Download **FDR (Failures-Divergence Refinement)** from cocotec.io/fdr, which is a **model checker for CSP (Communicating Sequential Processes)**.

🧩 Step 1: Define your process

In CSP notation, your student process is sequential, doing three events (`getup`, `lecture`, `eat`), and then stopping.

```
Student = getup -> lecture -> eat -> STOP
```

🧩 Step 2: Define an alphabet (optional)

If you want to make the alphabet explicit (not required for this simple example), you could do:

```
alphabet Student = {getup, lecture, eat}
```

This just tells FDR what events the process can perform.

🧩 Step 3: (Optional) Add assertions or refinements to check

If you just want to **run or visualize** the trace of the process, you don't need assertions.

But if you want FDR to **check something**, for example, that the process can perform a certain trace, you can write something like:

```
assert Student :[ trace <= <getup, lecture, eat> ]
```

or more simply:

```
assert Student :[deadlock free]
```

🧩 Step 4: Run it on FDR

1. Open https://cocotec.io/fdr/
2. Paste your CSP code into the editor pane:
3. `Student = getup -> lecture -> eat -> STOP`
4. Click **"Check"** (or the ▶ "Run" icon if it's the web-based IDE).
5. You should see a trace like:
6. `<getup, lecture, eat>`

   and the process halts (`STOP`).

🧠 Notes

- Each `->` means *"then do…"*.
- `STOP` means no more events.
- You can replace `STOP` with another process to make it cyclic or branching (e.g., `-> Student` to loop forever).

⚙️ How FDR Works

FDR doesn't interpret CSP code *line-by-line* like Python;
instead, it loads a **CSP specification file** (ending in `.csp`) all at once.

So when you type:

```
Student = getup -> lecture -> eat -> STOP
```

directly into the **interactive prompt**, it complains because `=` isn't a valid console command.

✅ Here's how to do it correctly:

Option 1: Use the File Editor (recommended)

1. Go to https://cocotec.io/fdr/.
2. Click **File → New CSPM file** (or the "+" button).
3. Paste this code:

```
-- Define the process
Student = getup -> lecture -> eat -> STOP

-- Simple check
assert Student :[deadlock free]
```

9. Save it (e.g. `student.cspm`).
10. Click **Run** or **Check** in the toolbar.
    → It should run without errors and show that `Student` is *deadlock free*.

Option 2: Use the REPL for queries (if you must)

If you insist on using the interactive console, you can type:

```
:load student.cspm
```

…but that requires that you've already saved the file first.

🧠 Quick sanity test

Try this minimal working example in a new file:

```
Student = getup -> lecture -> eat -> STOP

assert Student :[deadlock free]
```

If you run it, FDR should confirm the process has no deadlocks.

FDR needs to know that `getup`, `lecture`, `eat`, and `sleep` are **events** (symbols in the alphabet).
By default, it doesn't know what they are unless you **declare them first**.

✅ Fix

At the top of your file, **declare your events** as names in a set (alphabet).
Here's the **correct minimal working example**:

```
-- Declare the events (the alphabet)
channel getup, lecture, eat, sleep

-- Define the process
Student = getup -> lecture -> eat -> sleep -> STOP

-- Optional: check for deadlock freedom
assert Student :[deadlock free]
```

🧩 Explanation

- `channel getup` declares `getup` as an event name.
  (You can also write `channel a, b, c` for multiple.)
- You can think of these as "actions" the process performs in order.
- Each `->` means "then do".
- `STOP` means "the process ends".

✅ How to run

1. Open your file in **FDR** (e.g. `test.csp`).
2. Make sure it contains exactly the snippet above.

3. Click **Run** or **Check assertions**.
4. You should now see no errors, and `Student` will be marked as *deadlock free*.

⚙ Optional: view the trace

If you want to **see the event sequence** (`<getup, lecture, eat, sleep>`):

Add this at the bottom:

```
-- Run the trace explorer manually
-- In the GUI: right-click on Student → "Explore" → "Traces"
```

It will show the order of events and termination (`STOP`).

What FDR is showing

- You defined:
- `channel getup, lecture, eat, sleep`
- `Student = getup -> lecture -> eat -> sleep -> STOP`
- `assert Student :[deadlock free]`
- FDR shows: **"Deadlock Counterexample"** and the trace
- `getup → lecture → eat → sleep → STOP`

  with the word **Deadlock Counterexample** in red.

💡 Why this happens

FDR's **deadlock-free** check means:

"Can this process always perform *some* event, or terminate successfully, rather than get stuck waiting?"

In plain terms:

- A **deadlock** is a point where the process can't do anything — it's *stuck*.
- In CSP, `STOP` is exactly such a point: the process halts and has **no further events**.

So your process ends with `STOP` → that's a *deliberate termination*,
but from the checker's point of view, it's a **deadlock** because no further communication is possible.

Hence, FDR reports a *Deadlock Counterexample*, even though your sequence is perfectly fine.

