PROGRESS® Corticon®

WHITE PAPER

www.progress.com

PROGRESS® CORTICON® NEXT-GENERATION BUSINESS RULES

Exploring Model-driven BRMS and the DeTI Engine

PROGRESS

TABLE OF CONTENTS

INTRODUCTION

The primary objective of this document is to educate decision makers on the evolution and current state-of-the-art in the business rules industry, as well as the critical success factors for employing business rules technologies in business automation initiatives. The focus of discussion will be a comparison of two fundamentally different approaches to rule authoring and rules execution. It is not intended to be a comprehensive review of the Progress® Corticon® product line.

Because the business rules industry has for years been largely influenced by the first vendors in the industry, who embraced the same common technology and algorithms, erroneous perceptions have settled into the collective psyche. Having personally suffered through the shortcomings of legacy business rules offerings, Progress Corticon founders developed a breakthrough solution that addresses the real needs of decision automation and fulfills the order-of-magnitude improvements in automation project timelines, agility, and business control that have been the promise and potential of business rules.

After a brief primer on business rules, this white paper will discuss characteristics of first-generation business rules management systems (BRMS), with particular emphasis on key issues hindering customer success. This will set the stage for a deep dive into the next-generation Corticon BRMS, tracing its unique model-driven approach and raison d'être (reason for being) back to fundamental customer needs that have been underserved by first-generation products. Finally, this paper will compare the technical differences between rule execution between the Corticon engine and other BRMS vendors' engines, providing insights into performance and scalability.

BUSINESS RULES PRIMER

DEFINITION OF BUSINESS RULES

Let's start with the basics of business rules. Ron Ross, a guru of business rules, provides the following definition:

- A business rule is a discrete operational business policy or practice.
- A business rule may be considered a user requirement that is expressed in a non-procedural and non-technical form.

For additional clarity, we should further explore two of the terms used above: "non-procedural" and "non-technical." To express a business rule in a procedural way would be to define the control flow or exactly how to process the logic. Non-procedural, therefore, is equivalent to declarative, which means that business rules specify what logic is to be applied, not how to execute a set of operations (the control flow and sequencing). As this topic will be covered in greater detail later, suffice it to say that expressing rules declaratively is fundamental to the definition of business rules.

Also critical to the definition of business rules is that they should be expressed in a non-technical format. Business rules should be business-friendly, understandable to business stakeholders, without requiring knowledge of programming or technical languages.

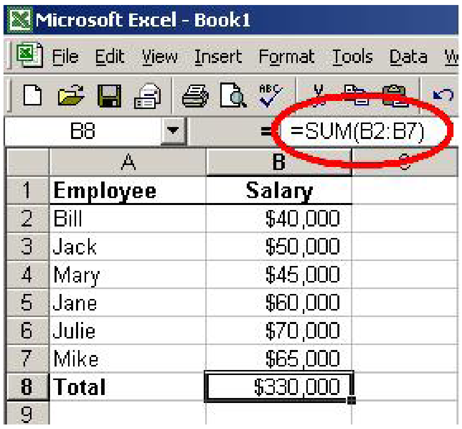
Traditionally, business rules that were automated were coded directly into applications as "business logic." Because business logic is the most volatile aspect of business applications, however, the pains of maintaining rules in application code have become more obvious and significant as the pace of business demands greater agility. The obvious answer is to externalize business rule logic and to manage this logic in a way that is easy-to-build, easy-to-change, and easy-to-integrate (as decision services) while still meeting enterprise performance and scalability demands.

PROCEDURAL VS. DECLARATIVE

Why is "non-procedural" such an important capability in rule authoring? For the simple reason that rule authoring should be a natural exercise for business domain experts, who understand the business policies and underlying business logic that is to be implemented. These business domain experts should be empowered to complete the entire rule authoring process from inception to deployment-ready without relying on others. Rule authoring that employs a procedural language or approach means that rule authors must master a technical programming language and concepts and the complex interactions of rules and how they are processed. Such a requirement often forces rule authoring to be segregated into two phases (requirements capture and rule implementation), often with different individuals owning and participating in each phase and multiple iterations between the two. Rule authoring that involves procedural elements in rule definitions demands the traditional coding methodology of business-driven requirements thrown over the fence to developers who implement in a technical language. Realizing the goal of "non-procedural" enables a very different way to author and manage business rules and to achieve order-of-magnitude better business results.

The distinction between declarative and procedural may be best articulated with a familiar example: the spreadsheet.

The spreadsheet is a good example of a declarative metaphor. Users define financial models using declarative operators—such as "= SUM ()"—and may combine them into sophisticated expressions. The common usage pattern is that the user defines what needs to be done. In the example in Figure 1 below, cell B8 is defined simply as the sum of cells B2 through B7.



	A	B
1	Employee	Salary
2	Bill	$40,000
3	Jack	$50,000
4	Mary	$45,000
5	Jane	$60,000
6	Julie	$70,000
7	Mike	$65,000
8	Total	$330,000
9		

Figure 1
A spreadsheet is an example of a declarative metaphor.

