Gradual Program Verification

Johannes Bader[1], Jonathan Aldrich[2], and Éric Tanter[3]

[1] Microsoft Corporation, Redmond, USA,
jobader@microsoft.com
[2] Institute for Software Research, Carnegie Mellon University, Pittsburgh, USA
jonathan.aldrich@cs.cmu.edu
[3] PLEIAD Lab, Computer Science Dept (DCC), University of Chile, Santiago, Chile
etanter@dcc.uchile.cl

A Supplementary Material

A.1 Prototype Implementation

We have implemented a gradual verifier realizing GVL_{IDF} as a web application at `http://olydis.github.io/GradVer/impl/HTML5wp/`.

We implemented a superset of the statements specified for GVL_{IDF}: The prototype also features a cast statement, as well as "release" and "hold" statements to give programmers explicit control over releasing and framing off access to certain fields. In particular, hold allows bounding imprecision, *e.g.* by making sure that an imprecise contract may in fact not acquire access to *all* fields that the call site possesses. Currently, the prototype is not integrated with an SMT solver; it does not handle arithmetic operations, only (in)equalities.

The following figures showcase our prototype implementation. We chose a number of interesting verification scenarios and describe the resulting behavior of the application.

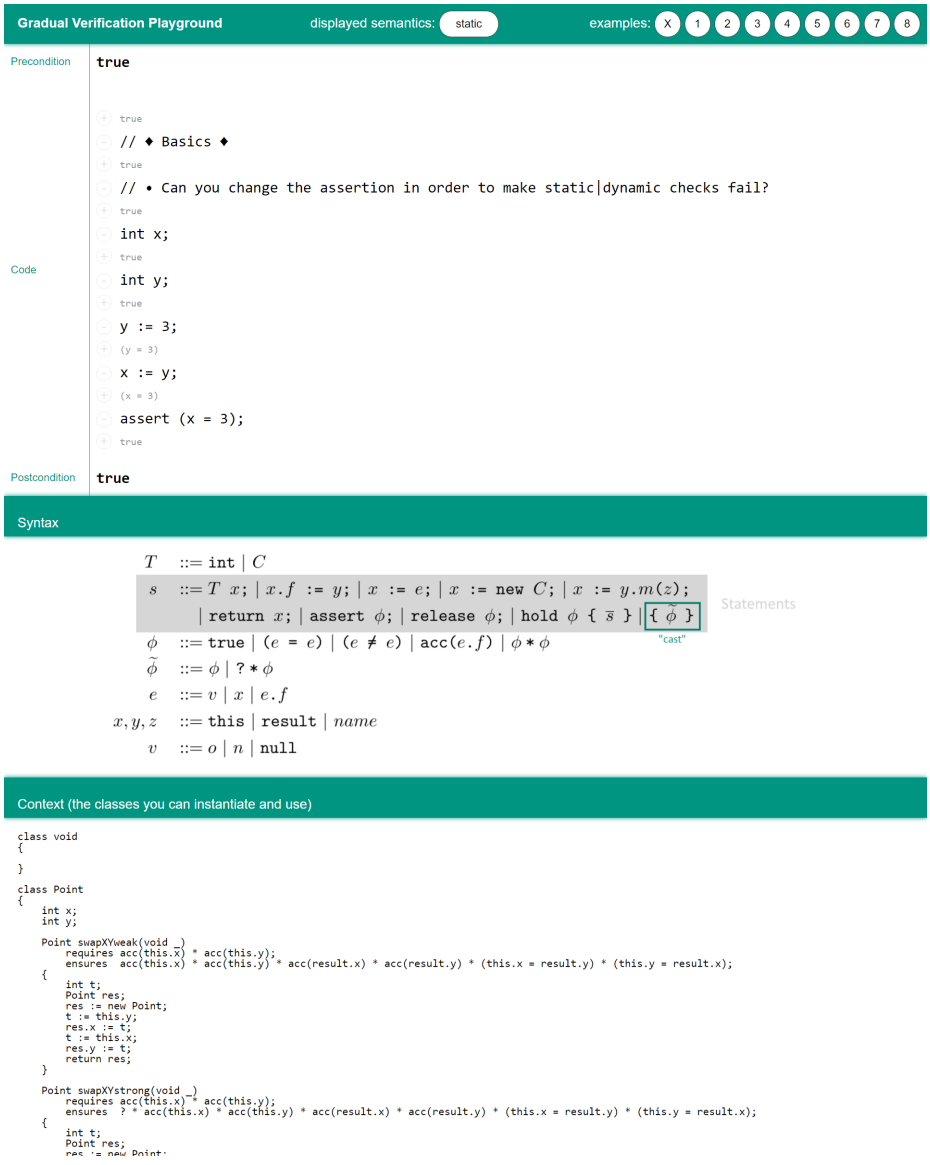

Fig. 1. Overall layout. The editing area is at the top, the instructions are interleaved by the $\widetilde{\mathsf{WLP}}$'s calculated during verification. Necessary runtime checks will also be shown (to be seen in later examples; this example requires no runtime checks). Further down we show the supported syntax, as well as the program context in which the edited statements are interpreted in (*i.e.* usable classes and procedures).

