The Canonical System by Emil Post as the Predecessor of Production Systems

Post systems are string rewriting (substitution) systems which are defined over a finite alphabet $V$ and consist of:

- a finite set of substitution rules and
- a finite set of strings in $V$ (Axioms)

Example: Production of all palindromes (strings that are equal in both reading directions, e.g.: ‘kayak’, ‘step on no pets’) over an alphabet.

Alphabet: $V = \{a, b, c\}$
Axioms: $a, b, c, aa, bb, cc$
Substitution rules: $\$ \rightarrow a\$a$  
$\$ \rightarrow b\$b$  
$\$ \rightarrow c\$c$

$\$ can be substituted by an arbitrary chain in $V$.  
$\$ $\rightarrow$ $aca$ $\rightarrow$ $bacab$ $\rightarrow$ $cbacabc$

Differences to production systems:

- Working memory consists of less structured sequences of symbols.

- Right side of the rules includes only substitution instructions but no operation instructions (e.g. delete) for the working memory.

- Conflict resolution is undefined (non-deterministic selection of matching rules that fire next)
Fundamental Terms of Production Systems

Each production (synonym: inference rule) consists of two parts:

\[ A \rightarrow B \]

- Premise
- Antecedence
- Evidence
- If - part
- Left Side (LHS)
- Condition

Conclusion
Consequence
Hypothesis
Then - part
Right Side (RHS)
Action

Productions are evaluated over a „database“. „Database“ is also referred to as ’Working Memory‘ (WM) or in Cognitive Psychology as ’Short Term Memory‘ (STM).

There are two different evaluation modes for productions:

<table>
<thead>
<tr>
<th>Forward Chaining</th>
<th>Backward Chaining</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ A \rightarrow B ]</td>
<td>[ A \leftarrow B ]</td>
</tr>
<tr>
<td>data-driven inference</td>
<td>goal-driven inference</td>
</tr>
<tr>
<td>antecedence-oriented inference</td>
<td>consequence-oriented inference</td>
</tr>
<tr>
<td>bottom-up inference</td>
<td>top-down inference</td>
</tr>
<tr>
<td>if-added methods</td>
<td>if-needed methods</td>
</tr>
<tr>
<td>LHS-controlled chaining</td>
<td>RHS-controlled chaining</td>
</tr>
</tbody>
</table>
Expert Systems Tools: History

- 1943: Post production rules
- 1957: GPS started (Newell / Simon)
- 1958: LISP
- 1973: MYCIN (expert system, medical diagn.)
- 1977: OPS
- 1979: Rete algorithm
- 1985: CLIPS (NASA)
- 1995: JESS (Sandia Corp.)
Problem P can be solved by using forward chaining (P) or backward chaining (P).

**Forward Chaining (X)**

WM := initial knowledge
UNTIL P is solved or no production is left to fire
DO:
   (1) Determine set K of all rules whose premises are fulfilled by WM
   (2) Choose a rule R out of K using the conflict resolution strategy
   (3) WM := evaluation result obtained by applying R on WM

**Backward Chaining (X)**

WM := initial knowledge
UNTIL P is solved or no production is left to fire
DO:
   (1) Determine set K of all rules whose conclusion can be unified with P
   (2) Choose a rule R out of K using the conflict resolution strategy
   (3) If premise of R not in WM, perform backward chaining (premise of R)

Backtracking is applicable in step (2).
Commutative Production Systems

In exceptional cases, irrevocably evaluation modes without backtracking can be used for production systems.

A production system is called commutative if it fulfills the following requirements relating to any arbitrary database D:

- Each production that is applicable to D is also applicable to all databases that are created by an applicable rule.

- If the termination condition of D is fulfilled, it is also fulfilled for each database that is created by a rule applicable to D.

- The database that is created by using an arbitrary sequence of applicable rules to D is invariant regarding permutations of the rule.

Example for a commutative production system:

The use of an ‘unfavorable rule‘ in the example will only prolong the inference path without preventing the successful termination.
Decomposable Production Systems

A production system is called decomposable if its database and the termination condition are decomposable.

The database is decomposable if it can be split into components which can be processed separately (parallel) by the production system interpreter.

