ASP Case Study:

Stock Portfolio Management

**Full name:** Chi Nguyen Chau

**Course:** AIWSU 7021 – Knowledge Representation and Reasoning

# **Scenario – English Description**

We study a **retail stock portfolio** managed by a simple **trading robot** over discrete time steps 0..n. The portfolio contains **cash** and shares of at least **ten stocks**.

**Goal:**

The robot follows a **predefined trading plan** consisting of **target prices and target quantities** for each stock (optionally only buy, only sell, or both). The robot **does not plan/optimize**; it only **executes** orders when the market price satisfies the target condition with the **exact target quantity**.

* **Buy rule:** The robot may **buy** exactly shares of stock **only if** the current price satisfies and sufficient cash is available.
* **Sell rule with shorting:** The robot may **sell** exactly shares of stock **only if** . If the target sell quantity exceeds the currently held shares, the **excess is executed as a short sale** (i.e., holdings can become negative).
* **Cash updates (deposits/withdraws):** The robot may execute a **deposit/withdraw** action that increases/decreases cash by an amount . Deposits/Withdraws are inertial updates to cash and take effect in the **next** time step (consistent with transition semantics). Deposits can be used to ensure later buy targets are fundable.
* Some stocks may have **targets set but no execution yet** (e.g., a buy target exists but price hasn’t dropped enough; or a sell target exists while holdings are zero), and that is expected in the fixed-plan regime. Targets persist until changed (in our model, targets will only be set at the beginning). If buy/sell orders are executed at target prices, target quantities will be set to 0 which implies that targets are no longer effective.

Additional simplifications:

* A stock cannot be bought and sold in the same step.
* Prices change exogenously and are inertial otherwise.
* Holdings (which may be negative due to shorting), targets, cash, and current prices are inertial.
* Orders are always executed successfully.

