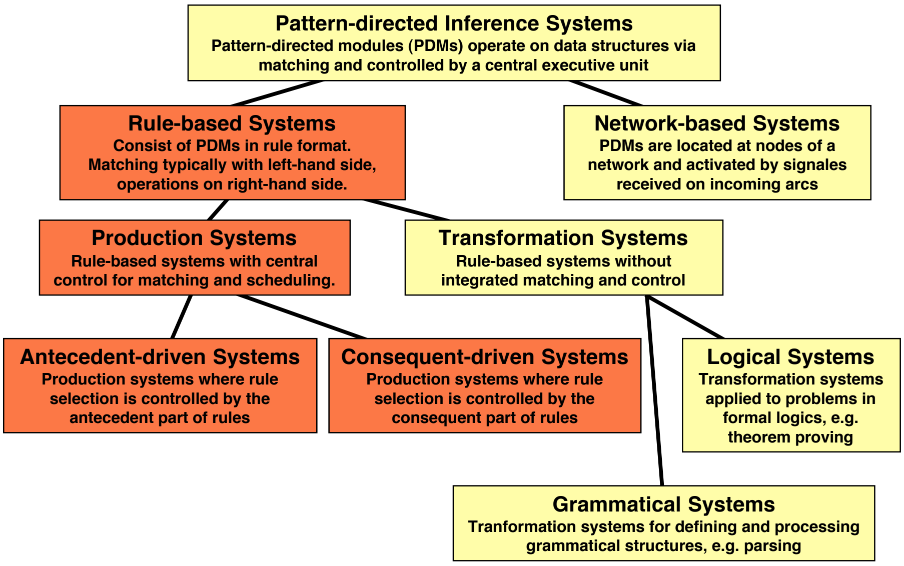
Pattern-directed Inference Systems



1

Rules for Knowledge Representation

Rule-based knowledge representation is useful for specifying inference steps in a declarative way.

Example (scene interpretation):

If	(region.color = green) and (region.location = picture-bottom)
then	(region.type = grass)

Rules may express different types of reasoning:

premise	→	conclusion	*logical implication*
antecedence	→	consequence	*inference from given preconditions*
evidence	→	hypothesis	*interpretation of facts*
situation	→	action	*situated behaviour*
IF	→	THEN	*informal paraphrase*
left-side	→	right-side	*can mean anything*

Rules typically refer to a frame-based knowledge base.

2

Rule-based Programming Language

Experimental AI programming language PLANNER (Hewitt, 1972)

Rules have format:

> **IF GIVEN <extensional data> THEN CONCLUDE <intensional data>**
> **IF WANTED <intensional data> THEN FIND <extensional data>**

Data	Consequent Theorems
(THASSERT (IN BIRD CAGE)) (THASSERT (IN TABLE ROOM)) (THASSERT (IN CHAIR ROOM)) (THASSERT (IN FLOWER VASE)) (THASSERT (ON CAGE TABLE)) (THASSERT (ON VASE TABLE))	(THCONSE (X Y Z) (IN ?X ?Y) (THGOAL (IN !X ?Z) (THUSE NIL)) (THGOAL (IN !Z !Y)) (THCONSE (X Y Z) (IN ?X ?Y) (THGOAL (ON !X ?Z)) (THGOAL (IN !Z !Y))

Query	
(THGOAL (IN FLOWER ROOM))	The intensional data (IN FLOWER ROOM) is derived with the help of consequent theorems

3

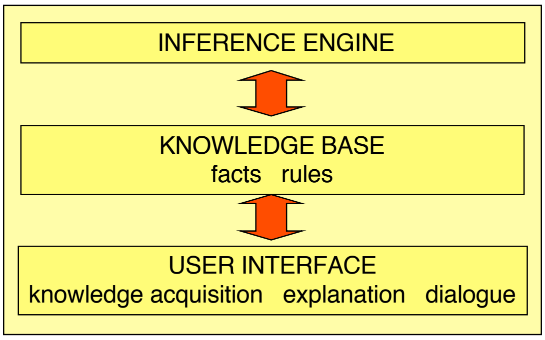
Rule-based Expert Systems

Developed 1970 - 1985 to

- **collect and preserve expert knowledge**
- **replace human experts by computer programs**
- **to automatically derive interesting knowledge.**

