Draft description of
Patterns Recognition Language PRL(TM)

Prepared by Dmitry Marienko

Last edited 04.04.11

Basic elements

There are 2 basic elements for representing markets movements: movement (M) and point (P). Each point is presented by it's time and value. For example:

p : P('01-05-2009 00:00', 1.2340) etc. *(may be it won't be implemented at first release)*

Each movement (M) may be presented by different ways. For example by 2 points (p0 and p1) or by it's length, duration and direction (len==50, bar==10, dir=UP) or shorter (==50 && bar==10 && UP). In it's turn duration may be presented either in time units (minutes, hours, days ...) or in number of bars between the start and end points *(to consider)*. Direction is used to show orientation of this movement (up or down).
For example:

m : M(p0, p1), where
p0 : P('01-05-2009 00:00', 1.2340),
p1 : P('02-05-2009 00:00', 1.3340)

or as:

m : M(len == 100 && bar == 24 && dir==UP) or **m : M(len == 50 && time == 10h && dir==DOWN)**

or in shortest notation:

m : M(==100 && 24h && UP) or **m : M(==50 && ==10 && DOWN)**

Where **m** – variable that contains information about found movement type M (length, duration, direction etc). Length, duration and direction can be accessed from movement's variable by using follow modificators **.len**, **.time**, **.dir**:

m.len, m.bar, m.time, m.dir

Any movement's parameter can be skipped: length, duration, time or direction. For example:

m:M(==100) presents any movement with 100 pipses length to any direction
m:M(time==10) movement that is continuing for 10 hours to any direction and has any length
m:M() any movement at any direction

It's possible to provide add operation between the movements:

m1:M(==100 && bar==5 && UP)
m2:M(==50 && bar=3 && DOWN)
m3 = m1 + m2
m3==m:M(==50 && bar==8 && UP)

Additional conditions can be applied to the movement's parameters:

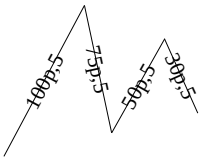
m : M(< 100 && bar >= 3) – any movement with length < 100 pipses and more or equals than 3 bars in duration
m : M(< 100 && > 50) – any movement with length in a range between 50 and 100 pipses

It's possible to use arithmetic expression inside conditions:
a : M() b:M() c : M(> a.len + 100 && < (a.len-b.len)/10)

Chains

Any set of movement can be included to the chain. To do that we need just to embrace movement's list by curly braces:

```
c0 : {
      m0:M(==100 && UP)
      m1:M(==75 && DOWN)
      m2:M(==50p && UP)
      m3:M(==30p && DOWN)
}
```



Take attention that movements of that the chain contsist of periodicaly changing directions: up, down, up, down .... If wrong movement's direction is occured than error should be generated. Example of wrong chain:

```
C0 : { m0:M(==100 && UP) m1:M(==100 && DOWN) m2:M(==100 && DOWN) }
```

Any chain can be named and different chains can be included in another chain again:

```
C1 : { m0:M(==100 && UP) m1:M(==100 && DOWN) }
C2 : { m0:M(>100) m1:M(<50) }
C3 : { c1:C1 c2:C2() }
```

Do we really need things like that : c1.m0.len, c1.m1.time, c3.c1.m0 ?
Or that: c1.elements[0].len ?
May be something like that
`c : { :M( > 100) :R(32) }` - chains without named elements ?

It is possible to set multiplicity to any chain's element.

`c : { m0:M(>50)* }` - set of movements length of each is more 50 pipses, **c.size** gives number of elements

`c : { m0:M(<100)|3,5| ] }` - more than 3 but not nire than 5 following movements have length less than 100 pipses each

Any chain may be presented as a movement. Therefore it can have a length, duration and direction also. So we could set restrictions on any chain:

`c : { m0:M( > 50)* } (> 1000 && bar<100)` – set of movements where each one has more than 50 pipses in length and summary length is more than 1000 pipses with duration is less than 100 bars.