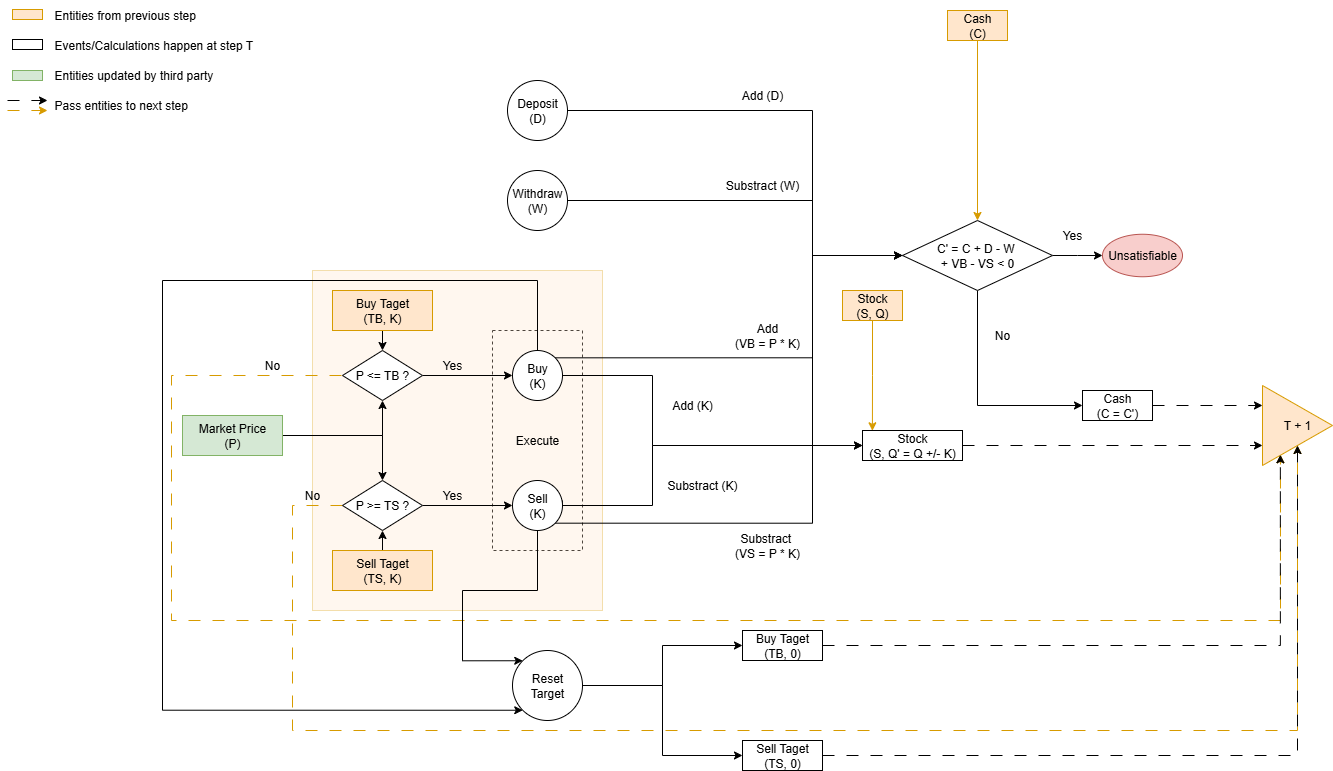
****

Figure : Trading Robot Execution

# Formal Domain Signature (AL-style)

**Domain**

* *(≥ 10 objects)*

**Inertial Fluents**

* where S:stock, Q:pos (can be negative due to shorting).
* where
* where S:stock, P:price.
* where S:stock, TB:target price, K:quantity.
* where S:stock, TS:target price, K:quantity.

**Actions**

* Robot:
  + with S:stock, K:qlot.
  + with S:stock, K:qlot.
  + with D:cashlot.
  + with W:withlot.
* Exogenous: with S:stock, P:price.

# System Description (SD) in AL

**Dynamic Causal Laws**

1. **Trading effects**
   * causes .
   * causes .
   * causes .
   * causes .
   * causes .
   * causes .
2. **Cash management**
   * causes .
   * causes .
3. **Market move**
   * causes .

**State Constraints**

* For each stock S, exactly one holds.
* For each stock S, exactly one holds.
* For each stock S, maximum one holds.
* For each stock S, maximum one holds.
* Exactly one holds with .

**Executability Conditions (fixed plan + shorting + cash bounds)**

* **Target compliance & quantities:**
  + .
  + .
  + .
  + .
  + .
  + .
* **Budget & position limits:**
  + (budget check at current step).
  + .
* **Cash constraint:**  
  Let be the **next** cash after applying all actions in the step.
  + .

**Inertia Postulates**

* , , , and are inertial.

**Initial State (example)**

* .
* .
* Prices at :  
  .
* Targets **with quantities** (e.g., , ) are specified **per scenario** to demonstrate pending targets and short execution.

# ASP Program (Translation of SD)

Increase #const n to extend the horizon.

% Domain

#const n = 2.

step(0..n).

stock(aapl;msft;goog;tsla;amzn;nvda;meta;baba;jpm;xom).

% Fluents

:- val\_shares(S,Q1,T), val\_shares(S,Q2,T), Q1 != Q2.

:- val\_price(S,P1,T), val\_price(S,P2,T), P1 != P2.

:- val\_cash(C1,T), val\_cash(C2,T), C1 != C2.

:- target\_buy(S,TB1,K1,T), target\_buy(S,TB2,K2,T), TB1 != TB2.

:- target\_sell(S,TS1,K1,T), target\_sell(S,TS2,K2,T), TS1 != TS2.

% Actions

occurs(buy(S,K),T) :- target\_buy(S,TB,K,T), val\_price(S,P,T), P<=TB, K>0.

occurs(sell(S,K),T) :- target\_sell(S,TS,K,T), val\_price(S,P,T), P>=TS, K>0.

-occurs(buy(S,K),T) :- target\_buy(S,TB,K,T), val\_price(S,P,T), P>TB, K=0.

-occurs(sell(S,K),T) :- target\_sell(S,TS,K,T), val\_price(S,P,T), P<TS, K=0.