Contrast this with a procedural approach. The procedural approach requires the user to tell the system **how** to execute the function. In the simple example used above, summing a group of values, the user would have to explicitly specify the technical steps required to loop through each value, adding each one to the accumulated total (see Figure 2 below). This is a far cry from empowering business domain experts to author business rules in an intuitive way and for everyone in the organization to understand the business logic that is driving their automated decisions.

```
Integer i
Float tmp

tmp = 0
For i = 2 to 7
        tmp+= B[i]
Next I


B[8] = tmp
```

Figure 2
Procedural equivalent of a simple summation function

INFERENCING

A working definition

Inferencing, simply defined, is the action of inferring a new fact based on known facts. Inferencing is often associated with deductive reasoning, which itself is defined as using deductive arguments to move from given statements (premises) to conclusions, which must be true if the premises are true (source: Wikipedia). A oft-quoted example by Aristotle is as follows:

- All men are mortal. (major premise)
- Socrates is a man. (minor premise)
- Socrates is mortal. (conclusion)

In the context of BRMS, inferencing is generally equated to the practice of rule engines determining the sequence of rule execution based on logical dependencies. Rule engines that support automatic sequencing of rule processing are commonly viewed as "inferencing capable rule engines" and offer a measure of flexibility in rule authoring because, theoretically, users do not need to concern themselves with rule sequencing.

An example of rule logical dependency would be the following:

1. IF a person is a skydiver, THEN the **person is high risk**.
2. IF a **person is high risk**, THEN the person is rejected.

Rule 2 in this example is logically dependent on Rule 1 and must execute after Rule 1.

Types of iteration

Inferencing-capable rule engines typically perform two types of iteration: fact iteration and logical loop iteration (aka "rule recursion"). Fact iteration is the simple concept of processing each rule against all relevant facts in working memory. Logical loop iteration, or rule recursion, happens when there is a cycle in the rule dependency graph. For instance, if Rule 2 depends on Rule 1, and Rule 3 depends on Rule 2, and Rule 1 depends on Rule 3, there is a logical loop. To support logical looping, a rule engine must iterate over the logical loop until there are no more updates in working memory.

For a detailed exploration of how iteration is employed to solve an inferencing use case, see Appendix A.

CATEGORIZATION OF RULE ENGINES

Because the earliest rule engines employed the RETE algorithm, RETE-based engines have often been synonymous with inferencing engines. But to generalize the problem or solution space (inferencing) to one implementation or approach (RETE) would be unwise. In fact, a leading industry analyst has identified three types of rule engines or rule processing algorithms: inferencing, sequential, and extended sequential.

If we look at algorithms more as a class of use cases, we can map engine architectures to one or more of the use case classes and gain a better understanding of what engines are best suited to which classes of use cases. As shown in Figure 3, inferencing algorithms arrived first on the scene and provided a degree of rule authoring flexibility, but were vulnerable to performance bottlenecks. The RETE-based rule engines fit into this classification. In an attempt to address performance issues, most first-generation vendors added a completely separate engine mode that employed sequential algorithms, but this came at a price of authoring flexibility, and/or the types of use cases that could be solved. Forrester has identified a third category of "algorithms" as somewhat of a hybrid and has called this category "extended sequential." From the perspective of the characteristics of this third category, notably rule authoring flexibility, high performance and scalability, and capable of solving inferencing problems, the Progress® Corticon® Deployment-Time Inferencing (DeTI) solution fits into the extended sequential category of engines.

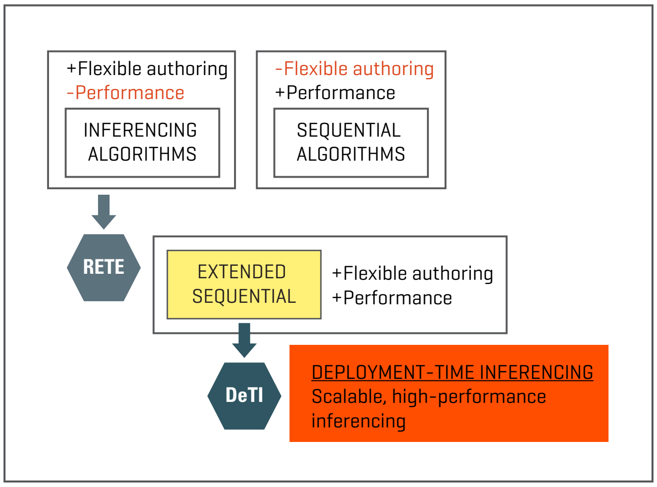


Figure 3
Rule engine classifications

FIRST-GENERATION BUSINESS RULES MANAGEMENT SYSTEMS (BRMS)

First-generation BRMS vendors have played an important role in developing the business rules market. Thanks to them, there is a growing recognition of the power and value of externalizing business rules and managing decisions as critical corporate assets with purposeful business rules management systems. However, first-generation BRMS do not fully realize the potential of business rules.

First-generation BRMS vendors can be characterized as such by their approach to processing rules. These vendors offer a RETE-based approach and by and large have introduced procedural elements to their rule languages. They handle inferencing types of problems by building dependency networks in the engine's working memory at runtime.

