Settlement Analysis Expert (SAX) -- Modeling Complex Business Logic In The Development Of Enterprise Solutions

John C. Ownby

Frito-Lay, Inc.
7701 Legacy Drive
Plano, Texas 75024

Abstract:
SAX is a diagnostic system designed to search for inventory related errors in a large transaction base, propose solutions for correcting the errors, and extrapolate the identified errors into patterns of behavior. SAX uses a modular combination of backward-chaining rules and mathematical algorithms to replicate domain knowledge. The system was developed in two phases, with the error seeking phase deployed in 1990 and the pattern recognition phase in 1992. Today, the complete system is in production and provides expert diagnostics to 13,000 salespersons, 1,000 sales managers, and 60 clerical accounting employees.

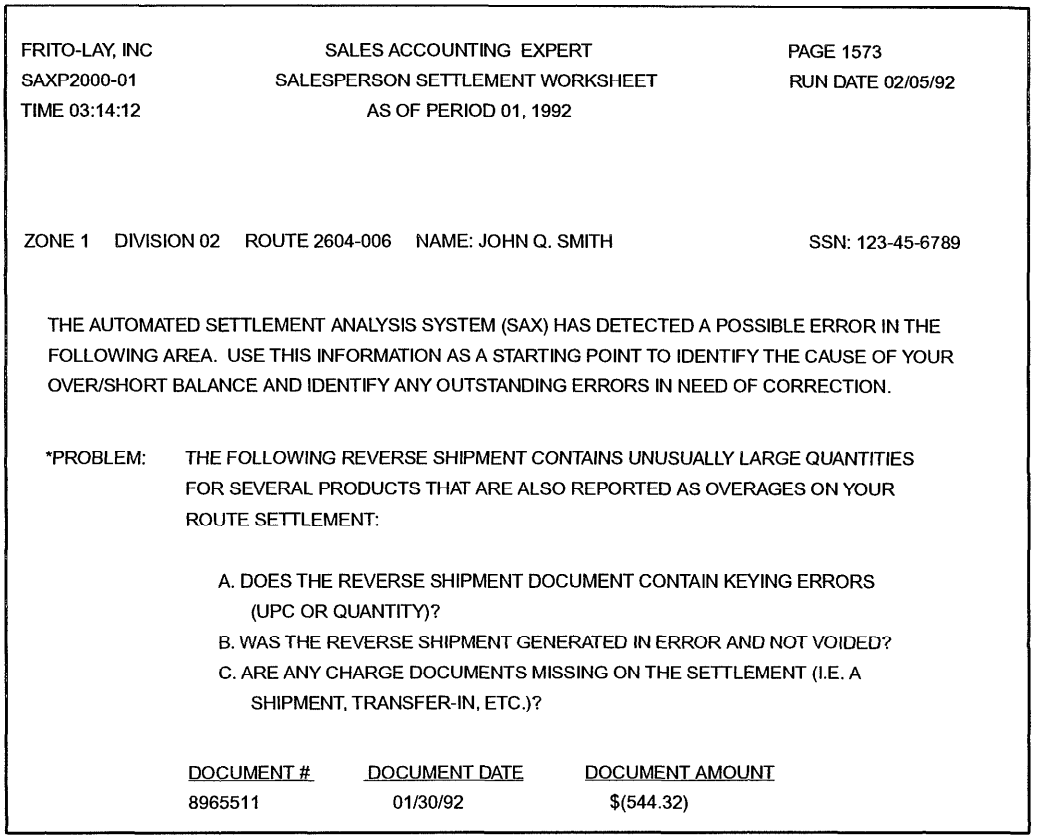
Background:
Frito-Lay's sales force consists of approximately 13,000 route salespersons, with each salesperson responsible for ordering, managing, and selling an inventory of various snack products. Salespersons order and sell their products using a hand-held computer and telecommunicate their transactions to a central host each evening. These transactions are captured in an accounting system, which computes each salesperson's book inventory. Every four weeks each salesperson performs a physical inventory of all the products in their possession, and the accounting system compares each salesperson's physical inventory to their book inventory and identifies any overage or shortage condition.