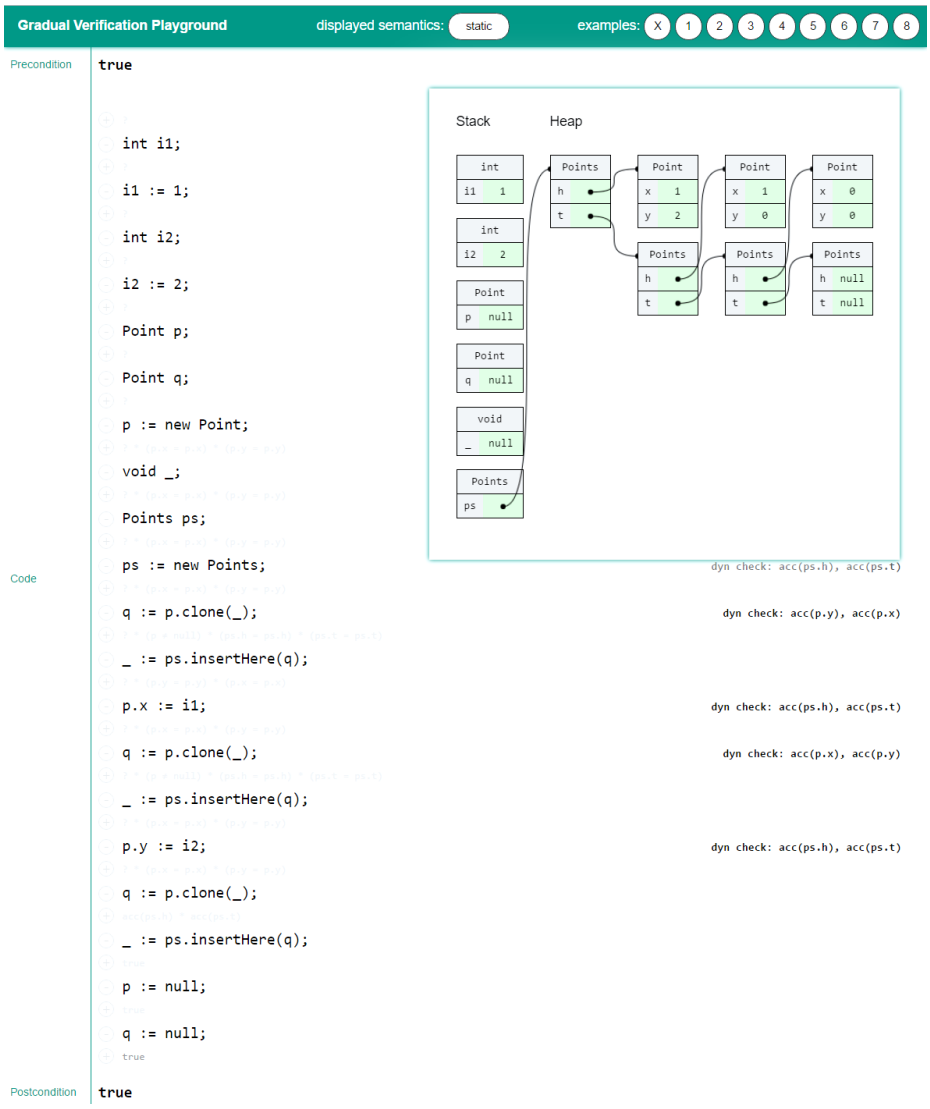

Fig. 2. Another more complex example. Hovering over the space before/after statements will visualize the memory that corresponds to that very spot during program execution (in this example, the very end of execution). Here, one can also see runtime checks being annotated. In this very example, they are necessary since the procedures that are called have very imprecise contracts. These contracts must be imprecise due to syntax limitations, which is one of our motivations of gradualization.