DESIGN PRINCIPLES: BUILT FOR DECISION SUPPORT

The core technology embraced by first-generation BRMS vendors, the RETE algorithm, evolved from the expert systems market. This heritage is evident from the lingering design principles originally employed to satisfy expert systems use cases. These use cases may be characterized as "decision support," centering on complex problems, defined by large and rather fluid rule bases, where one or a few experts tease out multiple possible "answers" that contribute to their decision-making process. The underlying technology and related first-generation BRMS architectures are suited to ongoing tinkering of the rule base and use of a technical language for maximum flexibility. But first-generation BRMS are vulnerable to a number of problems including non-deterministic behavior, a lack of integrity in rule sets, and performance bottlenecks due to larger data sets. Figure 4 below summarizes the design priorities and primary focus of the RETE algorithm and, by association, first-generation BRMS vendors.

PRIMARY SEGMENT SERVED	EXPERT SYSTEMS
Consumer of decisions	Expert
Use Case Category	Decision support
Deterministic	Not always
Inconsistencies (integrity)	Tolerated
Message size (data payload)	Small (query)
Workload	Small

Figure 4
Design priorities and characteristics of RETE and first-generation BRMS

SHORTCOMINGS OF FIRST-GENERATION BRMS

At the highest level, there are two categories of shortcomings associated with first-generation BRMS: ease-of-use and performance. First-generation BRMS products are difficult to use because they require most or all of the following:

- Proprietary programming languages
- Complex engine algorithms
- Proprietary rule development infrastructures
- Specialized debugging techniques to increase the chances of getting rules right

First-generation BRMS solutions are also challenged by performance and scalability bottlenecks. The fundamental nature of the RETE approach, in-memory dependency network processing, is taxing on servers. So it should be no surprise that there is an inherent architectural performance and scalability ceiling, which is typically reached when the data to be evaluated against a rule set is large and/or complex. This bottleneck is such a well-known phenomenon, it has a nickname: the "RETE wall." Optimizing RETE-type engine performance is also challenging because it requires extremely deep expertise in the detailed mechanics of a particular vendor's engine (and the logic of the rule sets themselves) to manually tune performance.

Multiple rule languages and the impact on ease-of-use

In the earliest days of the business rules industry, the rule languages in use included OPS5 and CLIPS. These languages were declarative, but they were very difficult to use by both technical and non-technical users.

To make the rule authoring language more appealing to technical users, legacy vendors introduced procedural constructs, violating the objective of keeping rule definition declarative. In addition, these vendors created their own specialized, proprietary syntax and required users to gain a deep knowledge of proprietary engine algorithms.

Having strayed off the path of a non-procedural, non-technical rule authoring solution, legacy vendors added secondary languages in an effort to appeal to non-technical users. These secondary languages often employed natural-language style constructs and may have had good intentions, but they introduce a set of new problems instead of solving the fundamental objective. Secondary languages, or business languages, invariably handle only the simplest business rule logic and often require a supporting proprietary technical infrastructure. In the end, synchronizing rules authored in two languages, by two classes of users, is fraught with problems—and in many ways is a step backward to a traditional coding methodology (business users author requirements; developers implement in technical languages).

Other ease-of-use issues

Beyond the more obvious challenges of the legacy approach identified above, there exist other crucial flaws in first-generation BRMS offerings. One of the major obstacles to the widespread adoption of business rules "referential rule integrity" (RRI). Essentially, RRI is the concept of "guaranteeing" logical integrity of a set of rules. With any set of rules there is the potential that there are conflicts (an instance of data can satisfy the conditions of multiple rules, yet the prescribed actions for those rules are in conflict), incompleteness (the set of rules does not cover certain instances of data), and circular logic. Conflicts are undesirable because they cause non-deterministic behavior. Incompleteness is undesirable because it can lead to a lack of an appropriate response. Circular logic is undesirable because it can cause infinite loops. First-generation BRMS offerings have yet to fully solve RRI.

Other ease-of-use challenges that continue to plague many first-generation BRMS offerings include the difficulty in managing large rule bases, the lack of a full-function, business-friendly testing environment, and a complex deployment and integration model.

FIRST-GENERATION BRMS PERFORMANCE BOTTLENECK: THE "RETE WALL"

Another major issue for first-generation BRMS is a performance and scalability ceiling: the RETE wall. This limitation can be attributed to the fundamental RETE architecture, and although attempts have been made to address this issue, the ceiling has been extended a bit but not removed.

The RETE algorithm requires a significant amount of pattern matching and tedious iteration through cycles of executing one rule, updating facts, re-evaluating pattern matches and updating the agenda. Given this approach, one can imagine the strain a large workload would put on server resources. For now, let's fast forward to commonly observed results. Later, we can explore the details, walking through a conceptual example step-by-step and measuring results from various test cases.

Because of the way RETE processes decisions, increasing the size/complexity of the data payload (the instances to be evaluated against the rules, such as customers, orders, and line items) can reach a point where server resources are exhausted and the performance degrades exponentially. This phenomenon is often referred to as the "RETE wall" because the graph of decision service execution time versus data complexity payload reveals a performance ceiling (see Figure 5).

In recent years, the RETE algorithm has been refined and/or enhanced (including a RETE-3 offering lead by Charles Forgy). While these enhancements have improved performance, they have essentially pushed out the wall as opposed to removing it (see Figure 5). The ultimate goal of linear performance scalability requires a different architecture.