Because of the high-volume nature of these routes, each salesperson can easily generate 100-200 separate transactions during a four-week time period, with each transaction encompassing any combination of up to 200 different products. Prior to the development of SAX these transactions were manually reviewed in an attempt to identify and correct errors causing overages and shortages. This activity was performed every four weeks by each of the 13,000 salespersons, 1,000 sales managers, and 60 clerical accounting employees, and was both complicated and time consuming.

Diagnostic Module:
SAX was developed in two phases. The diagnostic module (SAX-I) was designed to find errors within the transactions and identify a corrective action for the salesperson. The knowledge base was developed using a mainframe expert system shell and was built to replicate the expertise of a single individual who had fifteen years of domain experience. A rule-based system appeared to be the most natural approach, particularly since the expert was a very willing member of the team. The expert's knowledge was captured in a rule base containing 64 "chunks" of knowledge, expressed in 500 rules and applied against 1.7 million records of data every four weeks. SAX-I attempts to identify missing transactions, transactions containing errors, and transactions that indicate certain performance issues that need to be corrected.

The absence of a beginning inventory, for example, could indicate an error. The fact that this transaction is missing, however, could also indicate that the particular route is new, or that the route is a unique type of route that normally would not have an inventory of products on hand. So, when SAX-I detects a route that has a missing transaction, it then invokes the necessary logic to determine whether or not the route should have had such a transaction.

An example of an erroneous transaction would be a transaction that is a valid transaction for the route, but an error was made in the transaction's detail, such as an incorrect quantity or product code. When such an error is found, SAX-I searches the necessary transaction detail of potentially all other transactions on the route (and in some cases includes other routes' activity in its evaluation) and makes inferences based on relationships identified within the data. For example, SAX-I could identify one product code on a shipment transaction that contains a quantity that appears to be driving the over/short on that item for the entire route. If an error is made on a transaction between routes, SAX-I also checks the relationships on the other route to increase the certainty of it's inference.



FRITO-LAY, INC SALES ACCOUNTING EXPERT PAGE 1573
SAXP2000-01 SALESPERSON SETTLEMENT WORKSHEET RUN DATE 02/05/92
TIME 03:14:12 AS OF PERIOD 01, 1992

ZONE 1 DIVISION 02 ROUTE 2604-006 NAME: JOHN Q. SMITH SSN: 123-45-6789

THE AUTOMATED SETTLEMENT ANALYSIS SYSTEM (SAX) HAS DETECTED A POSSIBLE ERROR IN THE FOLLOWING AREA. USE THIS INFORMATION AS A STARTING POINT TO IDENTIFY THE CAUSE OF YOUR OVER/SHORT BALANCE AND IDENTIFY ANY OUTSTANDING ERRORS IN NEED OF CORRECTION.

*PROBLEM: THE FOLLOWING REVERSE SHIPMENT CONTAINS UNUSUALLY LARGE QUANTITIES FOR SEVERAL PRODUCTS THAT ARE ALSO REPORTED AS OVERAGES ON YOUR ROUTE SETTLEMENT:

A. DOES THE REVERSE SHIPMENT DOCUMENT CONTAIN KEYING ERRORS (UPC OR QUANTITY)?
B. WAS THE REVERSE SHIPMENT GENERATED IN ERROR AND NOT VOIDED?
C. ARE ANY CHARGE DOCUMENTS MISSING ON THE SETTLEMENT (I.E. A SHIPMENT, TRANSFER-IN, ETC.)?

DOCUMENT #	DOCUMENT DATE	DOCUMENT AMOUNT
8965511	01/30/92	$(544.32)

FIGURE 1: SAX-1 OUTPUT IDENTIFIES SPECIFIC ERRORS & ISSUES WITHIN THE TRANSACTION BASE EVERY 4 WEEKS

In addition to finding missing and erroneous transactions, SAX-I also looks for and identifies certain performance issues. If SAX-I identifies a route that has an unusually high amount of stales, for example, it makes an exhaustive attempt to find a reason. If it can detect a probable cause for the high stales, it reports its findings to the route salesperson. If it cannot find a reason, it simply reports that a large number of stales occurred on the route.

