s|l-1 \le s \le u-1\}) & \text{if } a = [l,u] \wedge l > 0 \end{array} \right. \\
& = & \lambda a.\left\{ \begin{array}{ll} \bot & \text{if } a = \bot \vee a = [x,x] \\ [0,u-1] & \text{if } a = [0,u] \wedge u > 0 \\ [l-1,u-1] & \text{if } a = [l,u] \wedge u > l > 0 \end{array} \right.
\end{array}$$

We can now use this operators to abstract the following operations on the triplets (with similar definitions for y and z):

$$\begin{array}{rcl}
 & : & \wp(\mathbb{N} \times \mathbb{N} \times \mathbb{N}) \to \wp(\mathbb{N} \times \mathbb{N} \times \mathbb{N}) \\
\text{[x++]} & = & \lambda S.\{\langle xv+1, yv, zv\rangle | \langle xv, yv, zv\rangle \in S\} \\
\text{[x--]} & = & \lambda S.\{\langle xv-1, yv, zv\rangle | \langle xv, yv, zv\rangle \in S \wedge xv > 0\} \\
\text{[x=0]} & = & \lambda S.\{\langle xv, yv, zv\rangle | \langle xv, yv, zv\rangle \in S \wedge xv = 0\} \\
\text{[x<>0]} & = & \lambda S.\{\langle xv, yv, zv\rangle | \langle xv, yv, zv\rangle \in S \wedge xv > 0\}
\end{array} \quad (26)$$

The optimal abstract versions are once again computed by composition with the abstraction and concretization functions. The calculations are straightforward and give the following abstract operators:

$$\begin{aligned}
&: \quad \mathit{Interval} \times \mathit{Interval} \times \mathit{Interval} \to \mathit{Interval} \times \mathit{Interval} \times \mathit{Interval} \qquad (27)\\
[\text{x++}]^{int} &= \lambda\langle xv, yv, zv\rangle.\langle +1^{int}(xv), yv, zv\rangle\\
[\text{x--}]^{int} &= \lambda\langle xv, yv, zv\rangle.\langle -1^{int}(xv), yv, zv\rangle\\
[\text{x=0}]^{int} &= \lambda\langle xv, yv, zv\rangle.\langle = 0^{int}(xv), yv, zv\rangle\\
[\text{x<>0}]^{int} &= \lambda\langle xv, yv, zv\rangle.\langle <> 0^{int}(xv), yv, zv\rangle
\end{aligned}$$

and same for y and z.

3.3 Abstract transition functions

Using the method of *pushing alphas*, we can now derive the transition function on the interval domain. The complete calculation is not shown in this report because quite long, and similar to the abstraction of the transition function over the *Parity* domain calculated in class. The resulting function is:

$$\begin{aligned}
F^{int}(S^{int}) \quad &= \quad \dot\perp.\,[1 \to \langle[0,+\infty],[0,0],[0,0]\rangle]\\
&\dot\cup \dot{\bigcup_{\substack{pc \in Dom(S^{int})\\ P_{pc} = \text{inc } var\\ var \in Var}}} \dot\perp.\,[pc+1 \to [var++]^{int} \circ S^{int}(pc)]\\
&\dot\cup \dot{\bigcup_{\substack{pc \in Dom(S^{int})\\ P_{pc} = \text{dec } var\\ var \in Var}}} \dot\perp.\,[pc+1 \to [var--]^{int} \circ S^{int}(pc)]\\
&\dot\cup \dot{\bigcup_{\substack{pc \in Dom(S^{int})\\ P_{pc} = \text{zero } var\ pc' \text{ else } pc''\\ var \in Var}}} \begin{array}{l} \dot\perp.\,[pc' \to [var = 0]^{int} \circ S^{int}(pc)]\\ \dot\cup\ \dot\perp.\,[pc'' \to [var <> 0]^{int} \circ S^{int}(pc)]\end{array} \qquad (28)
\end{aligned}$$