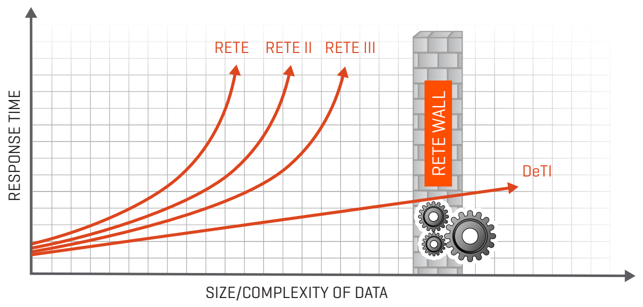


Figure 5
The RETE performance scalability wall

NEXT-GENERATION BRMS: CORTICON

The Corticon founders suffered the shortcomings of first-generation BRMS when working on rule projects in the 1990s, and their pains were the key motivation for the innovations that are at the heart of the Corticon solution. Addressing the first critical category of shortcomings, ease-of-use, they designed a BRMS solution with the guiding principles of being 100% declarative and business-friendly and achieved the only true model-driven BRMS available today.

MODEL-DRIVEN BRMS

Model-driven rule authoring is unique. The model-driven Corticon approach achieves full transparency of business logic. There are no proprietary technical languages. The Corticon business-friendly modeling metaphor and language are fully expressive (you can model any business rule logic), and the solution is ready out-of-the-box. There is no infrastructure to build or set up. The model automatically generates executable code that is optimized for peak performance. There is no manual "enhancing" at the code level for functionality or performance reasons.

BENEFITS OF SECOND-GENERATION BRMS

In addition to realizing the goals and vision of a model-driven architecture, Corticon BRMS offers a number of benefits to users, including:

- Guaranteed rule integrity
- A robust and intuitive metaphor that can handle large rule bases elegantly
- Business-friendly testing integrated into the authoring environment
- A SOA-centric solution with automated tools for generating deployment artifacts

With its comprehensive logical analysis tools, Corticon solves one of the key obstacles inhibiting the widespread adoption of business rules: rule referential integrity. As Corticon founders discovered, one of the most challenging aspects of implementing rules is ensuring they are free of logical flaws, such as conflict, or ambiguity, incompleteness, and logical loops. With the Corticon model-driven approach, rule sets are analyzed at design-time, and results are graphically displayed. Corticon offers the only BRMS solution that provides comprehensive logical analysis to ensure rule authors produce conflict-free, complete rule sets.

The proof points of Corticon success in offering a truly unique second-generation Corticon BRMS can be found in customer testimonials. Corticon BRMS has accelerated rule projects tenfold, while empowering business domain experts to manage the business rule logic from start to finish. And Corticon drives even more impressive results with respect to increasing the velocity of business change cycles. Corticon rule logic is declarative and transparent, making it easy and fast for rule authors to identify logic that needs to be changed, then make the changes, analyze the modified rule set for logical integrity, test for business intent, and redeploy without interrupting the server. Corticon customers have measured a 25X acceleration.

CORTICON EXCELS IN PERFORMANCE AND SCALABILITY

The initial Progress Corticon innovations in model-driven rule authoring solved the fundamental problem of BRMS ease-of-use, but as Corticon discovered through enterprise customer use, the execution performance of rules modeled in Corticon and executed on a RETE-based engine was still gated by the limitations of RETE. Because the Corticon architecture allows the rule dependencies and execution sequence to be resolved at deployment time (prior to runtime), a RETE engine with all of its runtime overhead and performance and scalability limitations was unnecessary. Progress Corticon subsequently developed a highly efficient, highly scalable rules engine that executes decision services that have been optimized and compiled prior to runtime. With its own engine, Progress Corticon addressed the second major shortcoming of first-generation BRMS: performance and scalability.

MEASURED CORTICON RESULTS

As discussed earlier, design-time optimization and compilation of the Corticon DeTI approach enables linear performance scalability with increasing payload size/complexity (see Figure 6 below). Elimination of the performance and scalability ceiling means that Corticon BRMS can uniquely address the performance and scalability needs of demanding enterprise environments.

Figure 7 below shows how the Corticon engine compares with a first-generation BRMS (RETE-based) engine. Note that

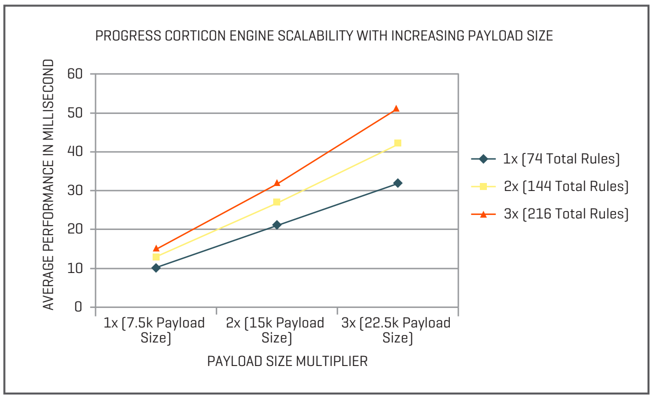


Figure 6
Corticon engine scalability with increasing payload size

Corticon scales linearly with increasing data payload size, whereas the RETE-based engine degrades exponentially. A deeper exploration into the mechanics of the RETE algorithm and the Corticon algorithm are covered in Appendix B.

