Suitable abstract model for representing the market data feed format of different exchanges

Language theory and its use in hardware

Milan Dvořák

FIT BUT

Outline

Basic terms

Exchange

- highly organized market
- trading of financial instruments

Financial instrument

- basically any tradable asset
- stock (Apple, IBM, Facebook)
- commodities (wheat, coffee, oil, metals)

History

Floor trading

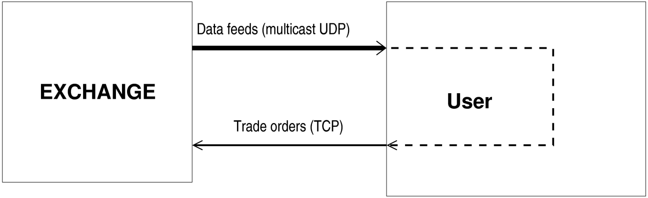
- traditional method, communication – *open outcry*

Electronic trading

- from 1990s, information technology

Electronic trading

- exchange sends information to users/traders – market data feed (UDP multicast)
  - traded instruments, prices, volumes, market sizes ...
- the received data are processed by user application
- user can make a decision (buy or sell)
- trade order is then sent back to the exchange

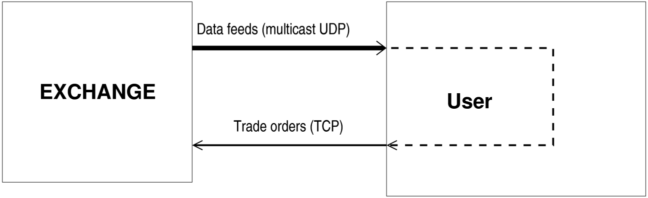


Electronic trading

- exchange sends information to users/traders – market data feed (UDP multicast)
  - traded instruments, prices, volumes, market sizes ...
- the received data are processed by user application
- user can make a decision (buy or sell)
- trade order is then sent back to the exchange

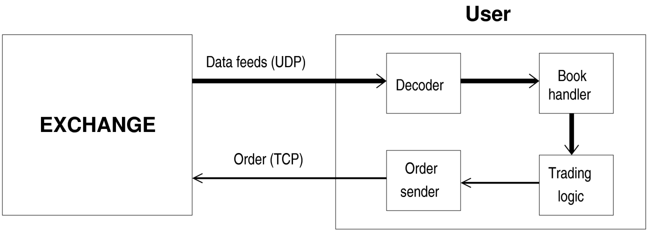
High Frequency Trading

- the decision is made by **algorithm**, not human



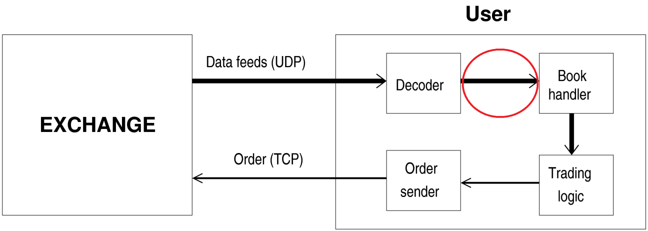
Typical HFT application

- Decoder – decoding of incoming data
- Book handler – processing the actual market data
  - storing best bids (asks) and offers for each instrument
- statistical functions
- trading logic (algorithm) makes decisions
- as fast as possible (low latency)



Typical HFT application

- Decoder – decoding of incoming data
- Book handler – processing the actual market data
  - storing best bids (asks) and offers for each instrument
- statistical functions
- trading logic (algorithm) makes decisions
- as fast as possible (low latency)



Control unit

Decoder output

- sequence of data fields (fixed bit width)
- fields grouped to messages
- message described by template
- each template has unique id (TID)
- some (sequence of) fields may be repeated (given number of iterations)

Control unit in Book handler

- recognition of messages and fields
- control (activation) of processing units
- synchronization of processing units

Suitable models for HW

Requirements

- automata (grammars are not suitable for HW)
- deterministic
- as simple as possible (but not simpler)

Pushdown Automata

- limited pushdown, we can store *some* values

Finite Automata

- perfect for use in hardware

Finite State Machine

A Finite State Machine is a sextuple

$$M = (Q, \Sigma, \Gamma, \delta, \omega, s)$$

where

Q is finite set of states

Σ is the input alphabet

Γ is the output alphabet

δ is the state-transition function

$$\delta : Q \times \Sigma \to Q$$

ω is the output function (Mealy model)

$$\omega : Q \times \Sigma \to \Gamma$$

$s \in Q$ is the start state

Finite Transducer

A Finite Transducer is a quintuple

$$M = (Q, \Sigma, R, s, F)$$

where

- Q is finite set of states
- Σ is an alphabet, $\Sigma = I \cup O$,
  I and O are input and output alphabets
- R is a finite set of rules of the form

$$pa \to qz$$

$$p, q \in Q, a \in I \cup \{\varepsilon\}, z \in O^*$$

- $s \in Q$ is the start state
- $F \subseteq Q$ is a set of final states

Differences in definitions

Let $M_1 = (Q_1, \Sigma_1, \Gamma_1, \delta_1, \omega_1, s_1)$ be a Finite State Machine and $M_2 = (Q_2, \Sigma_2, R_2, s_2, F_2)$ be a Finite Transducer ($\Sigma_2 = I_2 \cup O_2$).