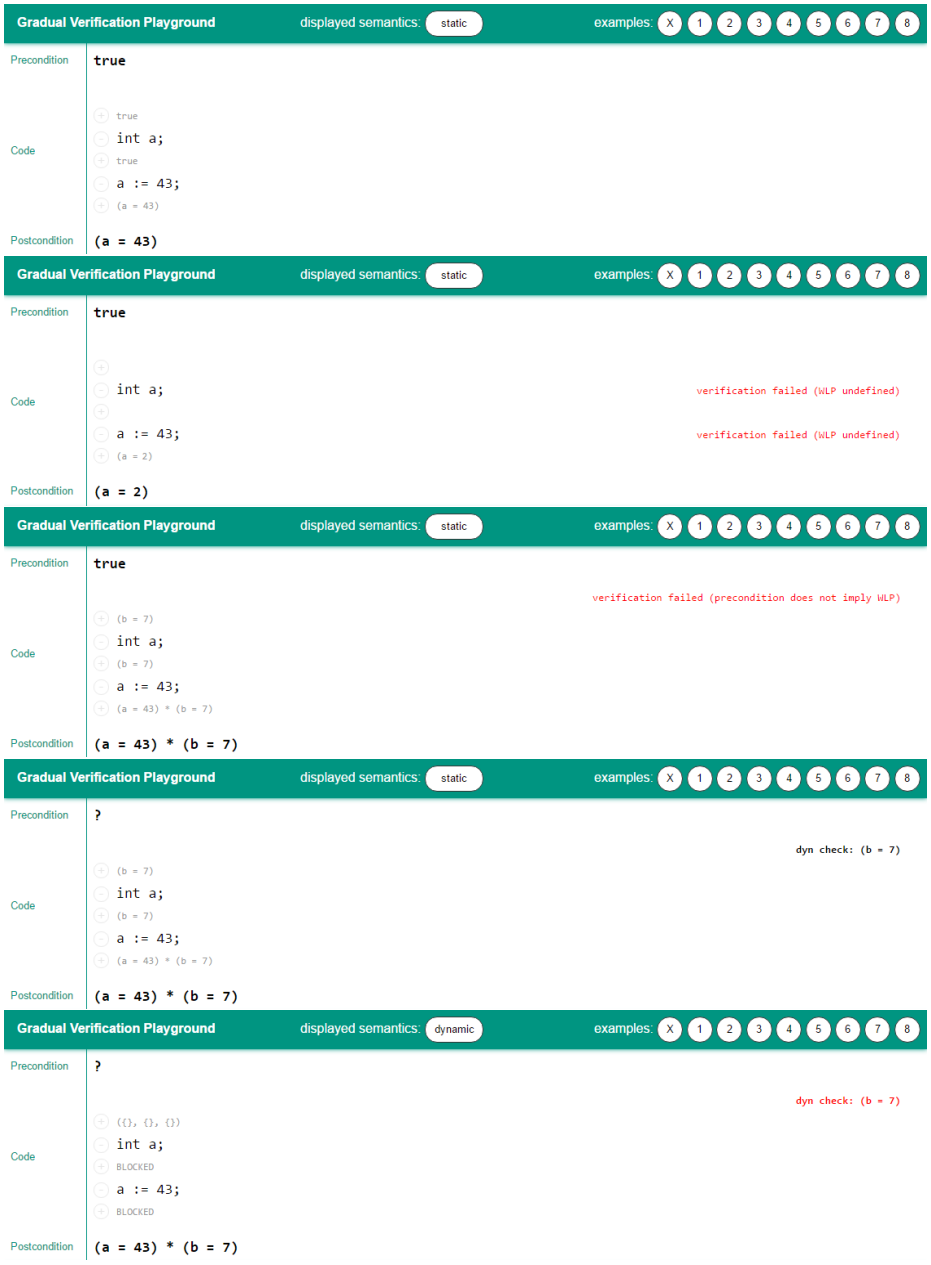

Fig. 3. Semantics overview. 1) A simple, precise, valid program. 2) An invalid program (see postcondition) that is statically rejected due to undefinedness of $\widetilde{\mathsf{WLP}}$. 3) An invalid program that is statically rejected since the procedure cannot be proven correct. 4) Gradualization of the precondition. Verification no longer fails statically, a runtime check is inserted. 5) However, verification fails dynamically for the same program. Note that we have toggled the "displayed semantics" to "dynamic".

A.2 Full Definitions

$$\frac{}{\langle \rho, \texttt{skip; } s\rangle \cdot S \longrightarrow \langle \rho, s\rangle \cdot S}\ \textsc{SsSeqSkip}$$

$$\frac{\langle \rho, s_1\rangle \cdot S \longrightarrow \langle \rho', s_1'\rangle \cdot S}{\langle \rho, s_1\texttt{; } s_2\rangle \cdot S \longrightarrow \langle \rho', s_1'\texttt{; } s_2\rangle \cdot S}\ \textsc{SsSeq}$$

$$\frac{\rho \vDash \phi}{\langle \rho, \texttt{assert } \phi\rangle \cdot S \longrightarrow \langle \rho, \texttt{skip}\rangle \cdot S}\ \textsc{SsAssert}$$

$$\frac{\rho \vdash e \Downarrow v \qquad \rho' = \rho[x \mapsto v]}{\langle \rho, x \texttt{ := } e\rangle \cdot S \longrightarrow \langle \rho', \texttt{skip}\rangle \cdot S}\ \textsc{SsAssign}$$

