Weekly Daily EMA

AOS

Václav Stibor

Algoritmus
Obecný přístup

- Candles
- Timeframes
- Weekly box
- Daily arrow
- Multi-timeframe Moving Average (MA)
- Concrete setups
  - Pullbacks
  - Candle close to MA
  - Candle direction
  - MAs order

Algoritmus

Obecný přístup - cBot / indicator

- **cBot**
  - Obchodní algoritmus obstarávající veškerou logiku
  - Řeší správu obchodů
  - Výpočet při každé změně ceny na trhu
  - OnTick( )
    - Volání při změně close price
  - OnStart( ), OnStop( )

- **indicator**
  - Analytický nástroj navržený pro analýzu dat a jejich vizualizaci
  - Nemá přístup ke správy obchodů
  - Výpočet je prováděn na základě změny dat
  - Calculate( )
    - Volání při změně předem definovaných datových bodů
  - Initialize( )

Algoritmus

Obecný přístup - cBot / indicator

- **cBot**
  - Weekly-Daily-EMA
- **indicator**
  - WeeklyDirection
  - DailyDirection
  - FastMA
  - MediumMA
  - SlowMA
  - RSI
  - …

Exponential Moving Average (EMA)

EURUSD M30 (10,40,160)

Algoritmus

A setup

```
public class A : ISetup
{
    3 references
    private Algo Algo { get; set; }

    1 reference
    public A(Algo algo)
    {
        Algo = algo;
    }

    4 references
    public void Setup(Candles candles, bool isAllowed)
    {
        if (!isAllowed)
        {
            return;
        }

        if (candles.W1.Direction == Candle.EDirection.Negative &&
            candles.D1.Direction == Candle.EDirection.Negative)
        {
            if (OpenSellPositionConditions(candles))
            {
                PositionManager positionManager = new(Algo);
                positionManager.SetSellPositionAsync();
            }
        }
        else if (candles.W1.Direction == Candle.EDirection.Positive &&
                 candles.D1.Direction == Candle.EDirection.Positive)
        {
            if (OpenBuyPositionConditions(candles))
            {
                PositionManager positionManager = new(Algo);
                positionManager.SetBuyPositionAsync();
            }
        }
    }
```

```
public bool OpenSellPositionConditions(Candles candles)
{
    PullBack pullBack = new();

    if (pullBack.MediumMA(TradeType.Sell))
    {
        if (candles.Medium.CloseShortFastMA)
        {
            if (candles.Slow.Direction == Candle.EDirection.Negative || candles.Slow.CloseShortFastMA)
            {
                if (MovingAverages.Slow[0] > MovingAverages.Medium[0] && MovingAverages.Medium[0] > MovingAverages.Fast[0])
                {
                    return true;
                }
            }
        }
    }

    return false;
}
```

```
public bool OpenBuyPositionConditions(Candles candles)
{
    PullBack pullBack = new();

    if (pullBack.MediumMA(TradeType.Buy))
    {
        if (candles.Medium.CloseLongFastMA)
        {
            if (candles.Slow.Direction == Candle.EDirection.Positive || candles.Slow.CloseLongFastMA)
            {
                if (MovingAverages.Slow[0] < MovingAverages.Medium[0] && MovingAverages.Medium[0] < MovingAverages.Fast[0])
                {
                    return true;
                }
            }
        }
    }

    return false;
}
```

Algoritmus

A setup

```
public class A : ISetup
{
    3 references
    private Algo Algo { get; set; }

    1 reference
    public A(Algo algo)
    {
        Algo = algo;
    }

    4 references
    public void Setup(Candles candles, bool isAllowed)
    {
        if (!isAllowed)
        {
            return;
        }

        if (candles.W1.Direction == Candle.EDirection.Negative &&
            candles.D1.Direction == Candle.EDirection.Negative)
        {
            if (OpenSellPositionConditions(candles))
            {
                PositionManager positionManager = new(Algo);
                positionManager.SetSellPositionAsync();
            }
        }
        else if (candles.W1.Direction == Candle.EDirection.Positive &&
                 candles.D1.Direction == Candle.EDirection.Positive)
        {
            if (OpenBuyPositionConditions(candles))
            {
                PositionManager positionManager = new(Algo);
                positionManager.SetBuyPositionAsync();
            }
        }
    }
```

Algoritmus

A setup

```
public bool OpenSellPositionConditions(Candles candles)
{
    PullBack pullBack = new();

    if (pullBack.MediumMA(TradeType.Sell))
    {
        if (candles.Medium.CloseShortFastMA)
        {
            if (candles.Slow.Direction == Candle.EDirection.Negative || candles.Slow.CloseShortFastMA)
            {
                if (MovingAverages.Slow[0] > MovingAverages.Medium[0] && MovingAverages.Medium[0] > MovingAverages.Fast[0])
                {
                    return true;
                }
            }
        }
    }

    return false;
}
```

Algoritmus

A setup

```
/// <summary>
/// Asynchronously sets sell positions by placing limit orders for a specified number of orders.
/// </summary>
1 reference
public void SetSellPositionAsync()
{
    for (int i = 0; i < amountOfOrders; i++)
    {
        Interlocked.Increment(ref _numberOfPendingPlaceOrderOperations);

        // Place a limit sell order with specific parameters and callback function.
        PlaceLimitOrderAsync(TradeType.Buy,
                             SymbolName,
                             Volume,
                             Symbol.Bid + (Symbol.PipSize * 5),
                             "Trade_" + i,
                             OnOrderPlaced);
    }
}

/// <summary>
/// Callback method executed when an order is successfully placed.
/// </summary>
/// <param name="result">The result of the order placement operation.</param>
2 references
private void OnOrderPlaced(TradeResult result)
{
    if (Interlocked.Decrement(ref _numberOfPendingPlaceOrderOperations) == 0)
    {
        Algo.Print("All orders have been placed.");
    }
}
```

```
/// <summary>
/// Asynchronously cancels all pending orders with labels containing "Trade_".
/// </summary>
0 references
private void CancelAllPendingOrdersAsync()
{
    var pendingOrders = Algo.PendingOrders.Where(o => o.Label.Contains("Trade_")).ToArray();

    foreach (var order in pendingOrders)
    {
        Interlocked.Increment(ref _numberOfPendingCancelOrderOperations);

        // Cancel a pending order with a specific callback function.
        CancelPendingOrderAsync(order, OnOrderCancel);
    }
}

/// <summary>
/// Callback method executed when an order is successfully canceled.
/// </summary>
/// <param name="result">The result of the order cancellation operation.</param>
1 reference
private void OnOrderCancel(TradeResult result)
{
    if (Interlocked.Decrement(ref _numberOfPendingCancelOrderOperations) == 0)
    {
        Algo.Print("All orders have been canceled.");
    }
}
```

```
private bool IsAllowedToOpenBuyPos(int allowedCount)
{
    // Count the number of currently open Buy positions for the current trading symbol.
    int numberOfLongPositions = Positions.Where(pos => pos.Symbol == Symbol)
                                         .Where(pos => pos.TradeType == TradeType.Buy)
                                         .Count();

    if (numberOfLongPositions >= allowedCount)
    {
        return false;
    }

    return true;
}
```

```
private bool IsAllowedToOpenSellPos(int allowedCount)
{
    // Count the number of currently open Sell positions for the current trading symbol.
    int numberOfLongPositions = Positions.Where(pos => pos.Symbol == Symbol)
                                         .Where(pos => pos.TradeType == TradeType.Sell)
                                         .Count();

    if (numberOfLongPositions >= allowedCount)
    {
        return false;
    }

    return true;
}
```