Similarly, the abstract backward transition function can be written as:

$$\begin{aligned}
B^{int}(S^{int}) \quad &= \quad \dot\perp.\\
&\dot\cup \dot{\bigcup_{\substack{pc \in Dom(S^{int})\\ P_{pc} = \text{inc } var\\ var \in Var}}} \dot\perp.\,[pc \to [var--]^{int} \circ S^{int}(pc+1)]\\
&\dot\cup \dot{\bigcup_{\substack{pc \in Dom(S^{int})\\ P_{pc} = \text{dec } var\\ var \in Var}}} \dot\perp.\,[pc \to [var++]^{int} \circ S^{int}(pc+1)]\\
&\dot\cup \dot{\bigcup_{\substack{pc \in Dom(S^{int})\\ P_{pc} = \text{zero } var\ pc' \text{ else } pc''\\ var \in Var}}} \begin{array}{l} \dot\perp.\,[pc \to [var = 0]^{int} \circ S^{int}(pc')]\\ \dot\cup\ \dot\perp.\,[pc \to [var <> 0]^{int} \circ S^{int}(pc'')]\end{array} \qquad (29)
\end{aligned}$$

Once again, the complete calculation is not detailed in the report, because it seems more interesting to look at the meaning of this definition, and see how the backward operations can be intuitively deduced from the forward operations.

4 Debugging

4.1 Reachability

The easiest bug we can look for is *dead code*, *ie.* parts of the program that are not reachable. This can indeed be done by computing a simple forward analysis $lfp(F^{int})$, by kleen iteration with widening. Then, for $pc \in PC$, the pc^{th} line of the program is not reachable if:

$$lfp(F^{int})(pc) = \langle XV, YV, ZV \rangle \wedge (XV = \perp \vee YV = \perp \vee ZV = \perp) \tag{30}$$

This computation can be improved by quotienting in the interval triplets domain all the triplets with at least one element equal to $\perp$, into the new bottom $\perp_3 = \langle \perp, \perp, \perp \rangle$ of this domain. This gives a more precise analysis, as shows table 4.1.

1:	inc x	$\langle[0,\infty],[0,0],[0,0]\rangle$	$\langle[0,\infty],[0,0],[0,0]\rangle$
2:	zero x 3 else 4	$\langle[1,\infty],[0,0],[0,0]\rangle$	$\langle[1,\infty],[0,0],[0,0]\rangle$
3:	inc y	$\langle\perp,[0,0],[0,0]\rangle$	$\perp_3$
4:	stop	$\langle[1,\infty],[\mathbf{0},\mathbf{1}],[0,0]\rangle$	$\langle[1,\infty],[\mathbf{0},\mathbf{0}],[0,0]\rangle$

Table 1: forward analysis with the complete interval triplet domain (second column) and the improved one (third column)

With this modification, the forward analysis finds the exact set of unreachable lines.

4.2 Termination

A special case of reachability is when the last line is not reachable, and therefore the program does not terminate. As this can occur either for any input or only for certain input values, we would like to report an approximation of the input values for which the program will not terminate.

To compute such a non-trivial approximation, we need to perform a forward/backward analysis, as proposed in [1]. For pc such that $P_{pc} =$ stop, we define the intermittent assertion $\Pi_{stop} = \perp_3.\ [pc- > \langle[0,\infty],[0,\infty],[0,\infty]\rangle]$, and we compute the analysis as the limit of the sequence:

$$\begin{aligned} I_0 &= lfp(F^{int}) && (31)\\ I_{3k+1} &= gfp(\lambda X.\ I_{3k} \cap B^{int}) && (32)\\ I_{3k+2} &= lfp(\lambda X.\ I_{3k+1} \cap (\Pi_{stop} \cup B^{int}(X))) && (33)\\ I_{3k+3} &= lfp(\lambda X.\ I_{3k+2} \cap F^{int}(X)) && (34) \end{aligned}$$

Noting I_∞ the limit of this sequence and $\langle X_{stop}, Y_{stop}, Z_{stop}\rangle = I_\infty(1)$, $\gamma^{int}(X_{stop})$ approximates the set of input values for which the program terminates.

We also check if the program ends in a final state, using, for P_{pc} = stop the invariant assertion $\Pi_{final} = \langle[0,\infty],[0,\infty],[0,\infty]\rangle.\ [pc- > \langle[0,0],[0,\infty],[0,0]\rangle]$, and computing the limit J_∞ of the following sequence:

$$\begin{aligned}
J_0 &= lfp(F^{int}) && (35)\\
J_{3k+1} &= gfp(\lambda X.\ J_{3k} \cap \Pi_{final} \cap B^{int}(X))) && (36)\\
J_{3k+2} &= lfp(\lambda X.\ J_{3k+1} \cap B^{int}(X))) && (37)\\
J_{3k+3} &= lfp(\lambda X.\ J_{3k+2} \cap F^{int}(X)) && (38)
\end{aligned}$$

If we note $J_\infty = \langle X_{final}, Y_{final}, Z_{final}\rangle$, then $\gamma^{int}(X_{final})$ approximates the set of input values for which the program ends in a final state.