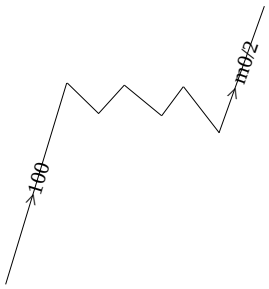
Any chain has following information: number of elements and array of these elements. This info can be accessible by using next modificators:

```
c : { m0:M(>50)* } (>1000 && time < 100)
c.size – number of elements in this chain
c.elements[0] – first elements of this chain  (not implemented yet)
c.elements[1] – second elements of this chain (not implemented yet)
...
```

Example: if we want to find pennon and following motion that is more than pennon's staff.

```
Pennon : {
            staff : M(>100)
            f : { :M() }* (<10)
         }

p:Pennon m:M(> p.staff.len/2)
```

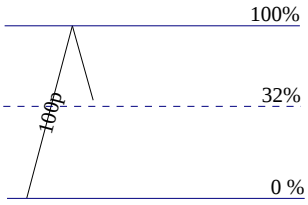


Retracements

Retracement shows how the market's price rolls back from the previous movement:

r1 : R(==32)
r2 : R(==68), where 32 and 68 are return in percentage from previous movement

```
: { m:M(==100) r:R(==32) }
```



Retracement can be presented in percents from previous movement's length. Rertracement is equal to movement in the reverse direction. For example if we have chain:

```
p : { m:M() r:R(>32) } → p : {  m:M() r:M(>100*len/m.len) }
```

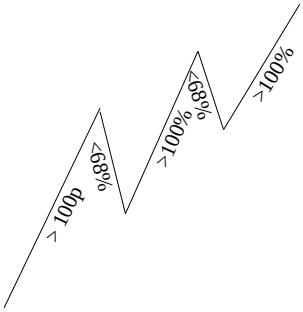
Retracement can also be used as ordinary movement in chains like this:

```
c : { m0:M(==100p) r0:R(==32) r1:R(==50) }
```

It means that after movement in 100 pips it should be 32% of 100 pips retracement (~32 pips) and than 50% of r0 retracement (again ~16 pips).

Retracements can be also included in the chain. Follow is an example of trend pattern. It seeks trends that starts from initial movement more than 100 pipses and subset of movements where each is less than 68% retracement from previous movement and retracement more than 100%:

```
trend : {
    m0:M(>100)
    m1 : { back:R(<68) forward:R(>100) }*
}
```



Def keyword

For ability of defining common rule for pattern **def** keyword is introduced. Defined pattern can be used inside another pattern again. Here we define triangle:

```
def Triangle : {
  first:M()
  t : { r0 : R(>32 && < 68)
        r1 : R(>95 && <105)*
      }
}
```

After that we can use this definition for searching 2 consecutive triangles (first is UP and second is down directed)

```
TwoTriangs : {
      t1 : Triangle(UP)
      t2 : Triangle(DOWN)
}
```

It allows us to create libraries of patterns for future use.

Trend lines (not implemented yet see Note8)

~~It should be the way to define lines and appropriate events: breakthrough, rejecting.~~

```
Triangle : { m0:M() :R(<50) :R(<100)* }
t:Triangle
sup:L(t.elements[1::2].p0)
res:L(t.elements[2::2].p0)
```

~~Another idea about the introducing the lines. We shouldn't produce lines from the found pattrens. For example we could describe line pattern a way like that:~~

~~line segment 1 (ls1), with length not more than 30 bars, angle is more 10 degrees (?) and line segment 2 (ls2) that is parallel to ls1 and distance between ls1 and ls2 isn't more than 100 pips.~~

~~It's line pattern model. After we need to describe pattern in terms of motions/retracements/points. Let's say it may be channel with more than 4 motions each is more than 50 pips length:~~

```
ch : { m:M(>50p)|,4| }
```

~~Now we could demand to satisfy the some conditions. For example: most of this channel's points should belong to the line pattern. If some point belongs to line it means that the distance between the point and line is less than some value.~~

```
ParaLines : { l1 : S(>100,>10) l2 : parallel(l1,<100p) }
pl : ParaLines
ch.points[].belong(lines)  // ????????????????
```

~~It doesn't look cool at all.~~

Candle patterns

We won't introduce candle patterns at this stage.

Some notes/ideas for further implementation

Note 1

We need to implement statistic calculations for defined pattern. We would be interesting in gathering information about frequency of considered pattern occurences and further price behaviour. For instance it would be extremely interesting to get next information:

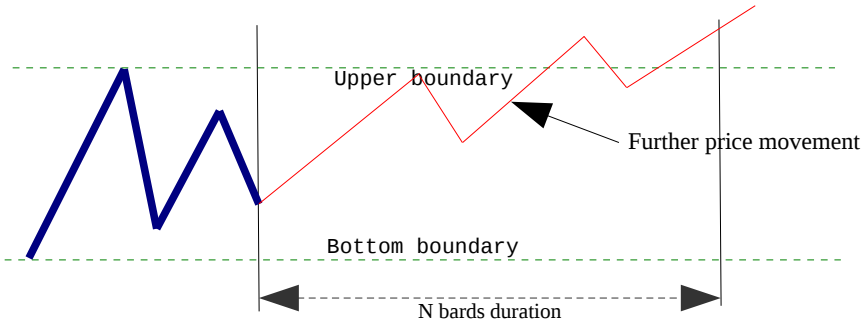
pattern X was found N times on given interval **(IMPLEMENTED)**

statistical tables for further price behaviour:

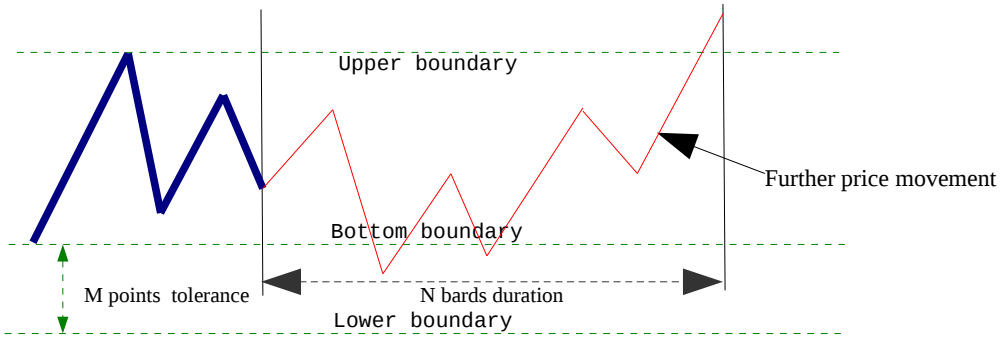
1. Statistics when price woved up/down by Y points without crossing bottom/up boundary of pattern during next N bars:

   Example: **Triang : { ….. }**

< 50 p	< 100 p	< 200 p	< 300 p
0.4	0.3	0.2	0.1



2. Statistics when price woved up/down by Y points wit crossing bottom/up boundary of pattern but not more than M points during next N bars:
   Example: **Triang : { ….. }**



Note 2 (IMPLEMENTED)

рассмотрим паттерн

c : { m:M() r:R(>32)* }

первое движение m - любое
потом должна идти последовательность из произвольного количества fibo retracenements, каждое из которых > 32 %

значение каждого паттерна R на этапе поиска рассчитывается исходя из текущего анализируемого движения и значения движения предшественника
т.е. в данной ситуации предшественником для всех R из последовательности r:R(>32)* будет m - (ссылка на него в r передается на этапе компиляции) { see ThisFiboRetracementValue class }

но есть и другая возможность трактовки такой записи - каждое r из последовательности r:R(>32)* - в качестве предшественника берет раннее найденный результат
уже на этапе поиска, т.е.

c : { m:M() r:R(>32)* }

что разворачивается в такую последовательность

c : { m:M() r0:M(m % len > 32) r1:M(r0 % len > 32) ... }

т.е. для 1-го r это будет m, для второго - предыдущий r и т.д. -- мы получим сужающуюся последовательность fibo откатов.

На текущий момент такое решение:

в классе SearchContext определен флаг
USE_RUNTIME_PRECEDESSOR_IDENTIFICATION по умолчанию он **== true**

Если false - в классе ThisFiboRetracementValue для вычисления возврата фиббоначи, будет использован предшествующий элемент найденный на этапе компиляции (1-й вариант)
если true - предшественник будет определяться на этапе выполнения поисковой процедуры в runtime (2-й вариант)

Note 3 (IMPLEMENTED)

Любой паттерн R-типа - просто свертка вот такого выражения:

..... x : M() r : R(>32) ... ----> x : M() r : M(> 100 * len / x.len)

вычисление размера отката в процентах (100 * len / x.len)
заменено оператором «%» :

100 * len / x.len ---> x.len % len ---> x % len

x : M() r : R(>32) развернется в **x : M() r : M(x % len > 32)**

Note 4 (IMPLEMENTED)

По поводу паттерна R-типа:

есть один ньюанс его использования такая запись:

r:R(>32) должна находить движение, которое больше чем 32% -й откат фибо от предыдущего движения - те внутри условия len - это значение отката в процентах (!!!!)