While Progress Corticon does not view the popular Miss Manners scenario (see Appendix A for additional background) as a representative decision automation problem, and, despite Miss Manners being developed primarily to be a RETE-specific stress test, Corticon performed better than industry-leading RETE engines. Figure 8 below shows the results of the tests conducted by James Owens of *InfoWorld*. Not only can Corticon

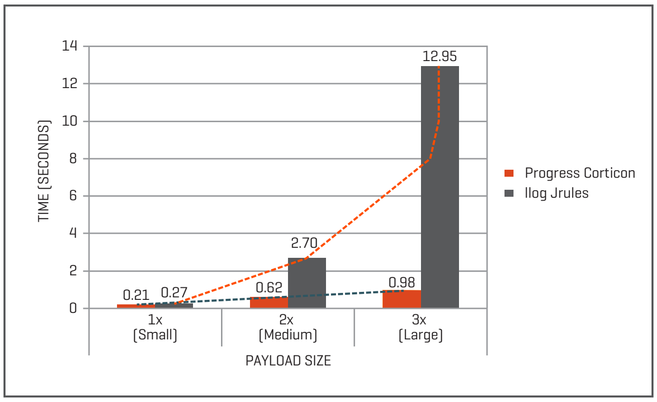
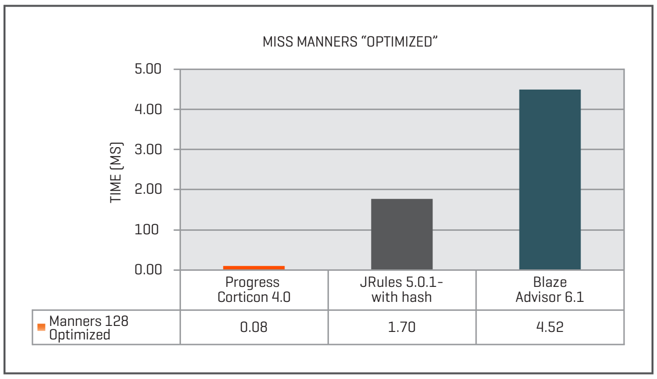


Figure 7
Performance benchmarking: Corticon vs. a RETE-based product

successfully model inferencing types of problems, but it performs well too.

A more thorough exploration of Corticon performance and scalability characteristics can be found in the Progress



	Progress Corticon 4.0	JRules 5.0.1-with hash	Blaze Advisor 6.1
🟧 Manners 128 Optimized	0.08	1.70	4.52

Figure 8
Miss Manners—run by James Owen for Info World—2006

Perspective, "Performance and Scalability: an Overview of Corticon Performance and Scalability."

RETE VS. CORTICON'S DETI: A DETAILED ANALYSIS

RETE APPROACH

To better understand the performance bottleneck inherent in RETE, it is important to explore the way RETE works. As illustrated in Figure 9 below, RETE starts with matching rule condition patterns against facts (data) to determine if the rules are eligible to fire against data instances. This rules/facts pattern matching step is performed in working memory and accounts for the bulk (often 90%) of the processing horsepower required to execute a decision service. Successful pattern matches result in activated rule instances, which are managed in a set called the "agenda" (aka the "conflict" set). The agenda may also be viewed as candidate rule instances eligible for execution at a particular point in time.

The RETE algorithm mandates that only one rule be selected from the agenda and executed, typically resulting in updates to facts in working memory. Note that the order of activated rule instances determines the selection of the next rule to be executed, and, in some cases this order can be arbitrary, absent explicit priorities defined by rule authors. As discussed in Appendix B, where a scenario is played out to illustrate the steps involved in RETE execution, the arbitrary nature of sequencing activated rule instances on the agenda can have catastrophic implications: the business results from the RETE decision service can be different when executed at different times, or, worse, the results could be consistently contrary to the intentions of the business policy because of undetected conflicting rules. (Note: Henceforth "pattern matches" will be referred to as "matches" for short.)

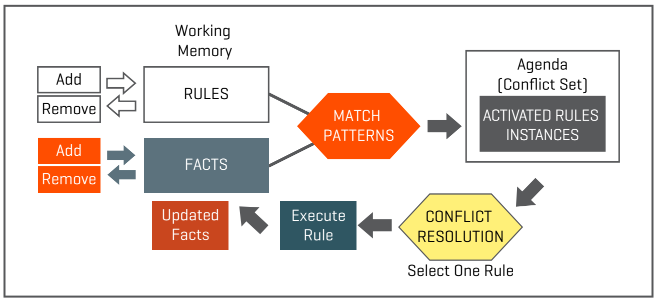


Figure 9
The conceptual workings of RETE

RETE does support changes to rules and facts (adding and removing), even during the course of processing a decision service. While this may be a valuable feature in decision support applications, this level of flexibility jeopardizes the integrity of a decision and can easily lead to unintended decision results.

COMPARISON OF ALTERNATE APPROACHES

As discussed in the section "Categorization of Rule Engines," and illustrated in the simplified Miss Manners example (see Appendix A), different algorithms or engine modes have distinct characteristics. Figure 10 highlights these characteristics. RETE, the traditional approach, automatically handles the sequencing of rule processing (albeit in some situations the use of the programming technique of rule prioritization is required to ensure deterministic behavior that meets the business intent) and provides support for fact iteration and rule recursion. These characteristics make RETE powerful in its coverage of inferencing use cases, but the downside includes a significant performance hit taken due to a rather heavyweight mechanism used for processing both fact iteration and rule recursion. Perhaps because of this performance limitation, RETE-based vendors introduced a sequential execution mode to their offerings.