Sax-I's logic is arranged hierarchically, in that it first attempts to prove a base hypothesis (i.e. there is a problem with high stales), then proceeds to refine the hypothesis based on all available pertinent facts (i.e. the high stales could be caused by stales occurring on the truck rather than in the store, or by mis-classification of another transaction as a stales transaction, or could be due to a keying error, etc.). For each hypothesis attempted, SAX-I remembers the last rule successfully fired, and formats the appropriate output text for reporting (***Figure 1***). Thus, SAX-I informs the salesperson in as much detail as possible the nature of the problem identified, as well as the appropriate corrective action.

Pattern Identification Module:

While SAX-I identifies transaction-related problems occurring within each four-week time period, the pattern identification module of SAX (SAX-II) is designed to identify patterns of behavior over a larger period of time. For example, certain patterns of errors can indicate that a salesperson needs training, or that write-off exposure exists, and in some cases can mean that a district-wide problem needs to be addressed.

SAX-II takes the output of SAX-I over a four month time frame, links it with other selected data elements, and employs a process of categorizing, scoring, selection, and analyses using hierarchical rules. Using 120 Level 1 rules (lowest level in the hierarchy), SAX-II categorizes

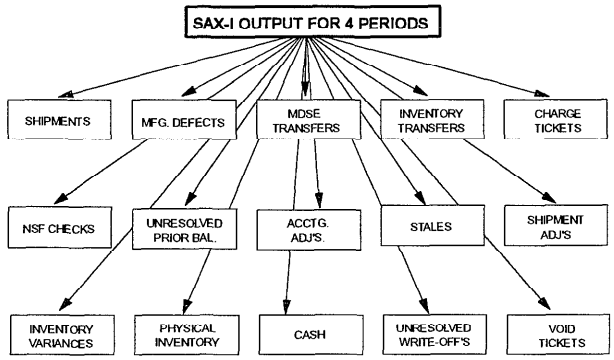


FIGURE 2: SAX-II CATEGORIES

the SAX-I rule firings as shown in ***Figure 2***. This categorization allows SAX-II to view the SAX-I output in terms of generic issues over time. In viewing shipment performance, for example, SAX-I can find any or all of the problems shown in ***Figure 3***.

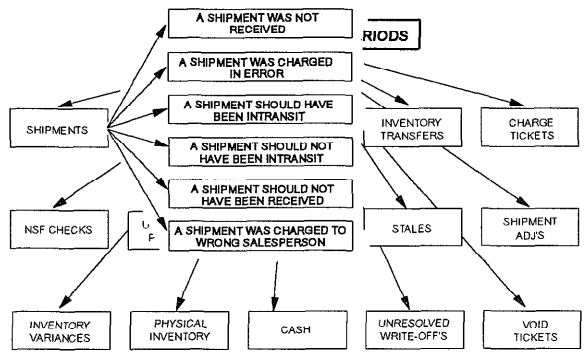


FIGURE 3: EACH CATEGORY CAN CONTAIN MULTIPLE ISSUES

Each of these error situations has a unique cause and subsequent corrective action that SAX-I addresses. For purposes of identifying patterns of behavior, however, SAX-II only needs to see that the salesperson has some form of recurring issue with processing shipments of merchandise. Therefore, SAX-II categorizes all shipment errors into one category.

After categorizing the SAX-I output, SAX-II then goes through a process of identifying and scoring frequency patterns found within each category. The frequency patterns are based on the categorized SAX-I rule firings, and assigned scores based on pre-defined patterns (***Figure 4***).

SCORE ASSIGNED	FREQUENCY PATTERN	EXPLANATION
1	CP	CURRENT PERIOD ONLY
2	CP+1P	CUR. PD. & IMMEDIATE PRIOR PD.
3	CP+2	CUR PD. & ANY 2 PRIOR PD'S.
4	CP+2P	CUR PD & 2 IMMEDIATE PRIOR PD'S.
5	CP+3	CUR PD & 3 PRIOR PERIODS

FIGURE 4: SAX-I FREQUENCY PATTERNS

After scoring, all categories with a defined frequency pattern will be selected for any salesperson with either a total score greater than a pre-defined threshold, or any salesperson that has one of five "special" conditions. These "special" conditions are patterns found that indicate further analyses and reporting is required, regardless of the salesperson's total score. For example, if a salesperson does not submit a physical inventory for two periods in a row, this lack of data could prevent other SAX-I rules from firing. Therefore, SAX-II would consider such a pattern as serious, regardless of the salesperson's total score, and select the route for analyses.