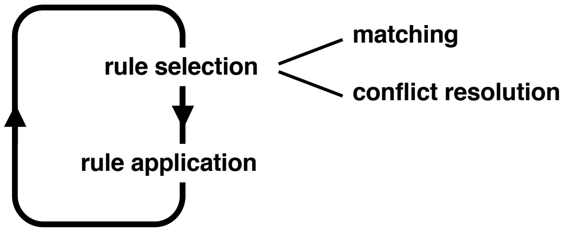
Basic idea: Represent expert knowledge in terms of IF-THEN rules

Basic structure:



4

Recognize-and-act Cycle



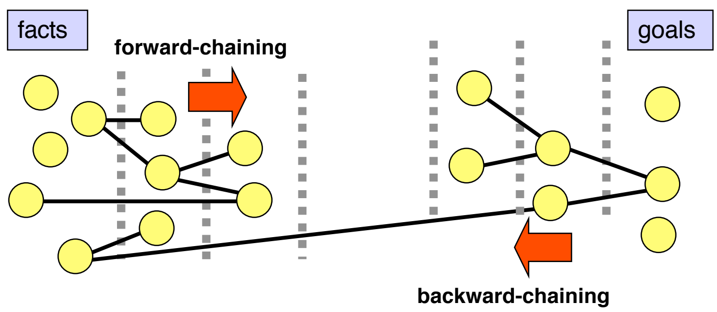
Determine applicable rules by *matching* the antecedent part (in case of forward-chaining) or the consequent part (in case of backward chaining) with data objects.

If more than one rule is applicable, invoke *conflict resolution* to select rule.

5

Forward and Backward Chaining

Rule systems may support forward and/or backward inferencing



6

Processing Steps of Recognize-and act Cycle

Forward Chaining:

Repeat until all goals have been derived:

- Determine rules which can be applied based on available facts
- Select one of those rules
- Apply rule, establish new facts

Backward Chaining:

Repeat until all goals have been derived:

- Determine rules which can be used to derive a goal
- Select one of those rules
- Apply rule, establish unsatisfied conditions as new goals

7

Knowledge-based Diagnosis of a Car Problem

INFERENZMASCHINE

REGELN

Regel 1 Wenn (Anlasser arbeitet normal)
dann (Batterie ok)

Regel 2 Wenn (Batterie OK)
und (Wert Tankuhr > 0)
und (Benzinfilter sauber)
dann (Problem = Zündanlage)

Regel 3 Wenn (Batterie OK)
und (Wert Tankuhr > 0)
und (nicht Benzinfilter sauber)
dann (Defekt = Benzinzuleitung)

Regel 4 Wenn (nicht Scheibenwischer OK)
und (nicht Licht OK)
dann (Defekt = Batterie leer)

Regel 5 Wenn (nicht Wert Tankuhr > 0)
dann (Defekt = Tank leer)

Regel 6 Wenn (Problem = Zündanlage)
und (Verteilerdose OK)
dann (Defekt = Zündspule)

FAKTEN

Fakten des aktuellen Problems

Anlasser arbeitet normal
Scheibenwischer OK
Licht OK
Wert Tankuhr > 0
Benzinfilter sauber
Verteilerdose OK

Abgeleitete Fakten

Batterie OK
Problem = Zündanlage
Defekt = Zündspule

8

Rule Selection

The order of execution cannot be completely controlled in a rule system. It is expected that the user abstracts from individual inference steps.

Rules are selected in a recognize-and-act cycle. If more than one rule can be applied, a "conflict resolution" process decides.

Conflict resolution strategies available in a typical rule system:

• prefer old facts (goals)	*breadth-first search*
• prefer new facts (goals)	*depth-first search*
• prefer more special rule	*more special = more conditions*
• prioritize rules	*e.g. by memory order (PROLOG)*
• use meta-rules	*rules about rule selection*

9

Conflict Resolution with Meta-rules (1)

Expert system for chemical spill treatment may have rules:

R1: If spill is sulfuric acid, apply treatment A

R2: If spill is acid, apply treatment B

Forward chaining may generate conflict set {R1, R2}.