но если получить значение r.len извне:

r:R(>32) x:M(>r.len)

то r.len в условии x - будет возвращать значение движения в пунктах а не в процентах (из соображений что значение в процентах нужно только внутри самого R паттерна, для всех внешних по отношению к r паттернов будет интересно его абсолютное значение)

Другой ньюанс:

Если, например, рассмотрим конструкцию типа:

m : { a:M() w:{ r1:R(>32) r2:R(>100)|2| } }

она разворачивается в следующую последовательность:

m : { a:M() w[0]:{ r1:R(>32) r2:R(>100) w[1]:{ r1:R(>32) r2:R(>100) } }

T.o. w[0].r1 будет использовать a как базовый элемент для расчета величины возврата, w[0].r2 использует w[0].r1 - здесь все OK
если теперь рассмотрим w[1], то получается, что w[1].r1 будет использовать уже значение w[0] целиком - те сумму w[0].r1 и w[0].r2 !!!
Хотя предполагалось, что на самом деле он должен использовать w[0].r2 !!!!

Для этого в класс AbstractNode был введен метод getLastLeaf(), который будет возвращать последнюю листовую (leaf) ноду.

те для **m : { a:M() b:{ c:M() d:M() } }** метод вернет элемент **d**.

Для настройки в класс SearchContext введен параметр
RETURN_LAST_LEAF_IN_PRECEDS_NODE == true
если он == true то будет при вызове getPrecedesResult() будет использован метод getLastLeaf()

Note 5 (IMPLEMENTED)

Введение 'def' keyword.
Для возможности объявления шаблона паттерна (например для его дальнейшего использования внутри поискового паттерна)
вводится ключевое слово 'def' :

Например определяем шаблон паттерна треугольника:

def Ttiangle : { first:M() t : { :R(>32 && < 68) :R(>95 && <105)* } }

m : { t1 : Triangle(UP) t2 : Triangle(DOWN) } // здесь используем ранее определенный шаблон

таким образом удобно создавать библиотеки готовых паттернов и подключать их в дальнейшем

Note 6 (IMPLEMENTED)

Для конструкции типа:

```
m : M(==100)* n:M( > m.len)
```

в условии паттерна n, m.len - будет равно суммарному значению всех повторений паттерна m.

Т.е. если m поворился например 3 раза на движениях: (+100 pips, 3 bars) , (-100 pips, 2 bars), (+100 pips, 1 bars)
то :

```
m.len == 100 // 100 - 100 + 100
m.bar == 6 // 3+2+1
m.dir > 0
```

Note 7

Планы на будущее: (Доступ к индикаторам)

Хотелось бы иметь возможность использовать в паттернах не только ценовой ряд, но и показания индикаторов - ну например я ищу паттерн типа тренда вверх и хочу чтобы выполнялось требование чтобы при этом падал объем или я ищу какой-то паттерн на цене и при этом хочу чтобы в одной точке например значение RSI было меньше чем RSI в другой точке

вариант реализации:

```
x:{
    m1:M()
    r1:R(<62)
    m2:M(>r1)
    r2:R(>62 && @RSI(m1.end) > @RSI(m1.start))
}
```

те на последнем движении r2 - должно выполняться еще и условие, что значение RSI в точке начала движения m1 меньше чем RSI в конце того же движения

т.е. вводим еще 2 keywords: **start / end** - будет возвращать точки начала и конца движения и **@IndicatorName(point, indicator_parameters_list)** - будет возвращать значение индикатора в точке point если indicator_parameters_list == empty - берутся параметры по умолчанию

Можно таким образом находить например дивергенции цены и индикатора и тп

Дивергенция

Можно так же будет искать дивергенции:
грубый подход к поиску медвежьей дивергенции на индикаторе RSI:

```
BearDiv : {
    m1:M()
    r1:R(<62)
    m2:M(>r1) r2:R(>62 && @RSI(m1.end) < @RSI(m1.start))
}
```

грубый потому-что мы берем значения индикатора в точках где начинается(оканчивается) движение цены - а пики индикатора могут быть смещены (из-за запаздывания индикатора)

решение тут может быть простое - пускаем **ZigZag/SwingWaves** по значениям индикатора, -- (делаем это просто как отдельный пользовательский индикатор), но в отличие от стандартного ZigZag заполняем не только те точки в которых есть минимумы и максимумы, а все точки серии - т.е. нашли минимум - запихнули найденную точку в серию индикатора и копируем ее пока не найдем максимум и т.п.

