Algorithm 1 HON+ rule extraction algorithm. Given the raw sequential data T, extracts arbitrarily high orders of dependencies, and output the dependency rules R. Optional parameters include $MaxOrder$, $MinSupport$, and $ThresholdMultiplier$

```
1:  define global $C$ as nested counter
2:  define global $D$,$R$ as nested dictionary
3:  define global $SourceToExtSource$, $StartingPoints$ as dictionary
4:
5:  function ExtractRules($T$, [$MaxOrder$, $MinSupport$, $ThresholdMultiplier = 1$])
6:      global $MaxOrder$, $MinSupport$, $Aggresiveness$
7:      BuildFirstOrderObservations($T$)
8:      BuildFirstOrderDistributions($T$)
9:      GenerateAllRules($MaxOrder$, $T$)
10:
11: function BuildFirstOrderObservations($T$)
12:     for $t$ in $T$ do
13:         for $(Source, Target)$ in $t$ do
14:             $C[Source][Target]$ += 1
15:             $IC$.add($Source$)
16:
17: function BuildFirstOrderDistributions($T$)
18:     for $Source$ in $C$ do
19:         for $Target$ in $C[Source]$ do
20:             if $C[Source][Target] < MinSupport$ then
21:                 $C[Source][Target] = 0$
22:         for $Target$ in $C[Source]$ do
23:             if then$C[Source][Target] > 0$
24:                 $D[Source][Target] = C[Source][Target]/(\sum C[Source][*])$
25:
26: function GenerateAllRules($MaxOrder$, $T$)
27:     for $Source$ in $D$ do
28:         AddToRules($Source$)
29:         ExtendRule($Source$, $Source$, 1, $T$)
30:
31: function KLDThreshold($NewOrder$,$ExtSource$)
32:     return $ThresholdMultiplier \times NewOrder/log_2(1 + \sum C[ExtSource][*])$
33: function ExtendRule($Valid$, $Curr$, $order$, $T$)
34:     if $Order \leq MaxOrder$ then
35:         AddToRules($Source$)
36:     else
37:         $Distr = D[Valid]$
38:         if $-log_2(min(Distr[*].vals)) <$ KLDThreshold($order + 1), Curr$ then
39:             AddToRules($Valid$)
40:         else
41:             $NewOrder = order + 1$
42:             $Extended =$ ExtendSource($Curr$)
43:             if $Extended = \emptyset$ then
44:                 AddToRules($Valid$)
45:             else
46:                 for $ExtSource$ in $Extended$ do
47:                     $ExtDistr = D[ExtSource]$
48:                     $divergence =$ KLD($ExtDistr, Distr$)
49:                     if $divergence >$ KLDThreshold($NewOrder, ExtSource$) then
50:                         ExtendRule($ExtSource, ExtSource, NewOrder, T$)
51:                     else
52:                         ExtendRule($Valid, ExtSource, NewOrder, T$)
```

Algorithm 1 *(continued)*

```
53: function AddToRules(Source):
54:     for $order$ in [1..len($Source$) + 1] do
55:         $s = Source[0 : order]$
56:         if not $s$ in $D$ or len($D[s]$) == 0 then
57:             ExtendSource($s$[1:])
58:         for $t$ in $C[s]$ do
59:             if $C[s][t] > 0$ then
60:                 $R[s][t] = C[s][t]$
61:
62: function ExtendSource($Curr$)
63:     if $Curr$ in $SourceToExtSource$ then
64:         return $SourceToExtSource[Curr]$
65:     else
66:         ExtendObservation($Curr$)
67:         if $Curr$ in $SourceToExtSource$ then
68:             return $SourceToExtsource[Curr]$
69:         else
70:             return $\emptyset$
71:
72: function ExtendObservation($Source$)
73:     if length($Source$) $> 1$ then
74:         if not $Source[1 :]$ in $ExtC$ or $ExtC[Source] = \emptyset$ then
75:             ExtendObservation($Source[1 :]$)
76:     $order = length(Source)$
77:     define $ExtC$ as nested counter
78:     for $Tindex, index$ in $StartingPoints[Source]$ do
79:         if $index - 1 \leq 0$ and $index + order < length(T[Tindex])$ then
80:             $ExtSource = T[Tindex][index - 1 : index + order]$
81:             $ExtC[ExtSource][Target] + = 1$
82:             $StartingPoints[ExtSource]$.add(($Tindex, index-1$))
83:     if $ExtC = \emptyset$ then
84:         return
85:     for $S$ in $ExtC$ do
86:         for $t$ in $ExtC[s]$ do
87:             if $ExtC[s][t] < MinSupport$ then
88:                 $ExtC[s][t] = 0$
89:             $C[s][t] + = ExtC[s][t]$
90:         $CsSupport = \sum ExtC[s][*]$
91:         for $t$ in $ExtC[s]$ do
92:             if $ExtC[s][t] > 0$ then
93:                 $D[s][t] = ExtC[s][t]/CsSupport$
94:                 $SourceToExtSource[s[1 :]].add(s)$
95:
96: function BuildSourceToExtSource($order$)
97:     for $source$ in $D$ do
98:         if $len(source) = order$ then
99:             if $len(source) > 1$ then
100:                $NewOrder = len(source)$
101:                for $startingin[1..len(source)]$ do
102:                    $curr = source[starting :]$
103:                    if not $curr$ in $SourceToExtSource$ then
104:                        $SourceToExtSource[curr] = \emptyset$
105:                    if not $NewOrder$ in $SourceToExtSource[curr]$ then
106:                        $SourceToExtSource[curr][NewOrder] = \emptyset$
107:                    $SourceToExtSource[curr][NewOrder].add(source)$
```