We can observe (intuitively):

- $Q_1 = Q_2$ – finite set of states
- $\Sigma_1 = I_2$ – input alphabet
- $\Gamma_1 = O_2$ – output alphabet
- δ_1 and ω_1 are combined in rules in R_2 (ω_1 allows just 1 output symbol, R_2 can generate string of output symbols).
- $s_1 = s_2$ – start state
- $\emptyset = F_2$ – finite state machine in HW never stops, thus there are no final states,

or

- $Q_1 = F_2$ – FSM in HW stops when the power is lost, thus all states are final

Conclusion

Finite Transducer is a generalization of a Finite State Machine used in HW.

Pushdown Transducers

Pushdown Transducer

A Pushdown Transducer is a quintuple

$$M = (Q, \Sigma, R, s, F)$$

where

Q, s, F have the same meaning as in the case of finite transducers

Σ is an alphabet, $\Sigma = I \cup O \cup P_D$,
I, O, P_D are input, output and pushdown alphabets,
$S \in P_D$ is the start pushdown symbol

R is a finite set of rules of the form

$$Apa \to uqz$$

$A \in P_D$, $p, q \in Q$, $a \in I \cup \{\varepsilon\}$, $u \in P_D^*$ $z \in O^*$

Pushdown Transducers – Computational Step

Configuration

$$\chi \in P_D^* Q I^* \{|\} O^*$$

Move

If

$$r : Aqa \rightarrow upv \in R,$$

$$\chi = z{\color{red}Aqa}w|y,$$

$$\chi = z{\color{red}up}w|y{\color{red}v},$$

then

$$\chi \Rightarrow \chi' [r]$$

Pushdown Transducers – Translation

Translation of a Word

M translates x into y if

$$Ssx| \Rightarrow^* zf|y \text{ where } f \in F$$

Translation Defined by M

$$T(M) = \{(x, y) \in I^* \times O^* : Ssx| \Rightarrow^* zf|y, f \in F\}$$

- $\Rightarrow^*$ denotes the reflexive and transitive closure of $\Rightarrow$

Fields recognition

Consider two messages with ID 1 and 2.

- msg 1 – fields A, B, C
- msg 2 – fields A, B, D

Example input

1 A B C 1 A B C 2 A B D 3 . . .

We need 6 output bits:

- 1 bit to denote message id
- 1 bit to denote unknown message id (error)
- 4 bits to denote fields A, B, C and D respectively

Only one output symbol per rule is sufficient:

- let $\langle 000000 \rangle$ denote the 6 output bits as one alphabet symbol

Example output

$\langle 100000 \rangle \langle 001000 \rangle \langle 000100 \rangle \langle 000010 \rangle$ . . .

Example

$$M = (Q, \{1, 2, X\}, R, q_{id}, Q)$$

where

X denotes *don't care* value – arbitrary number other than 1 or 2

$Q = \{q_{id}, q_{1A}, q_{1B}, q_{1C}, q_{2A}, q_{2B}, q_{2D}, q_{err}\}$,

$$\begin{aligned}
R = \{\ & q_{id}1 \rightarrow q_{1A}\langle 100000\rangle \\
& q_{id}2 \rightarrow q_{2A}\langle 100000\rangle \\
& q_{id}X \rightarrow q_{err}\langle 010000\rangle \\
& \\
& q_{1A}X \rightarrow q_{1B}\langle 001000\rangle \\
& q_{1B}X \rightarrow q_{1C}\langle 000100\rangle \\
& q_{1C}X \rightarrow q_{id}\langle 000010\rangle \\
& \ldots \\
& q_{err}X \rightarrow q_{err}\langle 010000\rangle\ \}
\end{aligned}$$

Fields recognition – Pushdown Transducer

Example

$$M = (Q, \{1, 2, X, \$\}, R, q_{id}, Q)$$

where

X denotes *don't care* value – arbitrary number other than 1 or 2

$Q = \{q_{id}, q_A, q_B, q_C, q_D, q_{err}\}$,

$$\begin{aligned} R = \{ \ & \$q_{id}1 \rightarrow \$1q_A\langle 100000\rangle \\ & \$q_{id}2 \rightarrow \$2q_A\langle 100000\rangle \\ & \dots \\ & 1q_BX \rightarrow q_C\langle 000100\rangle \\ & 2q_BX \rightarrow q_D\langle 000100\rangle \\ & \dots \end{aligned}$$