4.3 Decrement operations

To check that the decrement operations are not performed on registers with value 0, we define the invariant assertion:

$$\begin{aligned}
\Pi_{dec} &= \langle[0,\infty],[0,\infty],[0,\infty]\rangle.\\
&\dot\cap\ \dot\bigcap_{P_{pc}=\text{dec } x}\langle[0,\infty],[0,\infty],[0,\infty]\rangle.\ [pc- > \langle[1,\infty],[0,\infty],[0,\infty]\rangle]\\
&\dot\cap\ \dot\bigcap_{P_{pc}=\text{dec } y}\langle[0,\infty],[0,\infty],[0,\infty]\rangle.\ [pc- > \langle[0,\infty],[1,\infty],[0,\infty]\rangle]\\
&\dot\cap\ \dot\bigcap_{P_{pc}=\text{dec } z}\langle[0,\infty],[0,\infty],[0,\infty]\rangle.\ [pc- > \langle[0,\infty],[0,\infty],[1,\infty]\rangle] && (39)
\end{aligned}$$

and we compute the forward/backward analysis as the limit J_∞ of the sequence:

$$\begin{aligned}
K_0 &= lfp(F^{int}) && (40)\\
K_{3k+1} &= gfp(\lambda X.\ K_{3k} \cap \Pi_{dec} \cap B^{int}(X))) && (41)\\
K_{3k+2} &= lfp(\lambda X.\ K_{3k+1} \cap \cap B^{int}(X))) && (42)\\
K_{3k+3} &= lfp(\lambda X.\ K_{3k+2} \cap F^{int}(X)) && (43)
\end{aligned}$$

If $K_\infty = \langle X_{dec}, Y_{dec}, Z_{dec}\rangle$, then $\gamma^{int}(X_{dec})$ approximates the set of input values for which the program does not contain any invalid decrement operation.

5 Experiments and conclusion

The algorithms described above have been implemented in OCaml (building on the initial three counter machine source code provided by Jan Midtgaard), and tested on a benchmark of 3cm programs covering the different kinds of errors [1].

[1]the complete source code and the test files are available at http://www.pierre.chatelain.eu/au/absint/project

The results are for most of the cases satisfying, as they give the most precise information we could have using intervals. It succefully discovers infinite loops, invalid decrements operations and unreachable lines as long as these errors are localized in a single register.

An error can be undetected if it involves involves dependences between two registers. The program *test9.3cm* (table 5) illustrates this, as it involves an invalid decrement operation if and only if the input value is smaller than 2. The abstract debugger fails to detect this error because the following operation looses information:

$$\langle X, Y, Z\rangle \sqcup_3 \langle X', Y', Z'\rangle = \langle X \sqcup X', Y \sqcup Y', Z \sqcup Z'\rangle \tag{44}$$

```
1:  zero x 5 else 2
2:  inc y
3:  dec x
4:  zero x 5 else 2
5:  dec y
6:  dec y
7:  stop
```

Table 2: test9.3cm

This problem is however inherent to the abstraction of triplets we used, and besides this our abstract debugger seems to work well.

The determination of the source of the bugs is satisfying, although for some cases where several potential sources of errors interact it is difficult to give a precise description of the problem. A more thorough design of the assertions and analysis could give more precise information.

References

[1] F. Bourdoncle, "Abstract debugging of higher-order imperative languages," *SIGPLAN Not.*, vol. 28, pp. 46–55, June 1993. [Online]. Available: http://doi.acm.org/10.1145/173262.155095

[2] G. D. Plotkin, *A Structural Approach to Operational Semantics*. Journal of Logic and Algebraic Programming, 2004. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.163.4722&rep=rep1&type=pdf

[3] A. Miné, "Weakly relational numerical abstract domains," Ph.D. dissertation, École Polytechnique, Palaiseau, France, December 2004, http://www.di.ens.fr/~mine/these/these-color.pdf.