✅ How to fix it (if you want it to pass the "deadlock-free" check)

1. Replace the final `STOP` with a process that can continue.
   For example, looping back:

```
channel getup, lecture, eat, sleep

Student = getup -> lecture -> eat -> sleep -> Student
```

   Now `Student` repeats forever — it never reaches a state where no action is possible.

5. If you rerun

```
assert Student :[deadlock free]
```

   you'll see **no counterexample**.

🧩 What you've learned

Concept	Meaning
`STOP`	Process terminates, can do nothing more (deadlock from FDR's view).
`->`	Sequential composition ("then").
`P = a -> Q`	After event `a`, behave like `Q`.
Loop (`P = ... -> P`)	Process never halts → passes deadlock-free check.

---

1. Selected Behaviour

This area tells you which **process or property** is currently being explored or checked.

- **Student** → the process whose graph (state machine) you're viewing.
- **Probe / Graph / Structure**
  - **Probe** – opens the interactive probe window, where you can single-step through events.
  - **Graph** – shows the labelled-transition-system (LTS) diagram, like the one in the main window.
  - **Structure** – shows the syntactic structure of the CSP definition (tree view).

When you check an assertion (like *deadlock-free*), FDR creates an internal model of the process; this section lets you inspect it.

---

2. Trace to Behaviour

Appears only when FDR has found a counterexample (as in your screenshot).

- Displays the **sequence of events** leading to the failure, e.g.
- `getup → lecture → eat → sleep`
- Shows how far the tool progressed before the violation occurred.

---

3. Selected Node

This block shows data about the **currently highlighted state** in the LTS (the yellow box).

- **Unnamed** – the node label (if you haven't named it).
- **Graph / Inspect / Probe**
  - **Graph** – open the LTS for just this sub-process (lets you zoom into that state).
  - **Inspect** – shows the textual CSP expression corresponding to this node.
  - **Probe** – launches the interactive probe starting from this node.
- **Available Events** – list of events that can occur next from this state.
- **Minimal Acceptances** – the minimal sets of events the process can accept from this point (used for refinement checking).

---

4. Event Set Mode

A small area near the bottom with toggles:

Control	Meaning
Acceptances ○ Refusals	Choose whether you're viewing *acceptances* (what the process can do) or *refusals* (what it can refuse) for the selected node.
Hide Inactive Components	When you're checking systems of several processes, this hides the ones that are currently idle.
View Taus	Show or hide internal (τ) events—those that represent hidden communications.

Control	Meaning
Expand All / Contract All	Expands or collapses the lists of events and acceptances/refusals.

The **slider** just below that controls how many acceptances/refusals are displayed—handy when there are many possibilities.

🔍 Summary

The right-hand panel is basically your **inspection and debugging control center**:

- *Top* → which process you're exploring.
- *Middle* → what state (node) you're looking at and what it can do next.
- *Bottom* → how event sets and internal transitions are visualized.

---

1. Context

At any point in a CSP process, FDR considers what the process **can** do next and what it **may refuse** to do.
Both are crucial to check whether two processes behave the same (refinement) or if one is deadlocked.

---

2. Acceptances

- An **acceptance set** lists the **events the process is willing to engage in** from its current state.

Example:

```
Student = getup -> lecture -> eat -> sleep -> STOP
```

After doing `getup -> lecture`,
the process's **acceptances** set is `{eat}`
because the only possible next event is `eat`.

So, if FDR shows:

```
Minimal Acceptances:
{ eat }
```

it means the process is ready for `eat` next — that's its *offer* to the environment.

3. Refusals

- A **refusal set** lists **events the process is not willing to perform** from that state.
  In other words, events the environment could try to offer but would be ignored.

At the same state (`after getup → lecture`):

If your alphabet is `{getup, lecture, eat, sleep}`,
then the process *refuses* `{getup, lecture, sleep}`
because none of those can occur right now.

So:

```
Refusals:
{ getup, lecture, sleep }
```

---

4. Why It Matters

- **Deadlock** occurs when a process refuses **every event** in its alphabet — i.e. there's *nothing it can do next*.
  (`STOP` has a refusal set equal to the whole alphabet.)
- **Divergence** (not visible in your screenshot) occurs when a process gets stuck in internal `τ` actions forever.
- **Refinement checking** uses acceptances/refusals to decide if one process always behaves like or more deterministically than another.

---

5. In FDR's UI

When you toggle between:

- **Acceptances** → you see *what events are possible next*.
- **Refusals** → you see *what events are impossible next*.

The little `{}` on the last node of your diagram (after `sleep`) means the process has *no acceptances* — it's at `STOP`.
That's why FDR reported a **Deadlock Counterexample**: the assertion
`assert Student :[deadlock free]`
fails, because the process *ends* and refuses everything.

✅ **Quick Fix:**
If you wanted `Student` to be "deadlock free," make it cyclic:

```
channel getup, lecture, eat, sleep

Student = getup -> lecture -> eat -> sleep -> Student
```

Now it never reaches a `STOP` state — there's always another event possible — and FDR will show:

```
Result: Satisfied
```