After scoring and selection is completed using SAX-II's Level I rule base, the output from Level 1 becomes the input for 400 Level II rules. The Level II rule base attempts to further refine the trends and patterns identified in Level I and, using a hierarchical structure, attempts to identify meaningful relationships using a combination of Level I rule firings and other selected raw data elements. For example, assume the Level I rule base fired the rules noted in ***Figure 5***.

The Level II rules take the inferences created in the Level I rules, and through a higher level of reasoning create new inferences. In this example, for instance, the Level II rules would determine that Rules 1270 & 1280 are related (Rule 1280 is actually identified the cause of the problem identified by Rule 1270), Rules 1350 & 1460 are related (the salesperson's outstanding NSF's are due to a customer problem), and Rules 1375, 1377, 1378 & 1380 are related (the salesperson has a growing shortage pattern, and no payroll action is being taken to reduce it).

From these Level II rules a new inference would be created indicating that the salesperson's growing shortage is probably being driven by customer NSF checks, and

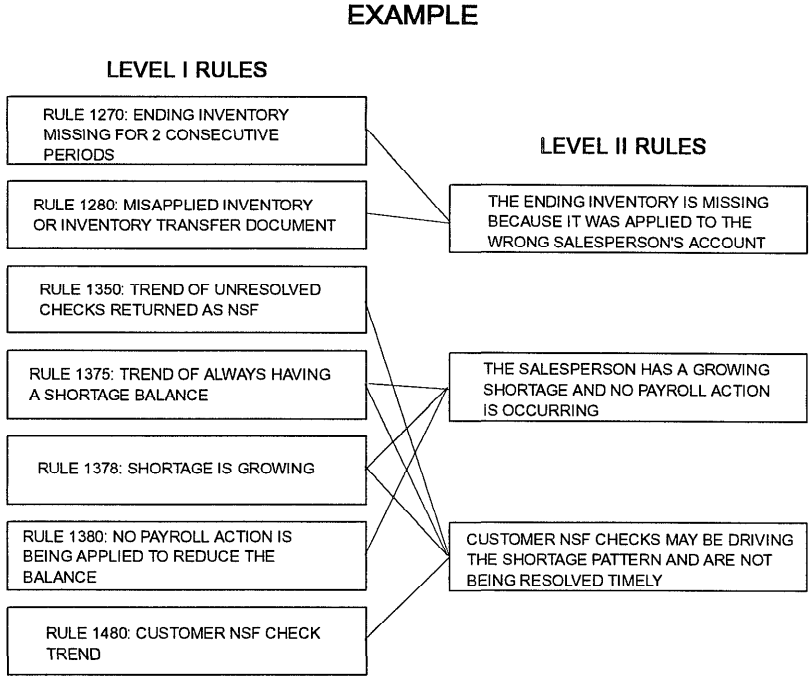


FIGURE 5: LEVEL II INFERENCES DRAWN FROM LEVEL-I RULE FIRINGS

that although no inventory related issues were detected, the misapplication of the salesperson's inventory could be keeping such issues from surfacing -- the same conclusion that an overworked district manager could have reached, but only after pouring over boxes of sales tickets and shipment invoices for many days.

The Level II rules are linked to a text file, allowing each selected salesperson's patterns to be formatted and distributed to field sales managers (***Figure 6***). SAX-II eliminates much of the detailed research efforts required by providing the field manager with a concise summary of balance related behavioral issues that need to be addressed.

Validation:

