An explanation of forwards/backwards simulation
Maarten Fokkinga, Version of October 24, 2005

Forwards and backwards simulation are techniques to prove that a concrete program satisfies the specification that has been formulated in terms of an abstract program. There are cases where forwards simulation can be applied and not backwards simulation; the converse holds as well.

All this is probably folklore, but I’ve never seen so clear a presentation as the one that I now present myself...

Simulations. We consider two sets, I and F (initial and final) and we are interested in programs that go from I to F. The programs will be specified “at the abstract level” by means of a set A, an initialisation ai : I → A, various operations a, a′, ... : A → A, and a finalization af : A → F. Actually the programs are implemented “at a concrete level”, working on a set C, and they are composed of an initialization ci : I → C, various operations c, c′, ... : C → C (where c corresponds to a, and c′ to a′, etc), and a finalization cf : C → F:

In view of the abstract level as the specification, and the concrete level as an implementation, we require that each input-output pair of the concrete program can also be yielded by the abstract one:

\[
\text{ci} \circ c \circ c′ \circ \cdots \circ c′′ \circ cf \subseteq \text{ai} \circ a \circ a′ \circ \cdots \circ a′′ \circ af
\]

This inclusion is phrased “the abstract program simulates the concrete one”.

Proving simulation. We wish to have a way to prove simulation by “step-by-step simulation”. In order to be able to do so, we must be able to relate intermediate results at the abstract and concrete level to each other. To this end we consider a relation R : A → C:

We expect that if all triangles and squares commute, then so do the entire \(a\) and \(c\) paths. There are however two obvious ways in which the squares can commute (and the triangles analogously):
Theorem. If $R$ is a forwards or backwards simulation, then the abstract program simulates the concrete one.

Proof.

In case $R$ is a forwards simulation:

\[
\begin{align*}
\text{ci} \subseteq ai \downarrow R & \quad R \downarrow \subseteq a \downarrow R & \quad R \downarrow cf \subseteq af \\
ai \downarrow R & \subseteq ai \\
ai \downarrow R & \subseteq ai \\
ai \downarrow R & \subseteq ai \\
\vdots & \\
ai \downarrow R & \subseteq ai \\
ai \downarrow R & \subseteq ai \\
\text{cf} & \subseteq \text{af}
\end{align*}
\]

In case $R$ is a backwards simulation:

\[
\begin{align*}
\text{ci} \downarrow R & \subseteq ai \downarrow \downarrow R \quad R \downarrow \subseteq \downarrow ai \downarrow R \quad R \downarrow \subseteq af \\
ai \downarrow R & \subseteq ai \\
ai \downarrow R & \subseteq ai \\
ai \downarrow R & \subseteq ai \\
\vdots & \\
ai \downarrow R & \subseteq ai \\
ai \downarrow R & \subseteq ai \\
\text{cf} & \subseteq \text{af}
\end{align*}
\]

Notice how in the proof in the forwards case, a longer and longer initial part of the concrete program (growing to the tail) is replaced by the “simulating” program. In the backwards case, a longer and longer final part (growing backwards to the front) is replaced by the simulating program. This explains the naming forwards and backwards. It could already have been seen in the explanatory diagram for the definitions of forwards and backwards simulation: the thick $R$ arrow indicates the position up to which commutativity should already hold, and the dotted $R$ arrow then indicates the position to which commutativity then extends. In the forwards case, the position moves forwards, whereas in the backwards case the position moves backwards.

Examples. The funny thing is: sometimes there exists a forwards and no backwards simulation, sometimes the converse holds.

Let $One = \{ \ddagger \}$ be a one-point set, whose only element is denoted $\ddagger$. We define two operations that are each others dual:
\[ \text{force} = \{ n : N \bullet n \mapsto \dagger \} \quad \text{non-injectivity is essential} \]
\[ \text{choose} = \{ n : N \bullet \dagger \mapsto n \} \quad \text{non-functionality is essential} \]

Now consider the following two cases (diagram F at the left, diagram B at the right):

\[ \text{force} \uparrow \text{id} \downarrow \text{id} \leq \text{id} \downarrow \text{id} \downarrow \text{force} \quad \text{id} \downarrow \text{id} \downarrow \text{choose} \leq \text{choose} \downarrow \text{id} \downarrow \text{id} \]

In both cases the abstract program equals the concrete one, so it certainly simulates the concrete one. The simulation in the F case can be proved by forwards simulation and not by backwards simulation; whereas the converse holds for the B case:

**Case F:** Taking \( R = N \times \text{One} \), the conditions for forwards simulation are readily verified:
\[ \text{force} \leq \text{id} \downarrow R, \quad R \downarrow \text{id} \leq \text{id} \downarrow R, \quad R \downarrow \text{id} \leq \text{force}. \]

Regarding backwards simulation, the condition ‘\( \text{id} \downarrow R \subseteq \text{choose} \uparrow \text{id} \)’ will be false for all \( R \neq \emptyset \):
\[ \text{force} \uparrow R \not\leq \text{id}. \]

**Case B:** Taking \( R = N \times \text{One} \), the conditions for backwards simulation are readily verified:
\[ \text{id} \downarrow R \subseteq \text{choose}, \quad \text{id} \downarrow R \subseteq R \downarrow \text{id}, \quad \text{choose} \subseteq R \downarrow \text{id}. \]

Regarding forwards simulation, the condition ‘\( R \downarrow \text{cf} \subseteq \text{af} \)’ will be false for all \( R \neq \emptyset \):
\[ R \downarrow \text{choose} \not\subseteq \text{id}. \]

In less formal and more intuitive terms, the example reads “initialize \( \uparrow \) compute \( \downarrow \) finalize” where:

<table>
<thead>
<tr>
<th></th>
<th>operation name</th>
<th>specification</th>
<th>implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>F:</td>
<td>initialize</td>
<td>do nothing</td>
<td>print “accepted!”</td>
</tr>
<tr>
<td></td>
<td>compute</td>
<td>do nothing</td>
<td>do nothing</td>
</tr>
<tr>
<td></td>
<td>finalize</td>
<td>print “accepted!”</td>
<td>do nothing</td>
</tr>
<tr>
<td>B:</td>
<td>operation name</td>
<td>specification</td>
<td>implementation</td>
</tr>
<tr>
<td></td>
<td>initialize</td>
<td>pick a random number</td>
<td>do nothing</td>
</tr>
<tr>
<td></td>
<td>compute</td>
<td>do nothing</td>
<td>do nothing</td>
</tr>
<tr>
<td></td>
<td>finalize</td>
<td>print that number</td>
<td>print a random number</td>
</tr>
</tbody>
</table>