The termination condition is decomposable if the condition can be described with the termination conditions of the decomposed database components.

Example
Database:  (C,B,Z)
Termination condition: database only consists of M's
Productions:  
R1: C → (D,L)
R2: C → (B,M)
R3: B → (M,M)
R4: Z → (B,B,M)

Decomposition of database:  C,B,Z
Decomposition of termination conditions:  each decomposed database leads to M's
Pros and Cons of Production Systems

**Pros**

- Expressive power of representation language
  - +
  - +
- Extensibility
  - +
- Static analysis of control flow
  - +
- Realization of complex control structures
  - +

**Cons**

- Very limited syntax and semantics
  - -
- Indirect rule interaction with working memory
  - -
- Programming with side effects
  - -

**Features**

- Machine Understanding
  - +
  - +
  - +
- Modular Knowledge Representation
  - +
  - +
  - +

- Consistency check
  - +
- Explanation component
  - +
- Knowledge acquisition
  - +
- Error correction
  - +
The Use of Meta-Rules in Rule-Based Systems

**Meta-knowledge** is knowledge that refers to other knowledge units in the knowledge base. **Meta-rules** refer to form and content of the productions that are stored in the knowledge base.

(1) **Meta-rules for conflict resolution**
Consider an expert system with the following rules that gives advise in the case of a chemical contamination:
R1: If contamination caused by sulfuric acid, use procedure A.
R2: If contamination caused by sulfuric acid, use procedure B.

*Forward chaining can lead to conflict set \{R1, R2\}.*
Assume the knowledge base includes the following information:
- Procedure A is expensive, procedure B is cheap
- Procedure A is not dangerous, procedure B is dangerous
- R1 entered by expert Mr. Meyer, R2 entered by novice Mr. Schulz

**Meta-rules:**
R3: Prefer rules that recommend inexpensive procedures instead of expensive procedures.
R4: Prefer rules that recommend less dangerous procedures instead of dangerous procedures.
R5: Prefer rules that were entered by experts instead of rules that were entered by novices.
Conflict Resolution with Meta-Rules

(1) R4, R5: R1 before R2
(2) R3: R2 before R1
The decision depends on:
- for common interpreter for rules and meta-rules: evaluation
- for separated interpreters for meta-rules:
  - for random selection: (1) more likely
  - for ‘majority decision‘: (1)

Reflexive use of meta rules as **meta-meta-rules**
- Assume that the knowledge base includes the following information:
  R4 was entered by expert Mr. Meyer,
  R3 was entered by novice Mr. Schulz
  R5: R4 before R3: R1 before R2
- Assume that the knowledge base includes the following information:
  Using R3 is dangerous,
  using R5 is less dangerous,
  using R4 is not dangerous,
  R4: R4 before R3, R5: R1 before R2

In the case of a conflict between meta-rules, a further conflict resolution strategy could be needed.
An infinite regress will be prevented on a certain meta-level by selecting a production out of the conflict set randomly.
In practice, more than 2 meta-levels haven`t been necessary so far.
The Rule Pyramid

The rule pyramid shows typical relative sizes of object-rule levels and metal-rule levels in production systems.

Because of the lower number of rules on the higher meta-levels, the probability of a conflict set with more than one element becomes smaller.

The application of **intelligent conflict resolution** (more resources needed) in the sense of a **best-first search strategy** makes only sense on the lower levels of the rule pyramid.
Conflict Resolution on the Basis of a Priority Scheme for Rules, Specialization Relations between Instantiations and Dwell Time in the WM