- 2 states saved
- more complex code $\rightarrow$ more programming mistakes

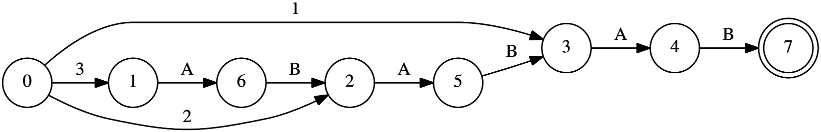
Repeated sequences

- messages can contain repeated sequences
- The sequence is preceded with the *length* field
- *length* is limited – e.g. 3

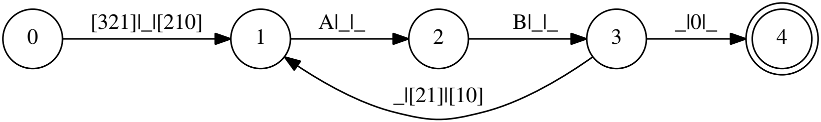
Example

- template – $LEN, (A, B)$
- valid inputs
  - $1, A, B$
  - $2, A, B, A, B$
  - $3, A, B, A, B, A, B$

Implementation of repeated sequences using finite automaton



Implementation of repeated sequences using pushdown automaton



Other tasks

Sequence number checking

- sequence number – 32 bit number (2^{32} symbols)
- compare incoming sequence number with value on pushdown
  1. set OK if values are equal
  2. set LOST if incoming value is greater
  3. set LATE if pushdown value is greater
  4. store incoming value $+ 1$ on pushdown

Detecting subsequent messages with the same instrument

- each instrument is labeled with SID - unique 32 bit value
- compare incoming SID value with value on pushdown
- set $NEW = 0$ if equal
- else set $NEW = 1$
- store incoming SID value on pushdown

Detecting new SID – rules snippet

Rules snippet

$\forall x : x q_{sid} x \Rightarrow x q_{next} \langle 0 \rangle, 0 \leq x < 2^{32}$

Example:

$8 q_{sid} 8 \Rightarrow 8 q_{next} \langle 0 \rangle$

$\forall x \forall y : x q_{sid} y \Rightarrow y q_{next} \langle 1 \rangle, 0 \leq x < 2^{32}, 0 \leq y < 2^{32} \wedge x \neq y$

Example:

$8 q_{sid} 9 \Rightarrow 9 q_{next} \langle 1 \rangle$

- we need $2^{32} + 2^{32} \cdot (2^{32} - 1) \doteq 2^{64}$ rules
- similar situation with the sequence number checking

Multiple values on pushdown

We need to store multiple values (SID, sequence number . . .). Pushdown transducer can acces only the topmost symbol on the pushdown $\rightarrow$ lot of states.

Extended Pushdown Automaton

- can read string from the pushdown
- extended PA is equivalent to the basic PA
- lot of rules (writing back of the other values, . . .)

Modification of the notation

- similar notation like Deep Pushdown Automata
- read and rewrite the mth topmost symbol
- $mApa \Rightarrow uqz$

Units synchronization

Computation of some units needs to be synchronized.

Inputs from units

- 1 if unit is ready
- 0 if unit is not ready

Example of synchronization

Do not continue until both units are ready.

Consequence

We have multiple inputs (data, control inputs).

Multiple inputs

Can we use the same trick as with multiple outputs?

- outputs have to be written every clock cycle (every transition)
- data input cannot be read while waiting for synchronization
- $\rightarrow$ we cannot join data and synchronization into one symbol

Modification of the notation

- split input symbols into multiple parts
- e.g. $\langle DATA, CTRL1, CTRL2 \rangle$
- each partial input can be ignored – ε

Units synchronization – example

Rules snippet

Suppose we are in state q_{sync} and we are waiting for synchronization before moving to the next state (q_{id}):

$q_{sync}\langle\varepsilon, 0, 0\rangle \Rightarrow q_{sync}\langle 000000\rangle$
$q_{sync}\langle\varepsilon, 1, 0\rangle \Rightarrow q_{sync}\langle 000000\rangle$
$q_{sync}\langle\varepsilon, 0, 1\rangle \Rightarrow q_{sync}\langle 000000\rangle$
$q_{sync}\langle\varepsilon, 1, 1\rangle \Rightarrow q_{id}\langle 000000\rangle$

We continue to read input data:

$q_{id}\langle 1, \varepsilon, \varepsilon\rangle \rightarrow q_{1A}\langle 100000\rangle$

. . .

1. High Frequency Trading
    - Financial Exchanges
    - HFT application
2. Abstract models
    - Finite State Machines
    - Transducers
3. Practical application
    - Modifications of the models
4. Summary

Summary

- High Frequency Trading – typical application
- control unit – recognizing messages/fields
- abstract models – finite state machines, finite transducers, pushdown transducers
- implementation of control unit
- proposed modifications (multiple inputs, *deep* pushdown)

Thank you for your attention

Questions?