% Change flags

changed\_shares(S,T) :- occurs(buy(S,\_),T).

changed\_shares(S,T) :- occurs(sell(S,\_),T).

changed\_cash(T) :- occurs(buy(\_,\_),T).

changed\_cash(T) :- occurs(sell(\_,\_),T).

changed\_cash(T) :- occurs(deposit(\_),T).

changed\_cash(T) :- occurs(withdraw(\_),T).

changed\_price(S,T) :- occurs(set\_price(S,\_),T).

changed\_target\_buy(S,T) :- occurs(buy(S,K),T).

changed\_target\_sell(S,T) :- occurs(sell(S,K),T).

% CWA

-occurs(buy(S,K),T) :- step(T), target\_buy(S,\_,K,T), not occurs(buy(S,K),T), K > 0.

-occurs(sell(S,K),T) :- step(T), target\_sell(S,\_,K,T), not occurs(sell(S,K),T), K > 0.

% Causal Laws

% SHARES

val\_shares(S,Q2,T+1) :-

step(T), T < n,

occurs(buy(S,K),T),

val\_shares(S,Q,T),

Q2 = Q + K.

val\_shares(S,Q2,T+1) :-

step(T), T < n,

occurs(sell(S,K),T),

val\_shares(S,Q,T),

Q2 = Q - K.

% PRICE

val\_price(S,P,T+1) :-

step(T), T < n,

occurs(set\_price(S,P),T).

% TARGET

target\_buy(S,TB,K2,T+1) :-

step(T), T < n,

target\_buy(S,TB,K,T),

changed\_target\_buy(S,T),

K2 = 0.

target\_sell(S,TS,0,T+1) :-

step(T), T < n,

target\_sell(S,TS,K,T),

changed\_target\_sell(S,T).

% Cash aggregation

sell\_val(V,S,K,P,T) :- occurs(sell(S,K),T), val\_price(S,P,T), V = K\*P.

buy\_val(V,S,K,P,T) :- occurs(buy(S,K),T), val\_price(S,P,T), V = K\*P.

cash\_plus(P,T) :-

step(T), T < n,

P = #sum { V,S : sell\_val(V,S,\_,\_,T); D : occurs(deposit(D),T) }.

cash\_minus(M,T) :-

step(T), T < n,

M = #sum { V,S : buy\_val(V,S,\_,\_,T); W : occurs(withdraw(W),T) }.

val\_cash(Cnext,T+1) :-

step(T), T < n,

val\_cash(C,T), cash\_plus(P,T), cash\_minus(M,T),

Cnext = C + P - M.

% Inertia

val\_shares(S,Q,T+1) :- step(T), T < n, val\_shares(S,Q,T), not changed\_shares(S,T).

val\_cash(C,T+1) :- step(T), T < n, val\_cash(C,T), not changed\_cash(T).

val\_price(S,P,T+1) :- step(T), T < n, val\_price(S,P,T), not changed\_price(S,T).

target\_buy(S,TB,K,T+1) :- step(T), T < n, target\_buy(S,TB,K,T), not changed\_target\_buy(S,T).

target\_sell(S,TS,K,T+1) :- step(T), T < n, target\_sell(S,TS,K,T), not changed\_target\_sell(S,T).

% Executability: targets & quantities

:- occurs(buy(S,K),T), not target\_buy(S,\_,K,T).

:- occurs(sell(S,K),T), not target\_sell(S,\_,K,T).

:- occurs(buy(S,K),T), target\_buy(S,TB,\_,T), TB=0, K>0.

:- occurs(sell(S,K),T), target\_sell(S,TS,\_,T), TS=0,K>0.

:- occurs(buy(S,0),T), target\_buy(S,\_,0,T).

:- occurs(sell(S,0),T), target\_sell(S,\_,0,T).

:- occurs(buy(S,K),T), val\_price(S,P,T), target\_buy(S,TB,K,T), P > TB.

:- occurs(sell(S,K),T), val\_price(S,P,T), target\_sell(S,TS,K,T), P < TS.

% Budget & position caps

:- occurs(buy(S,K),T), val\_cash(C,T), val\_price(S,P,T), K\*P > C.

:- occurs(buy(S,\_),T), occurs(sell(S,\_),T).

