Composition of

Time Interval Qualitative Relations

By: Shiru Hou

The algorithm for finding the qualitative relations (a:c) between two atomic time interval relations(a:b & b:c) can be implemented by referencing the relative start and stop points of the different relations. By noting where interval a starts and stops relative to interval b, and where interval b starts and stops relative to interval c, an correlation can be derived between a and c, and possible interval relations can be deduced. The five possible relative start or stop locations for any time interval (x) in relation to a second interval (y) can be expressed as existing before y (bf), at y-start(as), in y (i), at y-end (ae), and after y (af). The combinations of these relative locations create the thirteen different interval definitions.



Figure 1: Relative interval Positions

The thirteen relations; precedes (p), meets (m), overlaps (o), finished by (F), contains (D), starts (s), equals (e), started by (S), during (d), finishes (f), overlapped by (O), met by (M), and preceded by (P) become defined according to the start and end point locations (xs, xe). This allows comparison of relative interval limits. These comparisons can then be used to construct a series of conditional statements, or incorporated into a decision tree, to deduce to possible relations between different pairs of atomic intervals.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Relation | p | m | o | F | D | s | e | S | d | f | O | M | P |
| xs | bf | bf | bf | bf | bf | as | as | as | i | i | i | ae | af |
| Xe | bf | as | i | ae | af | i | ae | af | i | ae | af | af | af |

Figure : Relations by interval limit placement

Table :Program relational output chart 1/2

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | p | m | o | F | D | s | E |
| p | p | p | p | p | p | p | p |
| m | p | p | p | p | p | m | m |
| o | p | p | o m p | o p m | p m o F D | O | o |
| F | p | m | o | F | D | o | F |
| D | D p m o F | D o f | D o F | D | D | D o F | D |
| s | P | P | p m o | p m o | p m o F D | S | s |
| e | p | m | o | F | D | S | e |
| S | p m o F D | o F D | o F D | D | D | s e S | S |
| d | p | p | p m o d s | p m o d s | p m o F D s e S d f O M P | D | d |
| f | p | m | d o s | f e F | P D S O M | D | f |
| O | o F D p m | o F D | d f O o F D s e S | O D S | O M P D S | d f O | O |
| M | p m o F D | s e S | d f O | M | P | d f O | M |
| P | p m o F D s e S d f O M P | d f O M P | d f O M | P | P | d f O M P | P |

Table :Program relational output chart 2/2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | S | d | f | O | M | P |
| p | p | p m o s d | p m o s d | p m o s d | p m o s d | p m o F D s e S d f O M P |
| m | m | o d s | o d s | o d s | f F e |  |
| o | o F D | o d s | o s d | D S O o F e d f | D S O | D S O M P |
| F | D | o s d | F e f | D S O | D S O | D S O M P |
| D | D | D O S o F | D O S | D O S | D O S | D S O M P |
| s | s e S | d | d | d f O | M | P |
| e | S | d | f | O | M | P |
| S | S | d f O | O | O | M | P |
| d | d f O M P | d | d | d f O M P | P | P |
| f | P O M | d | f | P O M | P | P |
| O | O M P | d f O | O | O M P | P | P |
| M | P | d f O | M | P | P | P |
| P | P | d f O M P | P | P | P | P |

The algorithms and program used to generate these relations could be improved using a mathematical solution instead of Boolean indicators of relational position. If the problem were modified for use in 2-D Directional calculus, a mathematical solution could be treated as a three function piecewise evaluation of vector addition, where each vector is calculated with lengths of 1, 10, and 100 to demonstrate short, equal, and long interval or distance sizes.

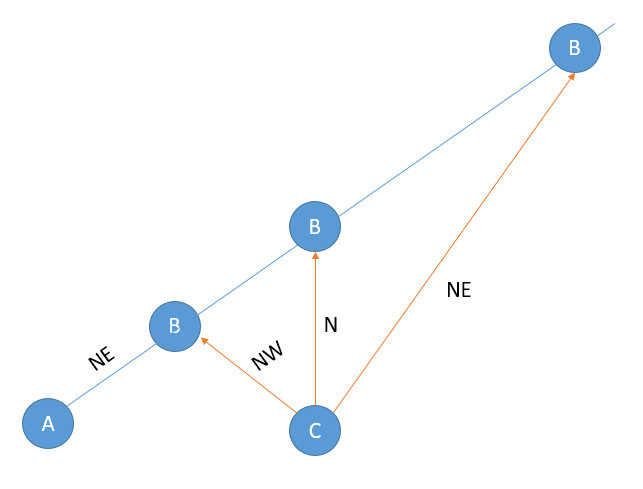
For directional calculus, the algorithm is similar to that we have discussed above. Let’s look at some examples.