WM ( (P S) (Q T) (P T) (R V) (Q S) (P V) ... (W V) (W T) )
P1( (Q =X) (P =X) / ... )
   I1₁ [ P1 ((Q T) (P T)) ]
   I1₂ [ P1 ((Q S) (P S)) ]
P2( (P S) (P =X) (W =X) / ... )
   I2₁ [ P2 ((P S) (P T) (W T)) ]
   I2₂ [ P2 ((P S) (P V) (W V)) ]
P3( (=X S) (=X =Y) (W =Y) (R =Y) (Q S) / ... )
   I3 [ P3 ((P S) (P V) (W V) (R V) (Q S)) ]
P4( (Q S) I(U S) (P =X) I(U V) I(U T) / ... )
   I4₁ [ P4 ((Q S) (P S)) ]
   I4₂ [ P4 ((Q S) (P T)) ]
   I4₃ [ P4 ((Q S) (P V)) ]
PO1 \{I₁₁, I₁₂\}
PO2 \{I₁₁, I₁₂, I₃\}
SC1 \{I₂₁, I₂₂, I₃, I₄₁, I₄₂, I₄₃\}
SC2 \{I₁₁, I₂₁, I₃, I₄₂\}
SC3 \{I₁₁, I₂₁, I₃, I₄₁, I₄₂, I₄₃\}
R1 \{I₁₂, I₂₁, I₂₂, I₃, I₄₁\}
R2 \{I₁₁, I₁₂, I₂₁, I₂₂, I₃, I₄₁\}
R3 \{I₁₁\}
R₄ \{I₁₁, I₁₂, I₁₄\}
Conflict Resolution Strategies

In the working memory, the data elements are ordered from left to right with respect to their increasing dwell time.

(P S) and (Q T) stored in the latest cycle, (P T) and (R V) stored two cycles ago, (Q S) and (P V) stored 3 cycles ago, (W V) and (W T) were stored 101 cycles ago in the working memory.

Productions are stored with respect to the order of their input. Variables are prefixed with ‘=’.

(1) Conflict resolution based on rule priorities

- PO1: Total order of rules, e.g. order of entry. with that: P1 is selected.
- PO2: Partial order of rules, P1 dominates P2, P3 dominates P4. with that: P1 and P3 are selected. PO2 is relevant for structured rule-sets.
Conflict Resolution Strategies

(2) Conflict resolution based on specialization relations between instantiations and productions

- SC1: S is a special case of production A, if
  * S has at least as many premises as A and
  * for each premise in A including constants, exists a corresponding premise in S which includes these constants as a subset and
  * S and A are not identical

Thus, P4 is a special case of P1. If P4 is instantiated, the instantiations of P1 are excluded. There are no specialization relations in the sense of SC1 between P2, P3 and the other productions.

- SC2: S is a special case of instantiation A, if S includes the data elements of A as a proper subset.
  With that, I3 is a special case of I1, I2, I4, and I4.

- SC3: Besides SC2, the production of S must include more premises than the production of A.
  With that, I4 and I4 would cease as special cases because P4 and P3 have the same number of premises.
Conflict Resolution Strategies

… based on dwell time in working memory

- R1: Order on the basis of the element of an instantiation with the shortest dwell time in the working memory. Time is defined as the number of executed actions since the storage of the element.

- R2: Same as R1, whereby time is defined as the number of interpreter cycles since the last storage.

- R3: Check the oldest elements of all instantiations. Choose the instantiation with the shortest dwell time in the sense of R1 and its element.

- R4: Time is defined as in R2. Check the oldest elements of all instantiations as in R3. The instantiations with an element \( \leq 100 \) dwell time cycles are preferred compared to instantiations with a dwell time \( \geq 101 \) cycles.

- R5: Time is defined as the number of executed actions. Compare the elements of all instantiations. First, compare the elements with the shortest dwell time. If the elements have the same dwell time, compare elements with second shortest dwell time. Once an instantiation is found that includes an element with a shorter dwell time than the comparison element, this instantiation is selected. Instantiations that have not enough elements for the successively comparison can not be selected. In R5, only instantiations with identical elements are equal.
Conflict Resolution Strategies

... based on a comparison of instantiations in the conflict set with instantiations that were used before.

- D1: Productions should be prevented from firing in two immediately successive cycles. Two instantiations are not identical, if the related productions are different. Check whether a different instantiation has fired before an interpreter cycle than the instantiation which can be chosen. If the comparison check is positive, the corresponding instantiation is preferred.

- D2: Instantiations should fire only once. Two instantiations are not equal if their productions or their data elements are different. Observe the whole firing sequence, whereby instantiations that are different to all previously executed instantiations are preferred.

random selection of an instantiation AD1 as the last decision support
Combinations of Conflict Resolution Strategies

'!' := Strategies are applied to the same set of instantiations. The result is the intersection of the calculated subsets.