The sequential execution mode can be characterized as a more streamlined execution algorithm where a manually defined sequence of rules can execute quickly. Sequential execution can handle fact iteration, but does not support rule recursion and, therefore, cannot support the types of use cases illustrated by Miss Manners. Sequential mode also has the downside of forcing the rule author to manually specify the correct sequence of rule execution to achieve the desired logical results (which violates the desired non-procedural nature of business rules).

The Corticon Deployment-Time Inferencing (DeTI) approach provides the best solution across decision automation use cases. With DeTI, the sequence of rule processing is automatically managed with no extraneous or artificial mechanisms such as rule prioritization. DeTI handles fact iteration in an optimized way, and since rule recursion can be expensive and is rarely required in the automation of business decisions, DeTI offers an optional advanced inferencing mode that supports rule recursion. This avoids the performance hit of rule recursion in the great majority (estimated to be over 95%) of business decision services, resulting in near-linear performance scalability for DeTI.

ALGORITHMS	SEQUENTIAL	RETE	DeTI
Processing Sequence	Manual	Automatic*	Automatic
Fact Iteration	Yes	Yes	Yes
Rule Recursion	No	Yes	Yes (optional)

Optimized Inferencing Modes

Advanced Inferencing Mode

Both types of iteration supported by same mechanism

Figure 10
Comparison of rule processing approaches

FIRST-GENERATION BRMS BETTER SUITED TO DECISION SUPPORT, NOT DECISION AUTOMATION

Likely because of RETE's origins in expert systems and first-generation BRMS' use of RETE, first-generation BRMS offerings are well-suited to address decision support use cases. These use cases may be characterized as centering on complex problems, defined by large and rather fluid rule bases, where one or more possible "answers" are acceptable and contribute towards a final human decision.

Decision automation, however, is the more common need in the business world. Making decisions reliably and consistently across the enterprise is what drives business automation. Fundamental objectives of decision automation include absolute logical integrity and deterministic behavior and high-performance, highly scalable execution environments (able to handle large and complex data payloads). Corticon was designed with just this set of objectives and ushers in the second-generation of BRMS with a focus on decision automation within services-oriented architectures (SOA). Figure 11 below summarizes the comparison.

PRIMARY SEGMENT SERVED	EXPERT SYSTEMS	BUSINESS AUTOMATION
Consumer of decisions	Expert	Applications Process Tasks End users
Use case category	Decision support	Decision automation
Deterministic	Not always	Always
Inconsistencies (integrity)	Tolerated	Unacceptablen
Message size (data payload)	Small (query)	Large (document)
Workload	Small	Large

Figure 11
Characterization of use cases aligned with first- and second-generation BRMS

A SYNOPSIS OF RETE AND DETI

Winding up the analysis of RETE and DeTI, it may be valuable to summarize the key points in a comparison chart (see Figure 12).

	RETE	DeTI
Strengths	Can support add/remove of individual rules	Scales linearly with increasing number of rules
	Scales well with increasing number of rules (but not data)	Scales linearly with increasing data
		Automated performance optimization
		Designed for decision automation
Weaknesses	Performance tuning requires deep expertise	Cannot add/remove individual rules (though this can be viewed as a positive since rule changes should be analyzed against the entire rule set to ensure deployment of an error-free decision service)
	Inherent performance ceiling	
	> Excessive, unnecessary pattern matches performed dynamically at runtime	
	> Rule agenda bottleneck due to one-at-a-time rule execution	

Figure 12
Summary of RETE and DeTI

SUMMARY

First-generation BRMS offerings are typically based on the RETE algorithm (which was born from expert systems), and adapted to attempt to solve the needs of decision automation. Whereas these systems offered incremental value over traditional application coding, first-generation solutions violated the most basic of business rules tenets (non-procedural and non-technical) and suffered from issues of rule referential integrity and performance bottlenecks. In the end, first-generation BRMS solutions leave much of the potential of business rules unrealized.

Corticon has pioneered the next generation of BRMS with a declarative model-driven rule authoring environment that is as easy to use as a spreadsheet, yet powerful enough to handle any business rules logic and perform well in the most demanding enterprise environments. With automated logical analysis, business-friendly testing, “snap fit“integration, and robust DeTI performance and scalability, Corticon BRMS accelerates customer rule projects, drives unprecedented agility and business control, and fulfills the promise of business rules.

APPENDIX A—DETAILS OF AN INFERENCING USE CASE

AN INFERENCING USE CASE: MISS MANNERS

While the business rules industry has no standard benchmarks, particularly for "black box" testing of business scenarios, a few test scenarios have become popular. Miss Manners is one such popular scenario that industry influencers like to use to evaluate inferencing performance, even though it may be more accurate to characterize this test as a "white box" RETE stress test.

The Miss Manners scenario involves seating guests at a table. The rules include alternating males and females and ensuring that adjacent guests have at least one hobby or interest in common. Guests and hobbies are defined such that a successful seating arrangement is possible; the larger the number of guests, the more complex the problem.