Figure 3:Directional calculus example

If point B is north east to point A and C is east to point A, then the relation between B and C will be like this which is shown below. One of the difference of these two algorithm is the number of possible relations between A and C, given the relation of A and B and A and C. Modification will involve reducing the if-else condition.

Interval Relation Java Program:

**import** java.util.Scanner;

**public** **class** Interval {

String rel = **new** String();

**boolean**[] spos = **new** **boolean**[5];

**boolean**[] epos = **new** **boolean**[5];

**boolean** pre = **false**;

**boolean** meet = **false**;

**boolean** over = **false**;

**boolean** start = **false**;

**boolean** during = **false**;

**boolean** finish = **false**;

**boolean** equal = **false**;

**boolean** overBy = **false**;

**boolean** metBy = **false**;

**boolean** after = **false**;

**boolean** startBy = **false**;

**boolean** contain = **false**;

**boolean** finishBy = **false**;

**public** String get(){

**return** rel;

}

**public** **void** full(){

rel = "p m o F D s e S d f O M P";

}

**public** **void** set(String x){

rel = (rel.isEmpty()) ? x : rel + " " + x;

**switch** (x){

**case** "p":

pre = **true**;

spos[0] = **true**;

epos[0] = **true**;

**break**;

**case** "m":

meet = **true**;

spos[0] = **true**;

epos[1] = **true**;

**break**;

**case** "o":

over = **true**;

spos[0] = **true**;

epos[2] = **true**;

**break**;

**case** "s":

start = **true**;

spos[1] = **true**;

epos[2] = **true**;

**break**;

**case** "d":

during = **true**;

spos[2] = **true**;

epos[2] = **true**;

**break**;

**case** "e":

equal = **true**;

spos[1] = **true**;

epos[3] = **true**;

**break**;

**case** "f":

finish = **true**;

spos[2] = **true**;

epos[3] = **true**;

**break**;

**case** "F":

finishBy = **true**;

spos[0] = **true**;

epos[3] = **true**;

**break**;

**case** "D":

contain = **true**;

spos[0] = **true**;

epos[4] = **true**;

**break**;

**case** "S":

startBy = **true**;

spos[1] = **true**;

epos[4] = **true**;

**break**;

**case** "O":

overBy = **true**;

spos[2] = **true**;

epos[4] = **true**;

**break**;

**case** "M":

metBy = **true**;

spos[3] = **true**;

epos[4] = **true**;

**break**;

**case** "P":

after = **true**;

spos[4] = **true**;

epos[4] = **true**;

**break**;

**default**:

System.***out***.println("ERROR! : Invalid Interval!");

**break**;

}

}

**public** Interval eval(Interval b){

Interval result = **new** Interval();

**if** (rel.equals("e")){ //if a == b then b->c == a->c

result.set(b.get());

} **else** **if** (b.get().equals("e")){ //if b == c then a->b == a->c

result.set(get());

} **else** {

**if**(epos[0]){ //A ends before B \_/0:\_/\_

//B starts after C (P) \_/0:4/\_

**if** (b.spos[4]) result.full();

//B starts at C-end/in C (M, O, F, D) \_/0:2/\_, \_/0:3/\_

**if** (b.spos[3] || b.spos[2]) {

result.set("p");

result.set("m");

result.set("o");

result.set("s");

result.set("d");

}

**if** (b.spos[1]){ //B starts at C-start (s, e, S) \_/0:1/\_

result.set("p");

}

**if** (b.spos[0]){ //B starts before C (p, m, o, F, D) \_/0:0/\_

result.set("p");

}

}

**if**(epos[1]){ //A ends at B-start \_/1:\_/\_

**if**(b.spos[0]){ //B starts before c (p, m, o, F, D) \_/1:0/\_

result.set("p");