% Next-state cash bounds

:- step(T), T < n, val\_cash(C,T), cash\_plus(P,T), cash\_minus(M,T), C + P - M < 0.

% Initial state T=0

val\_cash(1000,0).

val\_shares(S,0,0) :- stock(S).

val\_price(aapl,120,0).

val\_price(msft,100,0).

val\_price(goog,110,0).

val\_price(tsla,100,0).

val\_price(amzn,100,0).

val\_price(nvda,120,0).

val\_price(meta,110,0).

val\_price(baba, 90,0).

val\_price(jpm, 90,0).

val\_price(xom, 100,0).

#show occurs/2.

#show val\_cash/2.

#show val\_shares/3.

#show val\_price/3.

#show target\_buy/4.

#show target\_sell/4.

# Query Evaluations

**Query A — Buy executes at/under target with exact quantity**

Query:

% Buy exactly 2 AAPL if price ≤ 120  
target\_buy(aapl,120,2,0).  
  
% At t=0 price is 120 → allowed  
occurs(buy(aapl,2),0).

Expected:

- t=1: *val\_shares(aapl,2,1)* and *val\_cash(760,1)* (1000 − 2×120).

Output: *(show val\_shares and val\_cash only)\_*

clingo version 5.7.2 (6bd7584d)

Reading from stdin

Solving...

Answer: 1

val\_cash(1000,0) val\_cash(760,1) val\_shares(aapl,0,0) val\_shares(msft,0,0) val\_shares(goog,0,0) val\_shares(tsla,0,0) val\_shares(amzn,0,0) val\_shares(nvda,0,0) val\_shares(meta,0,0) val\_shares(baba,0,0) val\_shares(jpm,0,0) val\_shares(xom,0,0) val\_shares(aapl,2,1) val\_shares(xom,0,1) val\_shares(jpm,0,1) val\_shares(baba,0,1) val\_shares(meta,0,1) val\_shares(nvda,0,1) val\_shares(amzn,0,1) val\_shares(tsla,0,1) val\_shares(goog,0,1) val\_shares(msft,0,1) val\_shares(msft,0,2) val\_shares(goog,0,2) val\_shares(tsla,0,2) val\_shares(amzn,0,2) val\_shares(nvda,0,2) val\_shares(meta,0,2) val\_shares(baba,0,2) val\_shares(jpm,0,2) val\_shares(xom,0,2) val\_cash(760,2) val\_shares(aapl,2,2)

SATISFIABLE

Models : 1

Calls : 1

Time : 0.057s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)

CPU Time : 0.000s

**Query B — Buy → price rises → Sell (target quantity)**

Query:

% Update n = 3

#const n = 3.

target\_buy(aapl,120,2,0).  
target\_sell(aapl,130,1,0).  
  
occurs(set\_price(aapl,130),1). % market move

Expected:

- t=2: val\_shares(aapl,1,3), val\_cash(890,3).

Output: *(show val\_shares and val\_cash only)*

clingo version 5.7.2 (6bd7584d)

Reading from stdin

Solving...

Answer: 1

val\_cash(1000,0) val\_cash(760,1) val\_shares(aapl,0,0) val\_shares(msft,0,0) val\_shares(goog,0,0) val\_shares(tsla,0,0) val\_shares(amzn,0,0) val\_shares(nvda,0,0) val\_shares(meta,0,0) val\_shares(baba,0,0) val\_shares(jpm,0,0) val\_shares(xom,0,0) val\_shares(aapl,2,1) val\_shares(xom,0,1) val\_shares(jpm,0,1) val\_shares(baba,0,1) val\_shares(meta,0,1) val\_shares(nvda,0,1) val\_shares(amzn,0,1) val\_shares(tsla,0,1) val\_shares(goog,0,1) val\_shares(msft,0,1) val\_shares(msft,0,2) val\_shares(goog,0,2) val\_shares(tsla,0,2) val\_shares(amzn,0,2) val\_shares(nvda,0,2) val\_shares(meta,0,2) val\_shares(baba,0,2) val\_shares(jpm,0,2) val\_shares(xom,0,2) val\_shares(xom,0,3) val\_shares(jpm,0,3) val\_shares(baba,0,3) val\_shares(meta,0,3) val\_shares(nvda,0,3) val\_shares(amzn,0,3) val\_shares(tsla,0,3) val\_shares(goog,0,3) val\_shares(msft,0,3) val\_cash(760,2) val\_cash(890,3) val\_shares(aapl,2,2) val\_shares(aapl,1,3)