For the purpose of illustrating the basics of inferencing, and, more specifically, two types of iteration and how these may apply to BRMS vendor engine or execution modes, let's walk through a simplified version of Miss Manners. In the simplified version, consider four guests, each with two hobbies or interests (see Figure 13 below).

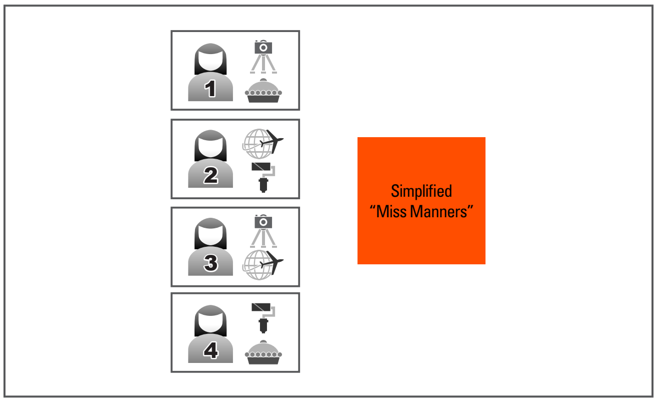


Figure 13
Simplified Miss Manners scenario

To get started, let's assume there is a rule to seat the first guest (see Figure 14). Consider Guest 1 to now be the "last-seated Guest." The key rule to evaluate is the following:

If "non-seated Guest" shares an interest with the "last-seated Guest," then seat Guest ("non-seated Guest" = "last-seated Guest").

Since Guest 2 does not satisfy the conditions of the rule (does not share a common interest), the rule is not executed and Guest 2 is not seated.

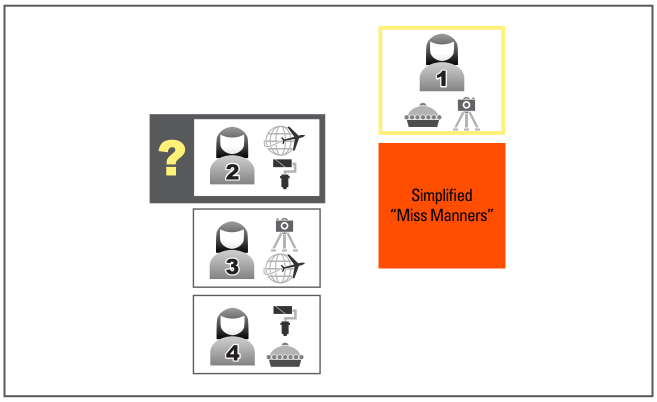


Figure 14
Guest 2 is evaluated for common hobbies with Guest 1.

Guest 3 must now be evaluated (see Figure 15 below). Guest 3 does satisfy the condition of the rule to share a common interest, the rule executes, and Guest 3 is seated next to Guest 1. Guest 3 will become the "last-seated Guest" for the next evaluation of the rule.

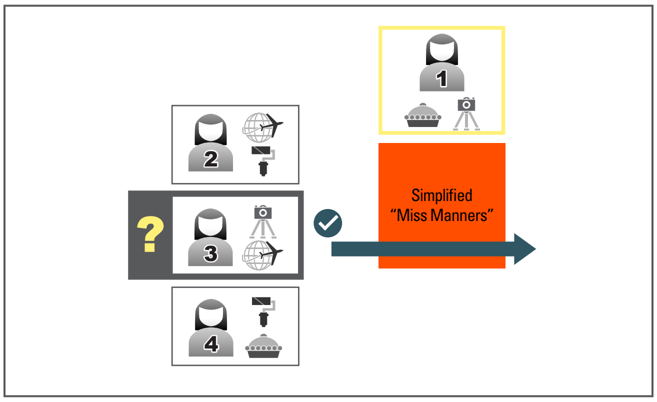


Figure 15
Guest 3 meets the condition of the rule and is seated.

Next up for evaluation is Guest 4 (see Figure 16). Guest 4 is the "non-seated Guest" in the rule's logic, and Guest 3 is the "last-seated Guest." They do not have an interest in common, so Guest 4 is not seated.

Note that so far processing has involved *fact iteration* (evaluating each of the non-seated Guests in turn). For decisions that employ a *sequential* engine mode, this is the end of the line and not all Guests will be seated. A sequential approach to this simplified problem will not be successful.

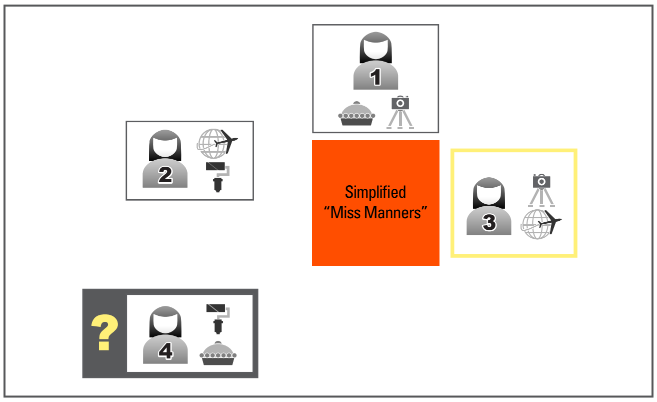


Figure 16
Completion of first pass of fact iteration

For decisions that employ logical loop iteration, or rule recursion, we reconsider "non-seated Guests" starting with Guest 2 (see Figure 17 below). Guest 2 does have a common interest with the "last-seated Guest," Guest 3, so Guest 2 is seated next to Guest 3.

