source file. At this stage, the Java class files contain all the information that will allow the client to check if the bytecode does not violate his requirements. In particular, the client will generate proof obligations from the untrusted annotated bytecode

internal array `list` **contains the object referenced by the argument** `obj` **in its postcondition(**`ensures`**). The method loop is also speci ed by its invariant (**`loop_invariant`**) which basically says that whenever the loop entry is reached the elements inspected already by the loop are all di erent from** `obj`**.**

```
public class ListArray {
  Object[] list;
  //@requires list != null;
  //@ensures -Tj 6st ==
```

the instruction at which the loop invariant must hold (the loop entry instruction). this is di erent from JML where loop invariants are written at the beginning of the declaration of the loop statement, while the BCSL speci cation are separated from the bytecode

predicates from rst order logic

expressions from the programming language, like eld

1. compile the Java source le. This can be done by any Javasource

_Va _Ta attributes. The
presence in the Javasource
format of these attributes
[7], ya almost all standard non optimizing
ata. The Line _Number_Ta describes the link between the
source line and the bytecode of a

ants, assertions at particular program point among which loop invariants (if there is no explicite speci cation

function update when assigning a value to a eld reference as, for instance in [3]. In Fig. 5 the rule for `putField` substitutes the corresponding eld function `Cl.f` with `Cl.f` updated for object o, in case the dereferenced object is not null. The de nition of update function is given in gure 6.

4.0.2 Method calls

Method calls are handled by using their speci cation. A method speci - cation is a contract betw

$$
\begin{aligned}
&\mathbf{wp}(\texttt{invoke}\ m;\ ;\ ^{exc}) = \\
&\quad {}^{pre}(m) \wedge \\
&\quad \forall_{j=1::s} e_j : (\\
&\qquad {}^{post}(m)[\texttt{lv[i]} \leftarrow \texttt{st(c + i - numArgs(m))}]_{i=0}^{numArgs(m)} \\
&\qquad\quad [\mathbf{n}\texttt{result} \leftarrow \texttt{fresh\_var}] \\
&\qquad) \ [\texttt{c} \leftarrow \texttt{c} - \texttt{numArgs}(m)][\texttt{st(c)} \leftarrow \texttt{fresh\_var}]) \wedge
\end{aligned}
$$

compilers perform mod-

ulo the names of the program variables. Another issue that deserves the attention is that even non

generated by the standard feature of JACK and the bytecode proof obligations generated by our implementation over the corresponding bytecode produced by a non compiler. Basically, they are the

Hypothesis on bytecode:	Hypothesis on source level:
lv[2]_at_ins_20 len(#19(lv[0]))	i_at_ins_26 len(ListArray:list(this))
#19(	