Knowledge base may contain following facts:

- **treatment A is expensive, treatment B is cheap.**
- **treatment A is not dangerous, treatment B is dangerous**
- **R1 has been entered by expert Miller, R2 by novice Johnson**

How can rule selection be controlled in a reasonable way?

10

Conflict Resolution with Meta-rules (2)

Meta-rules for conflict resolution:

R3: Prefer rules with less expensive treatment

R4: Prefer rules with less dangerous treatment

R5: Prefer rules entered by experts before rules entered by novices

Rules R4 and R5 recommend: R1 before R2
Rule 3 recommends: R2 before R1

{R3, R4, R5} is a meta-conflict set.

Meta-meta-rule for meta-conflict resolution:

R6: Prefer meta-rules entered by experts before meta-rules entered by novices.

In practical systems, one rarely needs more than 2 meta levels.

11

Conflict Resolution by Prioritizing

<u>Total</u> order:

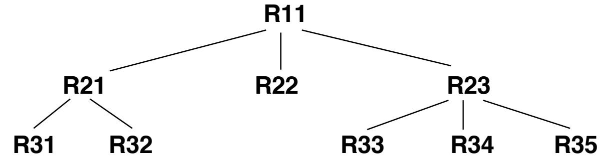
R1 < R2 < ... < RN

Examples:
- **order by storage**
- **order by indexing**

<u>Partial</u> order:

Ri < Rk, Rm < Rn, ...

Example: Rules are structured as a rule tree



12

Conflict Resolution Based on Specialization Relations

Prefer most special rule

1. Compare non-instantiated rules

A rule R1 is more special than R2 if

- R1 has at least as many premises as R2
- each premise in R2 subsumes at least one premise in R1
- R1 and R2 are not identical

Example:

A, B, C, ... attributes
a, b, c, ... constants
X, Y, Z, ... variables

R1: {[A a] [B e] [C X] [D Y] => ...}
R2: {[A X] [B e] [D Y] => ...}

2. Comparison of instantiated rules

Analogous to 1), however no subsumption test for variables required

Conflict Resolution Based on Data Seniority

Data may get time stamp from inference cycle.

- Prioritizing most recent data

  Prefer rules whose instantiation involves recently generated data

  => work on new facts first

- Prioritize least oldest data

  Prefer rules whose instantiation has younger elements than the oldest element of other rules

  => prefer rules which use the youngest facts

- Avoid rule repetition
- Avoid repeated instantiation

The Rule System OPS5

OPS5 ("Official Production System, Version 5")
- developed at CMU 1980 ...
- implementation language for successful expert systems (XCON, XSEL a.o.)

CLIPS
- reimplementation of OPS5 in C for NASA
- freeware

JESS
- reimplementation of OPS5 in Java
- freeware

15

Rules in OPS5

Syntax of a rule in OPS5:

```
<rule>::=            [P <rule-name> <antecedent> --> <consequent>]
<antecedent>::=      {<condition>}
<condition> ::=      <pattern> | - <pattern>
<pattern> ::=        [<object> {^<attribute> <value>}]
<consequent> ::=     {<action>}
<action> ::=         [MAKE <object> {^<attribute> <value>}] |
                     [MODIFY <pattern-number> {^<attribute> <value>}]
                     [REMOVE <pattern-number>] |
                     [WRITE {<value>}]
```

<u>Example</u>: "If there are 2 disks close to each other and with equal size, make them a wheel pair"

```
[P   find-wheel-pair [disk   ^location   <x1>                  ^size   <y>]
                     [disk   ^location   |<x2> - <x1>| < 10    ^size   <y>]    --> ... ]
```

Variable

- depth-first search
- limited expressiveness for constraints

16

RETE Algorithm in OPS5 (1)

A naive Implementation of the recognize-act cycle leads to unacceptably poor runtime performance except for small knowledge bases.

Improving efficiency:

1. **Pattern matching only for a small section of working memory (WM)**

   Rule applications usually lead to small changes of the WM and the conflict set does not change drastically.

=> Store pattern matching results and check only changed WM elements in next recognize-act cycle.

2. **Identical premises occuring in multiple rules must only be evaluated once**

   Premises of different rules often share common conditions,

=> Analyze rules for common premises and optimal order of premise evaluation.

