Să se dezvolte următorii algoritmi pornind de la specificaţii, utilizând cele 4 reguli de rafinare (atribuire, compunere secvenţială, alternanţa, iteraţia):

a. **rădăcină pătrată:**
   - **ϕ(X):** **n > 1**
   - **Ψ(X, Z):** **$r^2 \le n < (r+1)^2$**

b. **împărţire întreagă (cât şi rest):**
   - **ϕ (X):** **(x ≥ 0) ∧ (y > 0)**
   - **Ψ(X, Z):** **(x = q * y + r) ∧ (0 ≤ r < y)**

Exemple de algoritmi pentru examenul scris rezolvate în fişierul Rafinare.pdf:

- **Împărţire întreagă (cu cât şi rest);**
- **Rădăcină pătrată;**

Rafinare [Fre10]

- Regula atribuirii:
  $[\varphi(v/e), \psi] \prec v := e$
- Regula compunerii secvențiale:
  $[\eta_1, \eta_2] \prec [\eta_1, \gamma]$
  $[\gamma, \eta_2]$
  (γ - predicat auxiliar
  (*engl.* **middle predicate**))
- Regula alternanței:
  $cond = c_1 \vee c_2 \vee \ldots \vee c_n$;
  $[\eta_1, \eta_2] \prec$
  if $c_1 \rightarrow [\eta_1 \wedge c_1, \eta_2]$
  $\square$ $c_2 \rightarrow [\eta_1 \wedge c_2, \eta_2]$
  $\vdots$
  $\square$ $c_n \rightarrow [\eta_1 \wedge c_n, \eta_2]$
  fi
- Regula iterației:
  $cond = c_1 \vee c_2 \vee \ldots \vee c_n$
  $[\eta, \eta \wedge \neg cond] \prec$
  do $c_1 \rightarrow [\eta \wedge c_1, \eta \wedge TC]$
  $\square$ $c_2 \rightarrow [\eta \wedge c_2, \eta \wedge TC]$
  $\vdots$
  $\square$ $c_n \rightarrow [\eta \wedge c_n, \eta \wedge TC]$
  od

jmlc / jmlrac and Esc2Java
JML Compiler and Runtime Assertion Checker

default (no specification checking)

javac & java usage

- class `Account.java`
  - compile:
    - `javac Account.java`
    - output: a bytecode file `Account.class`
    - ignores any comments, i.e., JML specification
  - run with the standard VM:
    - `java Account`
    - possible specification inconsistencies not highlighted

specification checking

jmlc & jmlrac usage

- class `Account.java`
- compile:
  - `jmlc Account.java` or `jmlc -Q Account.java`
  - **output:** a bytecode file `Account.class` that enables automatic checks of JML assertions at the run time
  - **`jmlc` acts as a preprocessor for `javac`**
- run the **JML** run-time assertion checker:
  - `jmlrac Account`
  - possible specification inconsistencies identified and `Errors` are thrown
  - `jmlrac` script enables the automatic use of JML runtime classes (`jmlruntime.jar`) from the Java boot class path, required to run the checks on the assertions
  - **`jmlrac` acts as a wrapper for the standard VM**

jmlc / jmlrac and Esc2Java
Runtime Assertion Checker and Static Checker

compile-time checking

Esc2Java

- **checks specifications at compile-time**
- proves the specification correctness
- warns about likely runtime exceptions and violations.
- automatically tries to prove simple JML assertions at compile time

run-time checking

jmlc & jmlrac

- **checks specifications at run-time**
- tests the specification correctness only
- finds the specification violations at runtime
- **jmlc** = special compiler that inserts runtime tests for all JML assertions;
- **jmlrac** - any assertion violation results in a special exception at run-time.