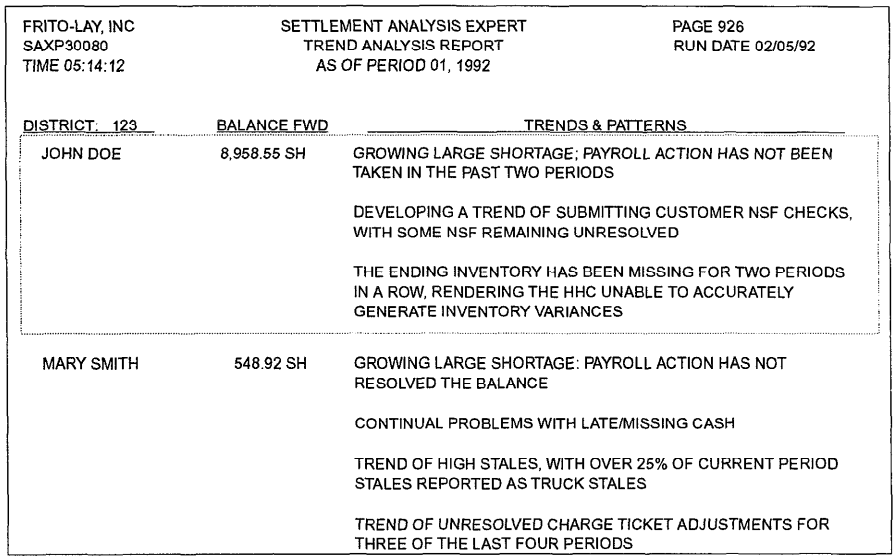
The knowledge base was validated against live data. Each day, as new rules were added, a nightly cycle was run against live data. After successful validation by the expert resource, the systems output was then submitted to one-half of a test team provided by the user group. This half of the test team was charged with proving the output right or wrong.

The other half of the test team was not given the systems output, but was charged with working from the data to develop conclusions. These conclusions were then validated against the conclusions made by the system. This method of testing from the conclusions backward and from the data forward enabled the project team to significantly refine the logic used by the system.

Deployment:

SAX-I was initially developed using a mainframe expert system shell. Because of the vast amounts of input data required from other systems (approximately 1.7 million records every four weeks), the SAX-I rule base was translated into procedural code prior to deployment to shorten cycle times. Total development time (including knowledge acquisition, prototype development, testing, and preparation for production) was seven months, and staffed with a full time project team consisting of one knowledge engineer, two systems analysts, and one domain expert. The system was placed into production in August, 1990.

SAX-II was developed and deployed in a similar fashion. Total development time was 11 months, and was staffed with two knowledge engineers and one system analyst. The system was placed into production in January, 1992.



FRITO-LAY, INC
SAXP30080
TIME 05:14:12

SETTLEMENT ANALYSIS EXPERT
TREND ANALYSIS REPORT
AS OF PERIOD 01, 1992

PAGE 926
RUN DATE 02/05/92

DISTRICT: 123	BALANCE FWD	TRENDS & PATTERNS
JOHN DOE	8,958.55 SH	GROWING LARGE SHORTAGE; PAYROLL ACTION HAS NOT BEEN TAKEN IN THE PAST TWO PERIODS DEVELOPING A TREND OF SUBMITTING CUSTOMER NSF CHECKS, WITH SOME NSF REMAINING UNRESOLVED THE ENDING INVENTORY HAS BEEN MISSING FOR TWO PERIODS IN A ROW, RENDERING THE HHC UNABLE TO ACCURATELY GENERATE INVENTORY VARIANCES
MARY SMITH	548.92 SH	GROWING LARGE SHORTAGE: PAYROLL ACTION HAS NOT RESOLVED THE BALANCE CONTINUAL PROBLEMS WITH LATE/MISSING CASH TREND OF HIGH STALES, WITH OVER 25% OF CURRENT PERIOD STALES REPORTED AS TRUCK STALES TREND OF UNRESOLVED CHARGE TICKET ADJUSTMENTS FOR THREE OF THE LAST FOUR PERIODS

FIGURE 6: EXAMPLE OF SAX-II OUTPUT

Although the systems were deployed using procedural code (to shorten the cycle times), the shell used in development was an essential tool in developing, testing, and refining the complex logic used by SAX in analyzing the transaction base.

Both systems have been in continuous production since their initial deployment, with the systems' output currently distributed to over 13,000 field salespersons and sales management personnel and 60 headquarters accounting personnel.

Is SAX Really AI?:

SAX-I was our first attempt at building an intelligent system, and we began the task with a somewhat "purist" approach, in that we intended to build the entire system within the expert system shell. Because of SAX-I's huge appetite for raw data (which is why we chose this task in the first place), reasoning over 1.7 million records within the shell was agonizingly slow, the cycle averaging roughly six CPU hours.

Rather than continue dimming the lights in the data center each time we ran SAX-I, we made a decision to experiment with pre-processing some of the incoming data elements by placing some of the easier rules into a module residing between the data and the shell. This was so successful that within a few months we made the decision to re-code the entire rule base into procedural code for the production system, and shortened the cycle time by 83%.