17

RETE Algorithm in OPS5 (2)

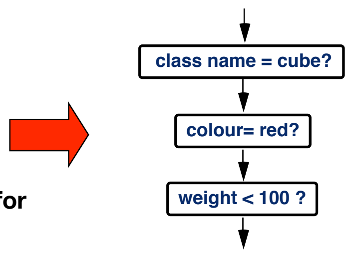
Contruct a net [lat. rete] from the rules which holds code for premise evaluation at its nodes.

Input:
Changes of WM data → net of RETE algorithm → Output:
New conflict set

Net consists of unary nodes coding a condition on a single WM element, and binary nodes for relations between unary nodes.

Example of unary node:

[cube ^colour red ^weight < 100]



A unary node receives WM elements marked for adding or deleting as input, and delivers as output elements which satisfy the conditions.

18

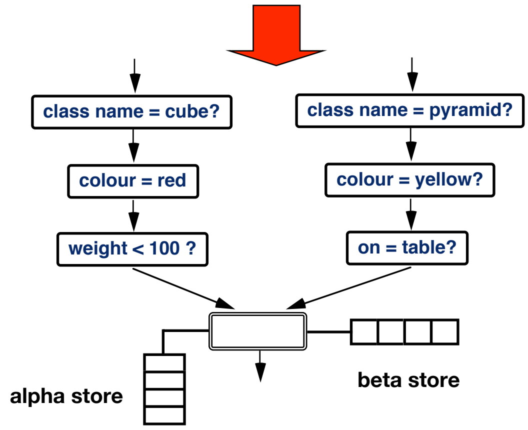
RETE Algorithm in OPS5 (3)

Single element conditions may be combined by binary nodes.

[cube ^colour red ^weight < 100]
[pyramid ^colour yellow ^on table]

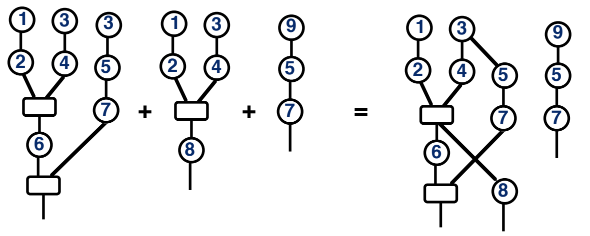
Binary nodes store the WM elements received via the two input lines to generate all possible combinations (cross product).

The output of the last node representing a rule consists of tuples of WM elements satisfying the rule.



RETE Algorithm in OPS5 (4)

Rules may be merged as long as their initial parts conincide.



rule1 rule2 rule3 merged rule1 - rule3

Order of premises influences effectiveness of rule merging.

Example for RETE Algorithm (1)

A rule: (P search-pyramid [cube ^name <cube1> ^on table]
[brick ^weight >200]
[pyramid ^colour <<yellow white>> ^on cube1 ^weight < 200]
-> (action part)

Previous contents of WM:

time stamp	class	name	colour	weight	on
1	cube	C1	blue	250	table
4	cube	C2	red	100	table
6	pyramid	P1	yellow	120	table
9	pyramid	P2	white	NIL	C1
12	brick	B1	blue	300	table

New data entered into WM: [15, brick, B2, blue, 280, NIL]

How does the RETE net compute changes of the conflict set?

21

The Expert Configuration System XCON

XCON has been developed in the early 80´s at CMU using OPS5. The task of XCON was to configure computer systems by Digital Equipment Company. XCON was the first commercially successful expert system.

incomplete component list based on customer wishes → XCON → complete and consistent component list, floor plan for cabinets, slot plan for components

History:

- 1982 start of operations with 1000 rules, 7 min per configuration
- XCON earned money by avoiding configuration errors and delayed customer payment
- 1988 more than 10000 rules:
  - average of 6 conditions per rule
  - average of 5 tests per condition
  - average of 4 actions per rule

22

Typical Rule of XCON

(paraphrase, not in OPS5 rule language)