}

**if**(b.spos[1]){ //B starts at C-start (s, S) \_/1:1/\_

result.set("m");

}

**if**(b.spos[2]){ //B starts within C (d, f, O) \_/1:2/\_

result.set("o");

result.set("d");

result.set("s");

}

**if**(b.spos[3]){ //B starts at C-end (M) \_/1:3/\_

result.set("f");

result.set("F");

result.set("e");

}

**if**(b.spos[4]){ //B starts after C (P) \_/1:4/\_

result.set("P");

result.set("M");

result.set("O");

result.set("S");

result.set("D");

}

}

**if**(epos[2]){ //A ends within B \_/2:\_/\_

**if**(spos[0]){ //A starts before B (o) 0/2:\_/\_

//B ends before/at C (p) 0/2:\_/0, 0/2:\_/1

**if**(b.epos[0] || b.epos[1]){

result.set("p");

}

**if** (b.epos[2]){ //B ends within C (o, s, d) 0/2:\_/2

result.set("o");

**if**(b.spos[0]){ //0/2:0/2

result.set("m");

result.set("p");

}

**if** (b.spos[2]){ // 0/2:2/2

result.set("d");

result.set("s");

}

}

**if** (b.epos[3]){ //B ends at C-end (f, F) 0/2:\_/3

result.set("o");

**if**(b.spos[0]){ //B starts before C (F) 0/2:1/3

result.set("p");

result.set("m");

}

**if**(b.spos[2]){ //B starts within C (f) 0/2:2/3

result.set("s");

result.set("d");

}

}

**if** (b.epos[4]){ //B ends after C 0/2:\_/4

**if**(b.spos[0] || b.spos[1]){ // 0/2:0/4, 0/2:1/4

//B starts before C (D) 0/2:0/4

**if**(b.spos[0]) result.set("p");

result.set("m");

}

result.set("o");

result.set("F");

result.set("D");

} **else** { // 0/2:2/4, 0/2:3/4, 0/2:4/4

result.set("D");

result.set("S");

result.set("O");

//B starts within C (O) 0/2:2/4

**if**(b.spos[2]){

result.set("o");

result.set("F");

result.set("e");

result.set("d");

result.set("f");

result.set("s");

}

//B starts after C (P) 0:2/4/4

**if** (b.spos[4]){ result.set("M");

result.set("P");

}

}

}

}

**if** (spos[1]){ //A starts at B-start (s) 1/2:\_/\_

//B ends before C/at C-start (p, M) 1/2:\_/0, 1/2:\_/1

**if**(b.epos[0] || b.epos[1]){

result.set("P");

}

**if**(b.epos[2]){ //B ends within C 1/2:\_/2

**if**(b.spos[0]){ //B starts before C (o) 1/2:0/2

result.set("p");

result.set("m");

result.set("o");

}

**if**(b.spos[1]){ //B starts at C (s) 1/2:1/2

result.set("s");

}

**if**(b.spos[2]){ //B starts within C (d) 1/2:2/2

result.set("d");

}

//B ends at C-end/after C

**if**(b.spos[3] || b.spos[4]){

System.***out***.println(

"Error!!! Invalid B Interval!");

}

}

**if**(b.epos[3]){ //B ends at C-end 1/2:\_/3

**if**(b.spos[0]){ //B starts before C (F) 1/2:0/3

result.set("p");

result.set("m");

result.set("o");

}

**if**(b.spos[1]){ //B starts at C-start 1/2:1/3

result.set("s");

}

**if**(b.spos[2]){ //B starts within C (f) 1/2:2/3

result.set("d");

}

**if**(b.spos[3] || b.spos[4]){

System.***out***.println(

"Error!!! Invalid B Interval!");

}

}

**if**(b.epos[4]){ //B ends after C 1/2:\_/4

**if**(b.spos[0]){ //B starts before C (F) 1/2:0/4

result.set("p");

result.set("m");

result.set("o");

result.set("F");

result.set("D");

}

//B starts at C-start (S) 1/2:1/4

**if**(b.spos[1]){

result.set("s");

result.set("e");

result.set("S");