'/' := The strategy on the right side is applied to the result of the strategy on the left side.

'[ ]' := All instantiations that are not preferred by the strategy in the brackets, are excluded from further analysis.

WM ( (P S) (Q T) (P T) (R V) (Q S) (P V) ... (W V) (W T) )

P1 ( (Q =X) (P =X) / ... )
  I1_1 [ P1 ((Q T) (P T)) ]
  I1_2 [ P1 ((Q S) (P S)) ]

P2 ( (P S) (P =X) (W =X) / ... )
  I2_1 [ P2 ((P S) (P T) (W T)) ]
  I2_2 [ P2 ((P S) (P V) (W V)) ]

P3 ( (=X S) (=X =Y) (W =Y) (R =Y) (Q S) / ... )
  I3 [ P3 ((P S) (P V) (W V) (R V) (Q S)) ]

P4 ( (Q S) I(U S) (P =X) I(U V) I(U T) / ... )
  I4_1 [ P4 ((Q S) (P S)) ]
  I4_2 [ P4 ((Q S) (P T)) ]
  I4_3 [ P4 ((Q S) (P V)) ]

[D2] / R1 / SC2 / R3 {I3}
[D2 x R4] / RI / SC2 {I1_2 I4_1}
[D2 x R4] / R5 {I1_2 I4_1}
[D2 x R4] / R5 / PO1 / AD1 {I1_2}
Multiple Derivation as Multiple And/Or-Tree

Inference rules with evidence values

\[(A \circ (B \cap C) / Z, EV_1)\]
\[(D \cap E / Z, EV_2)\]
\[(F / B, EV_3)\]
\[(G / B, EV_4)\]
\[(H \circ I / D, EV_5)\]

Goal statement

backward chaining

Factual knowledge
The Basic Syntax of OPS5

1) The structure of working memory elements

\[
\begin{align*}
\langle \text{element} \rangle & ::= \langle \text{vector-element} \rangle \mid \langle \text{av-element} \rangle \\
\langle \text{vector-element} \rangle & ::= \{\langle \text{value} \rangle \}^+ \\
\langle \text{av-element} \rangle & ::= \langle \text{object} \rangle \{^\langle \text{attribute} \rangle \langle \text{value} \rangle \}^+
\end{align*}
\]

2) The structure of production rules

\[
\begin{align*}
\langle \text{rule} \rangle & ::= \left( P \langle \text{rule-name} \rangle \langle \text{antecedent} \rangle \rightarrow \langle \text{consequent} \rangle \right) \\
\langle \text{antecedent} \rangle & ::= \{\langle \text{condition} \rangle \}^+ \\
\langle \text{condition} \rangle & ::= \langle \text{pattern} \rangle \mid -\langle \text{pattern} \rangle \\
\langle \text{pattern} \rangle & ::= \langle \text{object} \rangle \{^\langle \text{attribute} \rangle \langle \text{value} \rangle \}^+ \\
\langle \text{consequent} \rangle & ::= \{\langle \text{action} \rangle \}^+ \\
\langle \text{action} \rangle & ::= \left( \text{MAKE} \langle \text{object} \rangle \{^\langle \text{attribute} \rangle \langle \text{value} \rangle \}^+ \right) \mid \\
& \quad \left( \text{MODIFY} \langle \text{pattern-number} \rangle \{^\langle \text{attribute} \rangle \langle \text{value} \rangle \}^+ \right) \mid \\
& \quad \left( \text{REMOVE} \langle \text{pattern-number} \rangle \right) \mid \\
& \quad \left( \text{WRITE} \{\langle \text{value} \rangle \}^+ \right)
\end{align*}
\]
OPS5 includes two conflict resolution strategies: LEX and MEA. In each strategy, multiple filters are arranged in a cascade.

✓ **LEX** (default strategy)

The term 'LEX' is based on the fact, that the *lexicographical order* order plays an important role in the second filter.

LEX is primarily suitable for small rule-sets and for problems where the order in which the subtasks are solved is not important.

✓ **MEA**

The term ‘MEA’ should hint at the problem solving method ‘Means Ends Analysis’.

