```
abstract sig Program {
 required: some Course
}

one sig CSE, SWE extends Program {}

sig Course {
 enrolled: some Student,
 prerequisite: set Course
}

sig Student {
 id: one ID,
 batch: one Batch,
 program: one Program,
 transcript: set Course
}

sig RecordBook {
 students: set Student
}

sig ID, Batch {}

fact {
 all s: Student | let p= s.program | (p in CSE => p not in SWE) and (p in SWE => p not in CSE)

 all s: Student, r: RecordBook | s in r.students => s.program.required in s.transcript

 all disj s1, s2: Student | s1.program != s2.program => s1.transcript != s2.transcript

 all s: Student | s.transcript.^prerequisite in s.transcript

 CSE.required != SWE.required
}

assert a1 {
 // No two distinct students have different ids / Every student has the same id.
 // Found. Justification: It is possible to have different ids for two different students.
 no disj s1, s2: Student | s1.id != s2.id
}
check a1 for 2

assert a2 {
 // There are some courses where two distinct students are come from the same batch but from different programs.
 // Found. Justification: For a specific course, it is possible to have two different student whose batch and program are same.
 some c: Course, disj s1, s2: c.enrolled | s1.batch = s2.batch and s1.program != s2.program
}
check a2 for 3

assert a3 {
 // There are some courses where the students are come from two diffferent programs.
 // Found. Justification: It is possible to have course(s) where the students are come from only one program.
 some c: Course | #c.enrolled.program = 2
}
check a3 for 2

assert a4 {
  // There are some courses which are required for two programs
 // Found. Justification: It is possible to create a scenario where there is no any course which is required for two programs
 some c: Course | c in CSE.required and c in SWE.required
}
check a4 for 2
```