SAX-II was built and deployed using a similar approach. Even though the shell was not used in the final version of the production system, the use of knowledge based tools was critical in the development phase.

So, is SAX really an AI application? Our assessment after building, deploying, and living with both systems has led us to conclude that the AI in a system is not necessarily dependent upon the vehicle in which a system is developed or deployed, but is defined by the task the system performs. In the case of SAX-I & II, the systems replicate a highly complex reasoning process ***(Figure 7)*** that ultimately utilizes over 8 million records of raw data and answers the question of *"What does the data mean?,"* and *"What do you need to do about it?."* The AI in SAX is in the knowledge captured during the development phase and replicated by the systems' rules.

My guess is that over time the distinction between traditional systems and AI will become increasingly blurred. AI will become a widely accepted technique, and as AI techniques are embedded into traditional systems (using hopefully a variety of tools), AI will become more and more a part of mainstream system design.

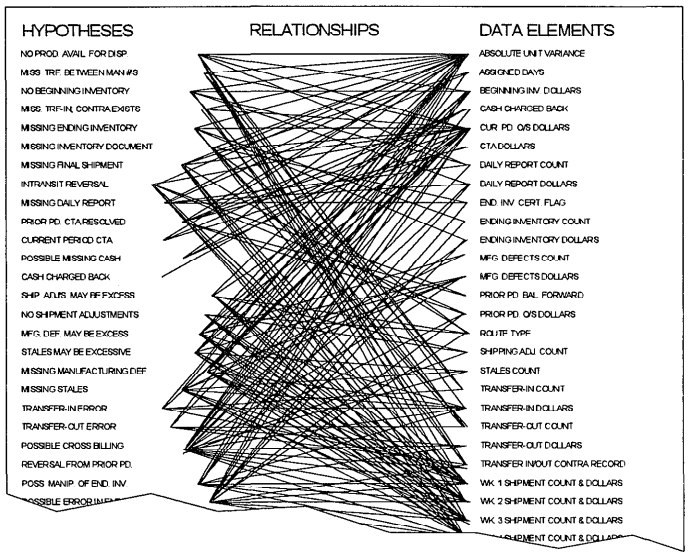


FIGURE 7: SAX REPLICATES A HIGHLY COMPLEX REASONING PROCESS

Innovative Aspects Of SAX:

The main innovative aspect of SAX-I & II comes from the complexity of the task performed ***(Figure 8)***. Working together, the two system modules efficiently turn over 8 million records of raw transactional data (sourced from a variety of systems) into roughly 6,000 concise and actionable conclusions. The input for SAX-I is raw data; the output is knowledge. The input for SAX-II is the *output of SAX-I*. Thus, the output for one expert system (SAX-I) becomes the input for another (SAX-II).

Maintenance:

As soon as we realized procedural code would be necessary to reduce cycle times, we took careful steps to ensure that SAX I & II would be maintainable. Variables, for example, that were shared by multiple rules were placed in user-maintainable tables (i.e. data aggregations