Пробой/отбой от средней

Можно также дополнять условия паттерна доп. требованием пробоя например средней:

```
Break : { ..... m1:M() m2:M(>m1 && DOWN && @SMA(m2.end,21) > m2.end ) }
```

т.е. ищем некий паттерн в котором есть 2 последних движения m1 и m2, при этом m2 должно быть больше m1, направленно вниз и должно пробивать 21-периодную простую среднюю

Отбой тоже просто:

```
Reject : {
      .....
      m1:M(DOWN)
      m2:M(UP && @SMA(m2.start,21)
            + 20 > m2.start && @SMA(m2.start,21)< m2.start)
}
```

т.е. 2 последних движения m1 - вниз, m2 - вверх и так чтобы точка начала m2 лежала выше SMA(21) в пределах 20 пунктов

Note 8

Линии поддержки / сопротивления:

Линии поддержки / сопротивления так же реализуются просто как отдельные индикаторы. Работа с ними будет происходить через механизм доступа к индикаторам. Т.о. получим довольно общий подход:

Пробой линии поддержки:

```
BreakLine : {
      .....
      m1:M(UP)
      m2:M(DOWN)
      m3:M(UP)
      mBreak:M(mBreak.end < @Line(mBreak.end, m1.start, m3.start)))
}
```

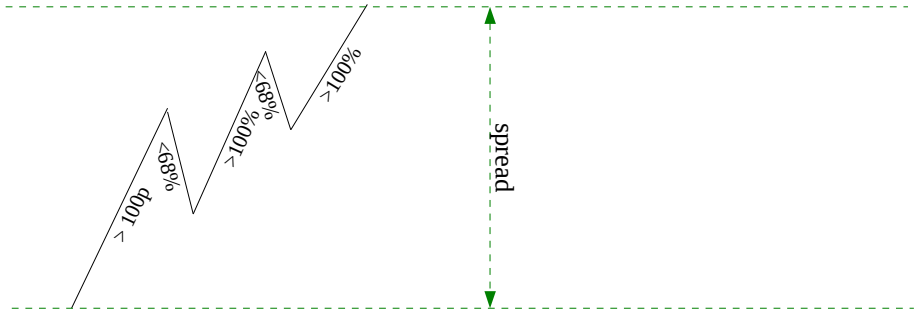
где **@Line(mBreak.end, m1.start, m3.start)** - это значение линии, постороенной по точкам (m1.start, m3.start) в точке mBreak.end.

Note 9

Рассмотреть необходимость введения поле num (?) - количество элементарных движений внутри более сложного
для **m:M()** num==1
для **x:{ m:M() n:M() }** num==2

Note 10

Ввести размах движения - фактически высота фигуры в пунктах.



Пример:

f0 : M(==100)
для такой фигуры f0.spread == 100 → f0.spread==f0.length (only in this case)

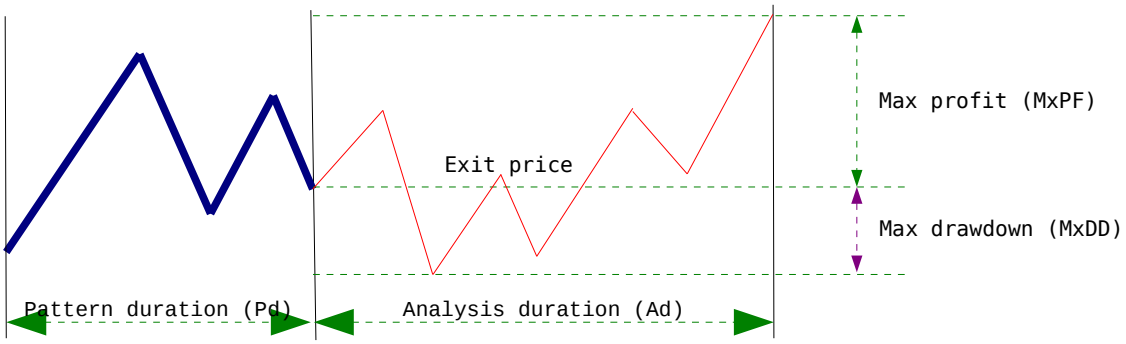
f1 : { a : M(==100 && UP) :R(<100) :M(==a.len + 50) }
для такой фигуры f1.spread == 150

Это позволит задавать ограничения по высоте фигуры.

PRL statistic (backtesting)

Let's consider some pattern occurence. It has duration (Pd) and exit price. We will analyse further price behaviour. For this purpose introduce time factor Tf. It will determine duration of analysing period (Ad).

Ad = Pd * Tf, where Tf >= 1



During analysing period we will find lower and upper price values. It will help us to define values of maximal profit and loss. Using these values we will create frequency histogramm of MxPF/MxDD ratio for all pattern occurences.
Also we will provide average values of **MxPF**, **MxDD** and **Pd** for all occurences. It gives us estimated ranges for take profit and stop loss values for trading strategy based on this pattern.