IF:	THE MOST CURRENT ACTIVE CONTEXT IS ASSIGNING A POWER SUPPLY
	AND AN SBI MODULE OF ANY TYPE HAS BEEN PUT IN A CABINET
	AND THE POSITION IT OCCUPIES IN THE CABINET IS KNOWN
	AND THERE IS SPACE IN THE CABINET FOR A POWER SUPPLY
	AND THERE IS NO AVAILABLE POWER SUPPLY
	AND THE VOLTAGE AND FREQUENCY OF THE COMPONENTS IS KNOWN
THEN:	FIND A POWER SUPPLY OF THAT VOLTAGE AND FREQUENCY
	AND ADD IT TO THE ORDER

Input of XCON

Typical component list based on customer wishes:

COMPONENTS ORDERED:

1	SV-AXMMA-LA	[packaged system]
1	FP780-AA	[floating point accelerator]
1	DW780-AA	[unibus adaptor]
1	BA11-KE	[unibus expansion cabinet box]
6	MS780-DC	[memory]
1	MS780-CA	[memory controller]
1	H9002-HA	[cpu expansion cabinet]
1	H7111-A	[clock battery backup]
1	H7112-A	[memory battery backup]
1	REP05-AA	[single port disk drive]
4	RP05-BA	[dual port disk drive]
1	TEE16-AE	[tape drive with formatter]
2	TE16-AE	[tape drive]
8	RK07-EA	[single port disk drive]
1	DR11-B	[direct memory access interface]
1	LP11-CA	[line printer]
1	DZ11-F	[multiplexer with panel]
1	DZ11-B	[multiplexer]
2	LA36-CE	[hard copy terminal]

Example of a Configuration Run (1)

Numbers correspond to rule applications, lines show context transitions

1. MAJOR-SUBTASK-TRANSITION
2. SET-UP
3. UNBUNDLE-COMPONENTS
53. NOTE-CUSTOMER-GENERATED-EXCEPTION
56. NOTE-UNSUPPORTED-COMPONENTS
57. CHECK-VOLTAGE-AND-FREQUENCY
104. CHECK-FOR-TYPE-OR CLASS-CHANGES
110. VERIFY-SBI-AND-MB-DEVICE-ADEQUACY
111. COUNT-SBI-MODULES-AND-MB-DEVICES
126. GET-NUMBER-OF-BYTES-AND-COUNT-CONTROLLERS
137. FIND-UBA-HBA-CAPACITY-AND-USE
146. VERIFY-MEMORY-ADEQUACY
148. PARTITION-MEMORY
160. ASSIGN-UB-MODULES-EXCEPT-THOSE-CONNECTING-TO-PANELS
177. VERIFY-UB-MODULES-FOR-DEVICES-CONNECTING-TO-PANELS
178. FIND-ATTRIBUTE-OF-TYPE-IN-SYSTEM
180. VERIFY-COMPONENT-OF-SYSTEM
207. NOTE-POSSIBLY-FORGOTTEN-COMPONENTS
213. CHECK-FOR-MISSING-ESSENTIAL-COMPONENTS
215. MAJOR-SUBTASK-TRANSITION
216. DELETE-UNNEEDED-ELEMENTS-FROM-WM
236. FILL-CPU-OR-CPUX-CABINET
240. ADD-UBAS
248. ASSIGN-POWER-SUPPLY

25

Example of a Configuration Run (2)

251. ADD-MBAS
252. DISTRIBUTE-MB-DEVICES
260. ASSIGN-SLAVES-TO-MASTERS
269. ASSIGN-POWER-SUPPLY
272. FILL-MEMORY-SLOTS
278. SHIFT-BOARDS
298. ADD-MEMORY-MODULE-SIMULATORS
306. ASSIGN-POWER-SUPPLY
312. FILL-CPU-SLOTS
318. ASSIGN-POWER-SUPPLY
322. ADD-NECESSARY-SIMULATORS
326. DELETE-TEMPLATES
340. DELETE-UNNEEDED-ELEMENTS-FROM-WM
353. FILL-CPU-OR-CPUX-CABINET
356. ADD-MBAS
359. ASSIGN-POWER-SUPPLY
382. ADD-UBAS
384. FILL-MEMORY-SLOTS
388. SHIFT-BOARDS
389. ADD-MEMORY-MODULES-SIMULATORS
398. ASSIGN-POWER-SUPPLY
399. TERMINATE-SBI
402. ADD-NECESSARY-SIMULATORS
406. DELETE-TEMPLATES
415. MAJOR-SUBTASK-TRANSITION
417. GENERATE-OPTIMAL-SEQUENCE