SATISFIABLE

Models : 1

Calls : 1

Time : 0.059s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)

CPU Time : 0.000s

**Query C — Short sale: sell quantity exceeds holdings**

Query:

% Update n = 3

#const n = 3.

% We hold 0 MSFT initially;  
target\_sell(msft,110,2,0).

occurs(set\_price(msft,110),1).

Expected:

- t=1: val\_shares(msft,-2,3) and val\_cash(1220,3).

Output: *(show val\_shares and val\_cash only)*

clingo version 5.7.2 (6bd7584d)

Reading from stdin

Solving...

Answer: 1

val\_cash(1000,0) val\_cash(1000,1) val\_cash(1000,2) val\_shares(aapl,0,0) val\_shares(msft,0,0) val\_shares(goog,0,0) val\_shares(tsla,0,0) val\_shares(amzn,0,0) val\_shares(nvda,0,0) val\_shares(meta,0,0) val\_shares(baba,0,0) val\_shares(jpm,0,0) val\_shares(xom,0,0) val\_shares(xom,0,1) val\_shares(jpm,0,1) val\_shares(baba,0,1) val\_shares(meta,0,1) val\_shares(nvda,0,1) val\_shares(amzn,0,1) val\_shares(tsla,0,1) val\_shares(goog,0,1) val\_shares(msft,0,1) val\_shares(aapl,0,1) val\_shares(aapl,0,2) val\_shares(msft,0,2) val\_shares(goog,0,2) val\_shares(tsla,0,2) val\_shares(amzn,0,2) val\_shares(nvda,0,2) val\_shares(meta,0,2) val\_shares(baba,0,2) val\_shares(jpm,0,2) val\_shares(xom,0,2) val\_shares(xom,0,3) val\_shares(jpm,0,3) val\_shares(baba,0,3) val\_shares(meta,0,3) val\_shares(nvda,0,3) val\_shares(amzn,0,3) val\_shares(tsla,0,3) val\_shares(goog,0,3) val\_shares(aapl,0,3) val\_cash(1220,3) val\_shares(msft,-2,3)

SATISFIABLE

Models : 1

Calls : 1

Time : 0.061s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)

CPU Time : 0.000s

**Query D — Quantity mismatch is rejected (UNSAT)**

Query:

% Only target is buy 2 at 120  
target\_buy(aapl,120,2,0).  
% Attempt to buy 1 (no matching target for K=1)  
occurs(buy(aapl,1),0).

Expected:

- UNSATISFIABLE (violates “exact target quantity” rule).

Output:

clingo version 5.7.2 (6bd7584d)

Reading from stdin

Solving...

UNSATISFIABLE

Models : 0

Calls : 1

Time : 0.050s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)

CPU Time : 0.000s

**Query E — Short-cover: buy back to close after price drops**

Query:

% Update n = 3

#const n = 3.

% Open short at ≥100 for 2 shares; cover at ≤95 for 2 shares  
target\_sell(msft,100,2,0).  
target\_buy(msft,95,2,0).  
  
occurs(set\_price(msft,90),1). % price falls

Expected:

- t=2: val\_shares(msft,0,3), val\_cash(1020,3).

Output: *(show val\_shares and val\_cash only)*

clingo version 5.7.2 (6bd7584d)

Reading from stdin

Solving...