The *prioritization of the first single constraint* is used to control the system. Thereby, this constraint describes the *current subtask*. 
OPS: The Conflict Resolution Strategies LEX and MEA
OPS: The Conflict Resolution Strategies LEX and MEA

- arbitrariness
- specificity
- novelty of instantiation
- novelty of the working memory elements which meet the first single constraint
- uniqueness

LEX   MEA
Conflict Resolution Strategies in the OPS5 Rule Interpreter

✓ Uniqueness

All instantiations are filtered out which **already fired** but still remained in the conflict set because their conditions were still fulfilled by the elements of the working memory.

The underlying heuristic for this filter: **firing the same instantiation once again will bring nothing new.**

✓ Novelty

This second filter underlies a **dwell time strategy** in which instantiations based on the newest data elements are preferred.

The **time stamps** of an instantiation are arranged in descending order and can be understood as a **vector** according to their size.

In this order, the **biggest vector** belongs to the instantiation that is able to pass this filter.
Conflict Resolution Strategies in the OPS5 Rule Interpreter

✓ Novelty

It is possible that a few instantiations pass the filter because several instantiations can have the same vector of time stamps.

If a vector is included in another vector, the longer vector is seen as the bigger one in this order. In doing so, besides a dwell time strategy, a special case strategy is also realized.

Example:

**Input** on this filter level:

<table>
<thead>
<tr>
<th>Rule</th>
<th>Value 1</th>
<th>Value 2</th>
<th>Value 3</th>
<th>Value 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rule_3</td>
<td>27</td>
<td>19</td>
<td>11</td>
<td>6</td>
</tr>
<tr>
<td>Rule_1</td>
<td>27</td>
<td>19</td>
<td>11</td>
<td>6</td>
</tr>
<tr>
<td>Rule_2</td>
<td>27</td>
<td>19</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>Rule_7</td>
<td>27</td>
<td>19</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>Rule_2</td>
<td>21</td>
<td>19</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>Rule_4</td>
<td>17</td>
<td>12</td>
<td>13</td>
<td>6</td>
</tr>
</tbody>
</table>

**Output** of the filter:

<table>
<thead>
<tr>
<th>Rule</th>
<th>Value 1</th>
<th>Value 2</th>
<th>Value 3</th>
<th>Value 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rule_3</td>
<td>27</td>
<td>19</td>
<td>11</td>
<td>6</td>
</tr>
<tr>
<td>Rule_1</td>
<td>27</td>
<td>19</td>
<td>11</td>
<td>6</td>
</tr>
</tbody>
</table>
Conflict Resolution Strategies in the OPS5 Rule Interpreter

✓ Specificity

All instantiations leaving the novelty filter are build with the same elements from the working memory. The third filter that the instantiations have to pass is another special case strategy.

The number of tests for variables and constants of a condition part is used as the parameter for the specificity.

What does count as a test?

- each constant (that has a predicate)
- each selection of a constant
- each comparison with a bounded variable

The rules instantiation with the maximum number of tests passes the specificity filter.

✓ Arbitrariness

Finally, an arbitrary (but not randomly) decision for exactly one instantiation is made, so that a reproducible and unambiguous choice can be made.
LEX-Algorithm

1. Remove all instantiations that have already fired.

2. Order the remaining instantiations by the novelty of their bounded WM elements and choose the biggest instantiation with respect to that order.

3. If there are some instantiations with the same size, order these instantiations by the number of tests and choose the biggest instantiation with respect to that order.

4. If there are also some instantiations with the same size, choose one of the biggest instantiations arbitrarily.
The RETE Algorithm in OPS5

A naive implementation of the recognize-act cycle in which for each loop iteration all database elements are compared with the conditions of all rules, would lead to bad runtime performance, especially for large rule sets.

Two fundamental ideas to improve efficiency:

1) Limitation of database elements that have to be checked:

   During the evaluation of a rule, only a small part of the working memory is changed. A pattern matching with unchanged elements doesn`t change the conflict set. The results of a comparison can be saved. In the next cycle, only the changes in the database are checked whether they lead to instantiations of new rules or whether they remove existing instantiations.

2) Limitation of conditions that have to be checked:

   In general, the condition parts of rules are not disjunct. So, the costs of the recognition cycle can be reduced by executing identical tests only once.