26

Example of a Configuration Run (3)

```
436.        ASSIGN-UBAS-TO-BOXES-TO-CABINETS
438.            ASSIGN-UBAS-TO-BOXES
441.            ATTRIBUTE-BOXES-AMONG-CABINETS
442.                SET-UP-FOR-BOX-ASSIGNMENTS
446.            ASSIGN-BOXES-TO-CABINETS
452.            COMPUTE-DISTANCES-FROM-UBAS-TO-BOXES
458.        SET-SEQUENCING-MODE
462.    FILL-BOXES
465.        FILL-HALF-BOXES
468.            SELECT-BOX-AND-UB-MODULE-FOR-NEXT-SU
470.            ASSIGN-BACKPLANE-TO-BOX
474.            GENERATE-SLOT-TEMPLATES
478.            PUT-UB-MODULE
482.                LEAVE-BACKPLAN
485.                    AUGMENT-UB-LENGTH
488.                    GET-UB-JUMPER
491.                    CHECK-NEED-FOR-UB-REPEATER
497.            SELECT-BOX-AND-UB-MODULE-FOR-NEXT-SU
501.            ASSIGN-BACKPLANE-TO-BOX
505.            GENERATE-SLOT-TEMPLATES
510.            PUT-UB-MODULE
518.                ADD-SUBOPTIMAL-UB-MODULE
527.                LEAVE-BACKPLANE
540.                    AUGMENT-UB-LENGTH
543.                    GET-UB-JUMPER
547.                    CHECK-NEED-FOR-UB-REPEATER
553.            LEAVE-HALF-BOX
```

27

Example of a Configuration Run (4)

```
559.                CHECK-FOR-UB-JUMPER-CHANGES
561.                CHECK-TERMINATION-CONDITIONS
568.            SELECT-BOX-AND-UB-MODULE-FOR-NEXT-SU
571.            ASSIGN-BLACKPLANE-TO-BOX
576.            GENERATE-SLOT-TEMPLATES
580.            PUT-UB-MODULE
581.                ASSOCIATE-MULTIPLEXER-WITH-PANEL-SLOT
590.                ASSOCIATE-MULTIPLEXER-WITH-PANEL-SLOT
598.                ASSOCIATE-MULTIPLEXER-WITH-PANEL-SLOT
604.                ADD-SUBOPTIMAL-UB-MODULE
608.                ASSOCIATE-MULTIPLEXER-WITH-PANEL-SLOT
615.                ADD-SUBOPTIMAL-UB-MODULE
617.                LEAVE-BACKPLANE
626.                    AUGMENT-UB-LENGTH
629.                    GET-UB-JUMPER
633.                    CHECK-NEED-FOR-UB-REPEATER
643.            LEAVE-HALF-BOX
644.                CHECK-FOR-UB-JUMPER-CHANGES
646.                CHECK-TERMINATION-CONDITIONS
657.            SELECT-BOX-AND-UB-MODULE-FOR-NEXT-SU
660.            ASSIGN-BACKPLANE-TO-BOX
663.            GENERATE-SLOT-TEMPLATES
667.            PUT-UB-MODULE
668.                ASSOCIATE-MULTIPLEXER-WITH-PANEL-SLOT
677.                ASSOCIATE-MULTIPLEXER-WITH-PANEL-SLOT
690.                LEAVE-BACKPLANE
711.                    AUGMENT-UB-LENGTH
```

28

Example of a Configuration Run (5)