Answer: 1

val\_cash(1000,0) val\_cash(1200,1) val\_shares(aapl,0,0) val\_shares(msft,0,0) val\_shares(goog,0,0) val\_shares(tsla,0,0) val\_shares(amzn,0,0) val\_shares(nvda,0,0) val\_shares(meta,0,0) val\_shares(baba,0,0) val\_shares(jpm,0,0) val\_shares(xom,0,0) val\_shares(msft,-2,1) val\_shares(xom,0,1) val\_shares(jpm,0,1) val\_shares(baba,0,1) val\_shares(meta,0,1) val\_shares(nvda,0,1) val\_shares(amzn,0,1) val\_shares(tsla,0,1) val\_shares(goog,0,1) val\_shares(aapl,0,1) val\_shares(aapl,0,2) val\_shares(goog,0,2) val\_shares(tsla,0,2) val\_shares(amzn,0,2) val\_shares(nvda,0,2) val\_shares(meta,0,2) val\_shares(baba,0,2) val\_shares(jpm,0,2) val\_shares(xom,0,2) val\_shares(xom,0,3) val\_shares(jpm,0,3) val\_shares(baba,0,3) val\_shares(meta,0,3) val\_shares(nvda,0,3) val\_shares(amzn,0,3) val\_shares(tsla,0,3) val\_shares(goog,0,3) val\_shares(aapl,0,3) val\_cash(1200,2) val\_cash(1020,3) val\_shares(msft,-2,2) val\_shares(msft,0,3)

SATISFIABLE

Models : 1

Calls : 1

Time : 0.061s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)

CPU Time : 0.000s

**Query F — Buy order isn’t executed if not enough cash, deposit to fund a later buy**

Query:

% Set target buy for multiple stocks, not enough money for amzn order

target\_buy(aapl,120,2,0).

target\_buy(msft,100,2,0).

target\_buy(goog,110,2,0).

target\_buy(tsla,100,3,0).

target\_buy(amzn,100,2,0).

occurs(set\_price(amzn,90),1).

Expected:

- UNSATISFIABLE (violates “cash ≥ 0” rule).

Output:

clingo version 5.7.2 (6bd7584d)

Reading from stdin

Solving...

UNSATISFIABLE

Models : 0

Calls : 1

Time : 0.058s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)

CPU Time : 0.000s

**Query G —Deposit to fund a buy order**

Query:

% Update n = 1

#const n = 1.

% Set target buy for multiple stocks, not enough money for amzn order

target\_buy(aapl,120,2,0).

target\_buy(msft,100,2,0).

target\_buy(goog,110,2,0).

target\_buy(tsla,100,3,0).

target\_buy(amzn,100,2,0).

% Deposit to fund amzn order

occurs(deposit(500),0).

occurs(set\_price(amzn,90),1).

Expected:

- t=1: val\_shares(aapl, 2,1), val\_shares(msft, 2,1), val\_shares(goog, 2,1), val\_shares(tsla, 3,1), val\_shares(amzn, 2,1) and val\_cash(340,1).

Output: *(show val\_shares and val\_cash only)*

clingo version 5.7.2 (6bd7584d)

Reading from stdin

Solving...

Answer: 1

val\_cash(1000,0) val\_cash(340,1) val\_shares(aapl,0,0) val\_shares(msft,0,0) val\_shares(goog,0,0) val\_shares(tsla,0,0) val\_shares(amzn,0,0) val\_shares(nvda,0,0) val\_shares(meta,0,0) val\_shares(baba,0,0) val\_shares(jpm,0,0) val\_shares(xom,0,0) val\_shares(amzn,2,1) val\_shares(tsla,3,1) val\_shares(goog,2,1) val\_shares(msft,2,1) val\_shares(aapl,2,1) val\_shares(xom,0,1) val\_shares(jpm,0,1) val\_shares(baba,0,1) val\_shares(meta,0,1) val\_shares(nvda,0,1)

SATISFIABLE

Models : 1

Calls : 1

Time : 0.060s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)

CPU Time : 0.000s

**Query H — Withdraw decreases cash at next step**

Query:

% Update n = 1

#const n = 1.

% Withdraw  
occurs(withdraw(500),0).

Expected:

- t=1: val\_cash(500,1)

Output:

clingo version 5.7.2 (6bd7584d)

Reading from stdin

Solving...

Answer: 1

val\_cash(1000,0) val\_cash(500,1)

SATISFIABLE

Models : 1

Calls : 1

Time : 0.056s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)

CPU Time : 0.000s