CCS Modelling: Toy Examples

Khushraj

February 22, 2026

CCS Recap — Syntax

- **Actions:** a, $\overline{a}$ (co-action)
- **Prefix:** $a.P$ — do a, then behave like P
- **Choice:** $P + Q$ — nondeterministic choice
- **Parallel:** $P \mid\mid Q$ — concurrent composition
- **Restriction:** $P \setminus a$ — hide a
- **Relabeling:** $P[f]$ — rename actions using f

Motivating Example

- A user interacts with a vending machine
- The machine accepts a coin
- The user selects a product
- The machine delivers the item

Goal: Model the control flow using CCS.

Actions and Intuition

Visible actions

$$\textit{coin}, \ \textit{select}, \ \textit{item}$$

- `coin`: user inserts a coin
- `select`: user selects a product
- `item`: machine dispenses the product

We do not model prices or product types.

Vending Machine Process

Machine behaviour:

$$VM ::= coin.select.\overline{item}.VM$$

- Machine waits for a coin
- Then waits for a selection
- Then delivers the item
- Returns to idle state

User Process

User behaviour:

$$User ::= \overline{coin}.\overline{select}.item.User$$

- Inserts a coin
- Makes a selection
- Receives the item
- Repeats

Complete System

$$System = (VM \mid User) \setminus \{coin,\ select,\ item\}$$

Effect of restriction:

- All interactions are synchronized
- System becomes closed
- Only internal (τ) actions remain

Example Execution

$\tau \; \tau \; \tau$

- Coin insertion synchronizes
- Selection synchronizes
- Item delivery synchronizes

This sequence repeats indefinitely.

Correctness Properties

- No item without coin
- No item without selection
- No deadlock
- Deterministic machine behaviour

All properties follow directly from the CCS structure.

Possible Extensions

- Cancel action
- Multiple coins
- Choice of products
- Faulty machine (no item)

Each extension corresponds to a small CCS change.

Yet Another Motivating Example

- Two ATMs operate concurrently
- Both access a single shared database
- Database must serialize transactions
- Cash must be dispensed only after commit

Goal: Model transaction control using CCS, without data values.

Actions and Intuition

Visible actions

$$req_1,\ req_2,\ cash_1,\ cash_2$$

Internal (synchronizing) actions

$$debit,\ commit,\ abort$$

- Subscripts distinguish the two ATMs
- Database interactions will be hidden

Database Process

Idea: The database handles one transaction at a time.

$$DB ::= debit.DB_{busy}$$

$$DB_{busy} ::= commit.DB + abort.DB$$

- While busy, no new debit is accepted
- This enforces transaction serialization

ATM_1 Process

$$ATM_1 ::= req_1.\overline{debit}.(commit.\overline{cash_1}.ATM_1 + abort.ATM_1)$$

- User initiates request
- ATM asks DB to debit
- Cash is dispensed only after commit

ATM_2 Process

$$ATM_2 ::= req_2.\overline{debit}.(commit.\overline{cash_2}.ATM_2 + abort.ATM_2)$$

- Same structure as ATM_1
- Independent user interactions

Complete System

$$System = (ATM_1 \mid ATM_2 \mid DB) \setminus \{debit,\ commit,\ abort\}$$

Visible behaviour:

$$req_1,\ req_2,\ cash_1,\ cash_2$$

All transaction control becomes internal (τ).

Example Executions

Successful ATM$_1$ transaction

$$req_1 \ \tau \ \tau \ cash_1$$

Interleaving of requests

$$req_1 \ req_2 \ \tau \ \tau \ cash_1 \ \tau \ \tau \ cash_2$$

Requests may overlap, but transactions do not.

Atomicity and Safety Properties

- Cash is dispensed iff commit occurs
- No two debits occur simultaneously
- Abort never leads to cash

Atomicity:

Each transaction appears indivisible to the user.

Why This Example is Useful

- Realistic and intuitive
- Pure CCS (no data, no values)
- Demonstrates synchronization and serialization
- Bridges concurrency theory and databases

Possible Extensions

- Retry on abort
- Explicit locking protocol
- Buggy database allowing two debits
- Bisimulation with queue-based DB

Summary

- CCS models control, not data
- Transaction correctness emerges from synchronization
- Concurrency theory can explain real systems.

Motivating Example

- A user interacts with a vending machine
- The machine accepts a coin
- The user selects a product
- The machine delivers the item

Goal: Model the control flow using CCS.

Actions and Intuition

Visible actions

$$coin, \ select, \ item$$

- `coin`: user inserts a coin
- `select`: user selects a product
- `item`: machine dispenses the product

We do not model prices or product types.

Vending Machine Process

Machine behaviour:

$$VM ::= coin.select.\overline{item}.VM$$

- Machine waits for a coin
- Then waits for a selection
- Then delivers the item
- Returns to idle state

User Process

User behaviour:

$$User ::= \overline{coin}.\overline{select}.item.User$$

- Inserts a coin
- Makes a selection
- Receives the item
- Repeats

Complete System

$$System = (VM \mid User) \setminus \{coin,\ select,\ item\}$$

Effect of restriction:

- All interactions are synchronized
- System becomes closed
- Only internal (τ) actions remain

Example Execution

$\tau \ \tau \ \tau$

- Coin insertion synchronizes
- Selection synchronizes
- Item delivery synchronizes

This sequence repeats indefinitely.

Correctness Properties

- No item without coin
- No item without selection
- No deadlock
- Deterministic machine behaviour

All properties follow directly from the CCS structure.

Why This Example is Useful

- Small but expressive
- Demonstrates sequencing
- Shows synchronization clearly
- Ideal first CCS example

Possible Extensions

- Cancel action
- Multiple coins
- Choice of products
- Faulty machine (no item)

Each extension corresponds to a small CCS change.

Summary

- CCS models reactive control
- Synchronization enforces protocol order
- Vending machines are simple protocols