714. GET-UB-JUMPER
716. CHECK-NEED-FOR-UB-REPEATER
732. LEAVE-HALF-BOX
733. CHECK-FOR-UB-JUMPER-CHANGES
735. CHECK-TERMINATION-CONDITIONS
738. ASSIGN-UB-JUMPER-CABLES-TO-BOX
749. LEAVE-HALF-BOX
750. CHECK-FOR-UB-JUMPER-CHANGES
752. CHECK-TERMINATION-CONDITIONS
756. ASSIGN-UB-JUMPER-CABLES-TO-BOX
769. ACCEPT-UNIBUS-CONFIGURATION
832. MAJOR-SUBTASK-TRANSITION
833. ASSIGN-TERMINALS-TO-LINES
834. PUT-PANELS-IN-UBX-CABINET
848. MAKE-TERMINAL-ASSIGNMENT
854. MAJOR-SUBTASK-TRANSITION
855. LAY-OUT-SYSTEM
857. FIND-FLOOR-RANKINGS
882. DETERMINE-FLOOR-POSITIONS
888. DETERMINE-FLOOR-POSITIONS-OF-CABINETS
893. DETERMINE-FLOOR-POSITIONS-OF-DEVICES
900. DETERMINE-FLOOR-POSITIONS-OF-SLAVES
908. DETERMINE-FLOOR-POSITIONS-OF-DEVICES
920. DETERMINE-FLOOR-POSITIONS-OF-DEVICES
934. DETERMINE-FLOOR-POSITIONS-OF-DEVICES
942. DETERMINE-FLOOR-POSITIONS-OF-DEVICES
973.

Example of a Configuration Run (6)

974. COMPUTE-CABLE-LENGTHS
1021. FIND-LENGTHS-OF-CABLES-IN-ORDER
1135. ASSIGN-CABLES
1179. FIND-LENGTHS-OF-CABLES-IN-ORDER
1183. FIND-LENGTHS-OF-CABLES-IN-ORDER
1187. FIND-LENGTHS-OF-CABLES-IN-ORDER
1192. NOTE-POSSIBLY-FORGOTTEN-COMPONENT
1198. GENERATE-COMPONENT-NUMBERS-FOR-CABLES
1248. GENERATE-OUTPUT

The trace shows the complexity of the resulting process.

The context structure has been forced onto the process against the spirit of the data-driven operations of rule-based systems.

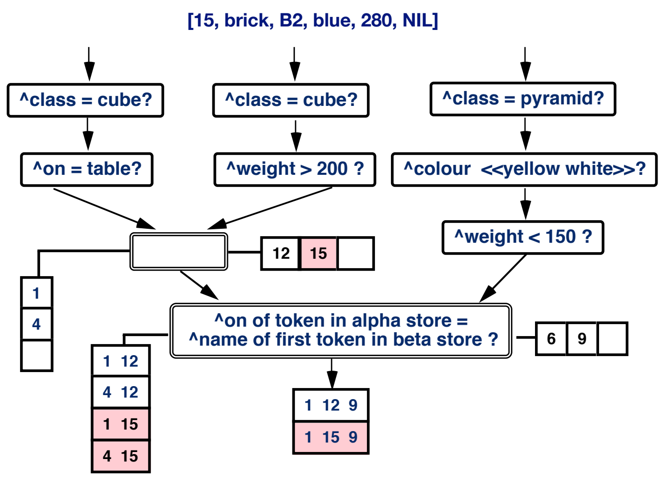
Example of a OPS5 Rule in XSEL

XSEL has been developed 1980 - 1982 by CMU for DEC as a companion system for XCON. The task was to support salespersons acquiring customer wishes.

```
[p  capacity-specified:1:adjust-requirement
;      if memory capacity was ordered on the same line as the system [e.g. system
;       with 4 meg of memory], then assume the user wants the requirement in total
;       and not in addition to what is returned as part of the system, therefore,
;       adjust the requirement
            [context ^status active ^cname capacity-specified]
            [line-item ^status input ^class  memory ^name nil ^units kilobytes
                        ^kilobytes {<required> >0 } ^token <token>]
            [line-item ^status pending ^class system ^parse-token <token>]
            [bus-node ^class memory ^name <device> ^ordered <count>]
       -    [local ^information count-memory-capacity ^source <device>]
            [component ^status reference ^name <device> ^number-of-kilobytes <kb>]
       -->
            [bind <ordered> [compute <quantity> * <kb>]]
            [bind <difference> [compute <required> - <ordered>]]
            [remove 3]
            [modify  2  ^kilobytes <difference>]
            [make local ^type temporary ^context capacity-specified
                        ^information count-memory-capacity ^source <device>]]
```

31

Example for RETE Algorithm (2)



32