}

**if**(b.spos[2]){ //B starts within C (O) 1/2:2/4

result.set("d");

result.set("f");

result.set("O");

}

**if**(b.spos[3]){ //B starts at C-end (M) 1/2:3/4

result.set("M");

}

**if**(b.spos[4]){ //B ends after C (P) 1/2:4/4

result.set("P");

}

}

}

**if**(spos[2]){ //A starts within B (d) 2/2:\_/\_

//B ends before/at C-start 2/2:\_/0, 2/2:\_/1

**if**(b.epos[0] || b.epos[1]){

result.set("p");

}

//B ends in C/at C-end 2/2:\_/2, 2/2:\_/3

**if**(b.epos[2] || b.epos[3]){

//B starts before C (o, F) 2/2:0/2, 2/2:0/3

**if**(b.spos[0]){

result.set("p");

result.set("m");

result.set("o");

result.set("d");

result.set("s");

}

//B starts at C-start/in C

//2/2:1/2, 2/2:2/2, 2/2:1/3, 2/2:2/3

**if**(b.spos[1]|| b.spos[2]){

result.set("d");

}

//B starts at/after C-end C

**if**(b.spos[3] || b.spos[4]){

System.***out***.println(

"Error!!! Invalid B Interval!");

}

}

**if**(b.epos[4]){ //B ends after C 2/2:\_/4

//B starts before C (D) 2/2:0/4

**if**(b.spos[0]){

result.full();

}

//B starts at C-start/in C (S, O)

// 2/2:1/4, 2/2:2/4

**if**(b.spos[1] || b.spos[2]){

result.set("d");

result.set("f");

result.set("O");

result.set("M");

result.set("P");

}

//B starts at/after C-end (M, P)

// 2/2:3/4, 2/2:4/4

**if**(b.spos[3] || b.spos[4]){

result.set("P");

}

}

}

**if** (spos[3] || spos[4]){

System.***out***.println("Error!!! Invalid A Interval!");

}

}

**if**(epos[3]){ //A ends at B-end \_/3:\_/\_

**if** (b.epos[0]){ //B ends before C \_/3:\_/0

result.set("p");

}

**if** (b.epos[1]){ //B ends at C-start \_/3:\_/1

result.set("m");

}

**if**(spos[0]){ //A starts before B 0/3:\_/\_

**if** (b.epos[2]){ //B ends in C 0/3:\_/2

//B starts before/at/in C

// 0/3:0/2, 0/3:1/2, 0/2:2/2

**if**(b.spos[0] || b.spos[1] || b.spos[2]){

result.set("o");

**if**(b.spos[2]){ // 0/3:2/2

result.set("s");;

result.set("d");

}

}

}

**if**(b.epos[3]){ //B ends at C-end 0/3:\_/3

//B starts before/at C-start

// 0/3:0/3, 0/3:1/3, 0/3:2/3

**if**(b.spos[0] || b.spos[1] || b.spos[2]){

result.set("F");

}

**if**(b.spos[2]){ //B starts in C 0/3:2/3

result.set("e");

result.set("f");

}

}

**if**(b.epos[4]){ //B ends after C 0/3:\_/4

result.set("D");

//B starts in C

// 0/3:2/4, 0/3:3/4, 0/3:4/4

**if**(b.spos[2] || b.spos[3] || b.spos[4]){

result.set("S");

result.set("O");

**if** (b.spos[4]){ //0/3:4/4

result.set("M");

result.set("P");

}

}

}

}

**if**(spos[2]){ //A starts in B 2/3:\_/\_

**if** (b.epos[2]){ //B ends in C 2/3:\_/2

result.set("d");

**if** (b.spos[0]){ //B starts before C 2/3:0/2

result.set("o");

result.set("s");

}

}

**if** (b.epos[3]){ //B ends at C-end 2/3:\_/3

result.set("f");

**if**(b.spos[0]){ //B starts before C 2/3:0/3

result.set("e");

result.set("F");

}

}

**if** (b.epos[4]){ //B ends after C 2/3:\_/4

result.set("P");

**if**(b.spos[0]){ //B starts before C 2/3:0/4

result.set("D");

result.set("S");

result.set("O");

result.set("M");