$$\frac{}{\langle \rho, T\ x\rangle \cdot S \longrightarrow \langle \rho[x \mapsto \mathsf{defaultValue(T)}], \texttt{skip}\rangle \cdot S}\ \textsc{SsDeclare}$$

$$\frac{\begin{array}{c}\mathsf{procedure}(m) = T_r\ m\texttt{(}T_1\ x_1'\texttt{, }\cdots\texttt{,}T_n\ x_n'\texttt{) requires } \phi_p \texttt{ ensures } \phi_q \texttt{ \{ } r \texttt{ \}} \\ \rho \vdash x_1 \Downarrow v_1 \qquad \cdots \qquad \rho \vdash x_n \Downarrow v_n \\ \rho' = [x_1' \mapsto v_1, \texttt{old(}x_1'\texttt{)} \mapsto v_1, \cdots, x_n' \mapsto v_n, \texttt{old(}x_n'\texttt{)} \mapsto v_n] \\ \rho' \vDash \phi_p\end{array}}{\langle \rho, y \texttt{ := } m\texttt{(}x_1\texttt{, }\cdots\texttt{,}x_n\texttt{); } s\rangle \cdot S \longrightarrow \langle \rho', r\rangle \cdot \langle \rho, y \texttt{ := } m\texttt{(}x_1\texttt{, }\cdots\texttt{,}x_n\texttt{); } s\rangle \cdot S}\ \textsc{SsCall}$$

$$\frac{\mathsf{post}(m) = \phi_q \qquad \rho' \vDash \phi_q}{\langle \rho', \texttt{skip}\rangle \cdot \langle \rho, y \texttt{ := } m\texttt{(}x_1\texttt{, }\cdots\texttt{,}x_n\texttt{); } s\rangle \cdot S \longrightarrow \langle \rho[y \mapsto \rho'\texttt{(result)}], s\rangle \cdot S}\ \textsc{SsCallExit}$$