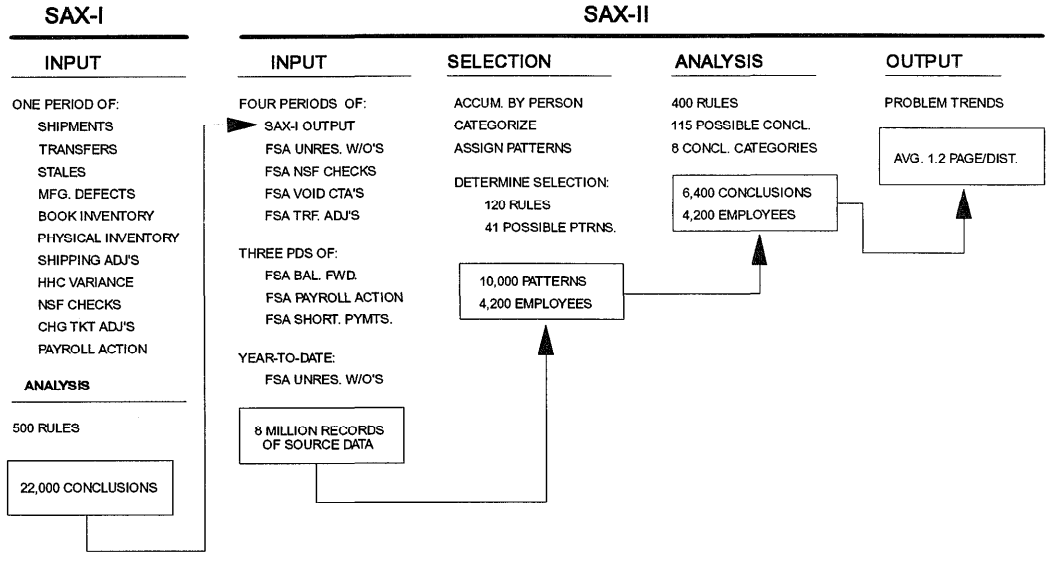


FIGURE 8: SAX TAKES 8 MILLION RECORDS OF RAW DATA AND TURNS IT INTO 6,000 CONCLUSIONS

that would be used repeatedly); algorithms were developed and re-used that enabled us to use fuzzy logic techniques in identifying transactions, such as mathematical relationships used to identify different data elements that "roughly offset" (a classic case of *"how tall is tall"*), and others that enabled SAX to determine the uniqueness of similar transactions.

Other algorithms were developed to allow for categorizing route types, dates, transaction types, etc. The definitions of these formulas were placed in a separate table and referenced by the system when needed to enable major reusable definitions to be maintained from a central system point. Thus far, maintaining the systems has not been difficult, however because the internal structure of the knowledge base is hierarchical, as our business continues to grow and become increasingly complex, we have to be vigilant as new logic is added over time. Should the business undergo drastic changes, we would probably have to consider re-development using the shell.

Learning's:

This system, particularly since it was our first attempt at replicating a difficult thought process, was a tremendous educational process. Some of our key learning's were:

- Traditional systems are usually designed two-dimensionally, in that most systems' ultimate output is expressed in some form of rows and columns, with summaries at various levels. Most business systems today (assuming they were well designed) were built with a high level of data integration. On the output side, however, a surprisingly high percentage of these systems resemble islands -- they may share raw data as inputs, but on the output side they are like strangers. Rarely does hindsight ever conclude that a low level of data integration within a highly integrated business process was a good idea -- the same is true of system outputs.

- As a business becomes increasingly "data rich," this overwhelming amount of data can cause it to simultaneously become "information poor" if the data is not efficiently converted into information. AI allows you to view your data multi-dimensionally, with an ultimate goal of identifying meaningful relationships. Therefore, AI opens the possibility of turning data into information by taking an integrated view of system outputs.

- Never underestimate the value of a willing domain expert. SAX-I was built to replicate a single person's knowledge, and not only was the expert willing, but was excited to play such an important role. SAX-II was more difficult, in that it represents a synthesis of several experts.

- Never underestimate human skepticism. As a general rule the average person accepts the concept of automated reasoning about as easy as they once accepted automated bank tellers. Focus on pre-selling and training before you deploy your system. We rolled the first system with little fanfare and subsequently had to spend a lot of effort getting people to understand and accept the power behind the system.

- Use every tool in your toolbox appropriately. Although the shell was invaluable in the development stage of the project, attempting to place the shell into production created lengthy cycle times. When we faced huge cycle times in the early stages of SAX-I, we looked at some fairly outrageous options before deciding upon using procedural code to replicate the shell's rule base. Our findings have prompted us to perhaps add a second set of conditions to the Turing Test: *"If a person in the next room can't tell what it's coded in, then ....."*

- Where transaction processing systems allowed us to eliminate the need for huge rooms filled with people punching numbers into calculators, AI can ultimately automate much of today's analytical tasks. Look around -- potential applications for automating human expertise are everywhere.

Acknowledgments:

SAX-I and SAX-II were the results of a number of highly talented and committed people. Mary McNeese provided critical domain expertise to SAX-I, and became so enthused with the possibilities she led the effort to build SAX-II. Audrey Holman not only came up with the solution to utilize procedural code, but spent many creative months figuring out how to replicate complex rules. Steve Shavel gave us a taste for where we could take these systems early on via his rapid prototyping expertise. Also, I would like to thank the numerous salespersons and district sales managers that provided valuable input to the team.