}

// 2/3:1/4, 2/3:2/4

**if**(b.spos[1] || b.spos[2]){

result.set("O");

result.set("M");

}

}

}

}

**if**(epos[4]){ //A ends after B \_/4:\_/\_

**if**(spos[0]){ //A starts before B (D) 0/4:\_/\_

result.set("D");

**if**(b.epos[0]){ //B ends before C 0/4:\_/0

result.set("p");

result.set("m");

result.set("o");

result.set("F");

}

//B ends at C-start/in C/at C-end

// 0/4:3/3, 0/4:2/3

**if** (b.epos[3] && b.spos[3] || b.spos[2]){

result.set("O");

result.set("S");

}

// 0/4:\_/2, 0/4:\_/1

**if**(b.epos[2] || b.epos[1] && !b.spos[1]){

result.set("o");

result.set("F");

}

**if** (b.epos[4]){ //B ends after C 0/4:\_/4

**if**(b.spos[2] || b.spos[3]){ // 0/4:2/4, 0/4:3/4

result.set("O");

result.set("S");

}

**if**(b.spos[4]) { //B starts after C 0/4:0/4

result.set("S");

result.set("O");

result.set("M");

result.set("P");

}

}

}

**if**(spos[1]){ //A starts at B-start (S) 1/4:\_/\_

**if**(b.epos[0]){ //B ends before C 1/4:\_/0

result.set("p");

result.set("m");

result.set("o");

result.set("F");

result.set("D");

}

//B ends at C-start (m) 1/4:\_/1

**if**(b.epos[1]){

result.set("o");

result.set("F");

result.set("D");

}

**if**(b.epos[2]){ //B ends in C 1/4:\_/2

//B starts before C (o) 1/4:0/2

**if**(b.spos[0]){

result.set("o");

result.set("F");

result.set("D");

}

//B starts at C-start (s) 1/4:1/2

**if**(b.spos[1]){

result.set("s");

result.set("e");

result.set("S");

}

**if**(b.spos[2]){ //B starts in C (d) 1/4:2/2

result.set("d");

result.set("f");

result.set("O");

}

}

**if**(b.epos[3]){ //B ends at C-end 1/4:\_/3

**if**(b.spos[0]){ //B starts before C 1/4:0/3

result.set("D");

}

**if**(b.spos[2]){ //B starts in C 1/4:2/3

result.set("O");

}

}

**if**(b.epos[4]){ //B ends after C 1/4:\_/4

**if**(b.spos[0]){ //B starts before C (O) 1/4:0/4

result.set("D");

}

//B starts at C-start (S) 1/4:1/4

**if**(b.spos[1]){

result.set("S");

}

**if**(b.spos[2]){ //B starts in C 1/4:2/4

result.set("O");

}

**if**(b.spos[3]){ //B starts at C-end (M) 1/4:3/4

result.set("M");

}

**if**(b.spos[4]){ //B starts after C (P) 1/4:4/4

result.set("P");

}

}

}

**if**(spos[2]){ //A starts in B (O) 2/4:\_/\_

//B ends before/at C-start 2/4:\_/0, 2/4:\_/1

**if**(b.epos[0] || b.epos[1]){

result.set("o");

result.set("F");

result.set("D");

**if**(b.epos[0]){ // 2/4:\_/0

result.set("p");

result.set("m");

}

}

**if**(b.epos[2]){ //B ends in C 2/4:\_/2

result.set("d");

result.set("f");

result.set("O");

**if**(b.spos[0]){ //B starts before C (o) 2/4:0/2

result.set("o");

result.set("F");

result.set("D");

result.set("s");

result.set("e");

result.set("S");

}

}

**if**(b.epos[3]){ //B ends at C-end 2/4:\_/3

result.set("O");

**if**(b.spos[0]){ //B starts before C 2/4:0/3

result.set("D");

result.set("S");

}

}

**if**(b.epos[4]){ //B ends after C 2/4:\_/4

result.set("P");

//B starts before C

// 2/4:0/4, 2/4:1/4, 2/4:2/4

**if**(b.spos[0] || b.spos[1] || b.spos[2]){

result.set("O");

result.set("M");

**if**(b.spos[0]){ // 2/4:0/4

result.set("D");

result.set("S");