Fig. 4. SVL: Small-step semantics

$$
\begin{aligned}
&\mathsf{WLP}(\texttt{skip}, \phi) && = \phi \\
&\mathsf{WLP}(s_1\texttt{ ; }\ s_2, \phi) && = \mathsf{WLP}(s_1, \mathsf{WLP}(s_2, \phi)) \\
&\mathsf{WLP}(x\ \texttt{:=}\ e, \phi) && = \phi[e/x] \\
&\mathsf{WLP}(\texttt{assert}\ \ \phi_a, \phi) && = \phi_a\ \wedge\ \phi \\
&\mathsf{WLP}(T\ \ x, \phi) && = \phi[\mathsf{defaultValue}(\mathsf{T})/x] \\
&\mathsf{WLP}(y\ \texttt{:=}\ m\texttt{(}\overline{x}\texttt{)}, \phi) && = \max_{\Rightarrow}\ \{\ \phi'\ \mid\ y \notin \mathsf{FV}(\phi')\ \wedge \\
& && \qquad \phi' \Rightarrow \mathsf{mpre}(m)[\overline{x}/\overline{\mathsf{mparam}(m)}]\ \wedge \\
& && \qquad \phi'\ \wedge\ \mathsf{mpost}(m)[\overline{x}, y/\overline{\texttt{old(}\mathsf{mparam}(m)\texttt{)}}, \texttt{result}] \Rightarrow \phi\ \}
\end{aligned}
$$

Fig. 5. SVL: Weakest precondition

$$
\begin{aligned}
&\widetilde{\mathsf{WLP}}(\texttt{skip}, \widetilde{\phi}) && = \widetilde{\phi} \\
&\widetilde{\mathsf{WLP}}(s_1\texttt{ ; }\ s_2, \widetilde{\phi}) && = \widetilde{\mathsf{WLP}}(s_1, \widetilde{\mathsf{WLP}}(s_2, \widetilde{\phi})) \\
&\widetilde{\mathsf{WLP}}(x\ \texttt{:=}\ e, \widetilde{\phi}) && = \widetilde{\phi}[e/x] \\
&\widetilde{\mathsf{WLP}}(\texttt{assert}\ \ \phi_a, \widetilde{\phi}) && = \phi_a\ \wedge\ \widetilde{\phi} \\
&\widetilde{\mathsf{WLP}}(T\ \ x, \widetilde{\phi}) && = \widetilde{\phi}[\mathsf{defaultValue}(\mathsf{T})/x] \\
&\widetilde{\mathsf{WLP}}(y\ \texttt{:=}\ m\texttt{(}\overline{x}\texttt{)}, \widetilde{\phi}) && = \begin{cases} \phi' & \text{if } \widetilde{\phi}, \mathsf{mpre}(m), \mathsf{mpost}(m) \in \textsc{Formula} \\ \phi'\ \wedge\ \texttt{?} & \text{otherwise} \end{cases} \\
& \text{where } \phi' = && \max_{\Rightarrow}\ \{\ \phi'\ \mid y \notin \mathsf{FV}(\phi')\ \wedge \\
& && \qquad \phi' \mathrel{\widetilde{\Rightarrow}} \mathsf{mpre}(m)[\overline{x}/\overline{\mathsf{mparam}(m)}]\ \wedge \\
& && \qquad \phi'\ \wedge\ \mathsf{mpost}(m)[\overline{x}, y/\overline{\texttt{old(}\mathsf{mparam}(m)\texttt{)}}, \texttt{result}] \mathrel{\widetilde{\Rightarrow}} \widetilde{\phi}\ \}
\end{aligned}
$$

Fig. 6. GVL: Weakest precondition