J v Byt ■o■ p ■ifi■ tion ıı· V r fi■ t oıı

the certifying compiler for properties like well typedness or safe memory access. As the certifying compiler is designed to be completely automatic, it will not

Bytecode Specification Language

Specification clau e in B SL that are taken from JML and inherit their emantic directly from JML include:

- cla pecification, i.e. cla invariant and hi tory con traint
- method precondition , normal and exceptional po tcondition , method frame condition (the location that may be modified by the method). We al o upport behavioral ubtyping by pecification inheritance (the keyword al o)
- inter method pecification, for in tance loop invariant
- predicate from fir t order logic
- expre ion from the programming language, like field acce expre ion , local variable , etc.
- pecification operator . For in tance `\old(`$\mathcal{E}$`)` which i u ed in method $\mathcal{E}$ in the pre tate
  of a method, `\result`

ub titution lemma .

Returning back to the example, the expre ion `c` and `st(c)` tand re pectively for the the tack counter and the element on the top of the tack. Thi i becau e the JVM i tack ba ed, i.e. the in truction take their argument from the method execution tack and put the re ult on the tack. The *wp* rule for `Type_Load` increment the tack counter `c` and load on the tack top the content of the local variable `lv[i]`.

In the re t of the ection, we con ider the following pecific

re ulting predicate i quantified over the expre ion that may be modified by the called method. We al o a ume that if the invok
abnormally, by throwing an exception of type Exc, on returning the c u trol to the invok*er* hold . The rule for tatic
Exc
rather the ame except for the number of tac

4.4 Exceptions and Subroutines

Exception handlers are treated by identifying the instruction at which the handler compilation starts. The JVM specification mandates that a Java compiler must supply for every method an **Exception Table** attribute that contains data structures describing the compilation of every implicit (in presence of subroutines) or explicit exception handler: the instruction at which the compiled

expre ion a integer (which

Thi validation can be done at ource or at bytecode level in a common environment: for in tance, to prove lemma en uring bytecode correctne all the current and future prover plugged in Jack can be u ed.

We envi age everal direction for future work:

- perfor ca e tudie
- build a P framework where the proof are done interactively over the ource code and then compiled down to bytecode. A we di cu ed in ection 5 in the perfor ed te t the proof obligation generated over a ource program and over it compilation with non optimizing compiler are yntactically the fu e modulo nu e and type . We aim to e tabli h