}

}

}

}

**if**(spos[3]){ //A starts at B-end (M) 3/4:\_/\_

**if**(b.epos[0]){ //B ends before C 3/4:\_/0

result.set("p");

result.set("m");

result.set("o");

result.set("F");

result.set("D");

}

**if**(b.epos[1]){ //B ends at C-start 3/4:\_/1

result.set("s");

result.set("e");

result.set("S");

}

**if**(b.epos[2]){ //B ends in C 3/4:\_/2

result.set("d");

result.set("f");

result.set("O");

}

**if**(b.epos[3]){ //B ends at C-end 3/4:\_/3

result.set("M");

}

**if**(b.epos[4]){ //B ends after C 3/4:\_/4

result.set("P");

}

}

**if**(spos[4]){ //A starts after B 4/4:\_/\_

**if** (b.epos[0]){ //B ends before C 4/4:\_/0

result.full();

}

//B ends at C-start/in C 4/4:\_/1, 4/4:\_/2

**if**(b.epos[1] || b.epos[2]){

result.set("d");

result.set("f");

result.set("O");

result.set("M");

result.set("P");

}

//B ends at/after C-end 4/4:\_/3, 4/4:\_/4

**if**(b.epos[3] || b.epos[4]){

result.set("P");

}

}

}

}

**return** result;

}

**public** **static** **void** main(String[] args) {

Scanner stdin = **new** Scanner(System.***in***);

Interval a;

Interval b;

Interval result;

String intIn;

String output = **new** String();

**do**{

a = **new** Interval();

b = **new** Interval();

System.***out***.println("Relationships = "

+ "{ p, m, o, F, D, s, e, S, d, f, O, M, P }");

System.***out***.print("Enter relation a (8 to exit):");

intIn = stdin.next();

**if** (!intIn.equals("8")){

a.set(intIn);

System.***out***.print("Enter a second relation:");

intIn = stdin.next();

b.set(intIn);

result = a.eval(b);

output = result.get();

System.***out***.println(output);

}

} **while** (!intIn.equals("0"));

stdin.close();

}

}

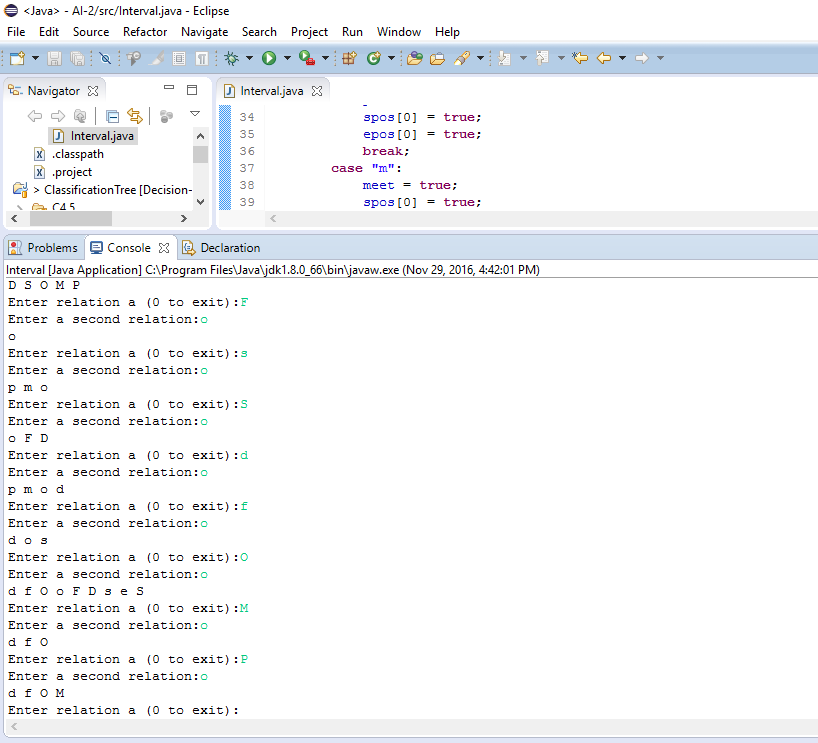


Figure : Interval Evaluation Program Execution