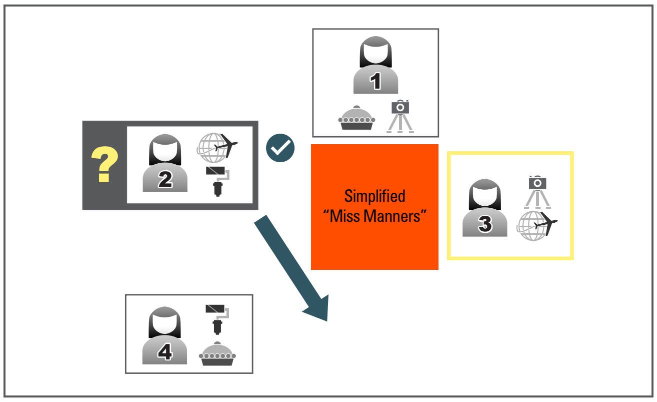


Figure 17
With rule recursion, Guests previously evaluated are evaluated again.

The last remaining "non-seated Guest," Guest 4, is now evaluated and found to have a common interest with the "last-seated Guest," Guest 2 (see Figure 18 below). Guest 4 is seated, and the processing is complete.

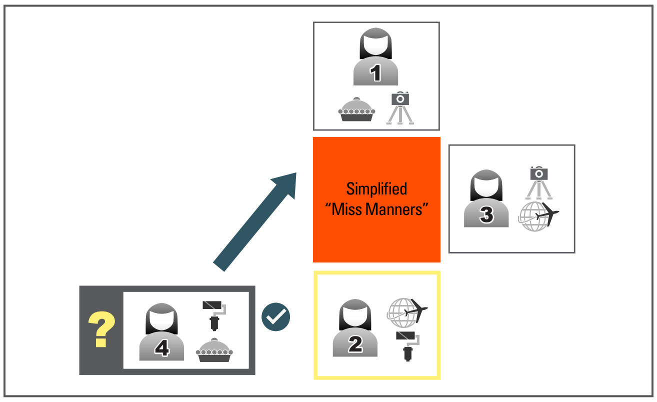


Figure 18
Completion of the last evaluation

APPENDIX B—COMPARING RETE AND DETI USING A SAMPLE SCENARIO

A BUSINESS SCENARIO IN RETE

To more fully appreciate the processing gyrations inherent in the RETE algorithm and some of the extraneous requirements forced on rule authors to compensate for logical holes in the algorithm, let's walk through a simple example.

Consider the following three rules:

- People who smoke are high risk.
- People younger than 30, female, and married are low risk.
- Reject high-risk people.

A more formal set of rule definitions might look like the following:

RULE	CONDITION(S)	ACTION
#1	Person.smoker = **T**	-> Person.risk = **High**

#2	Person.age < **30** *AND*
	Person.sex = **F** *AND*
	Person.married = **T**
		-> Person.risk = **Low**

#3	Person.risk = **High**	-> Person.risk = **T**

Figure 19
Formal rule defintions for RETE scenario

With RETE, there is a danger that rules may execute in an order that yields unintended business results. For instance, in this scenario, a married female smoker under 30 satisfies the conditions of Rule #1 and Rule #2. To ensure the business intent of smokers being rated high risk, the rule author must add a priority setting so as not to leave it to chance that the engine will sequence the rules in such a way as to have Rule #2 produce the final action. Note that although setting rule priorities for this simple example seems to be an almost trivial requirement, setting priorities for many rules can be an enormously difficult task with serious business implications if done incorrectly.

An additional requirement is levied on the rule author to add a condition to stop processing once the risk rating has been set (implemented with a null check AND added to rules #1 and #2). The added null check is required not only to prevent unintended overwrites, but also to avoid the results of an unintended rule execution to trigger other rules to fire and erroneously change other facts. The modified rule set is shown in Figure 20.

PRIORITY	RULE	CONDITION(S)	ACTION
10	#1	*Person.risk = null* *AND* Person.smoker = **T**	-> Person.risk = **High**

5	#2	*Person.risk = null* *AND*
		Person.age < **30** *AND*
		Person.sex = **F** *AND*
		Person.married = **T**
			-> Person.risk = **Low**

1	#3	Person.risk = **High**	-> Person.reject = **T**

Figure 20
Final rule set for RETE scenario

With the rules defined, let's look at a conceptual diagram of the resulting dependency network, and then step through the processing performed by the RETE engine. Figure 21 represents the dependency network. Each rule is evaluated against the data, and successful matches "activate" the rule for the data instance. As mentioned in the RETE overview section, activated rules are pooled into the agenda where the conflict resolution algorithm chooses one rule to execute, typically the one on top of the agenda, and updates the facts (data) with the results of the executed rule.

Rule logic corresponds to tracing through a "path" in the dependency graph. For instance, following the left-most path of the diagram, the nodes Risk = null and Smoker = T, combined in an AND expression defines Rule #1. Data that successfully traverses this path would activate Rule #1. Should Rule #1 execute, it would set Risk = High for the corresponding data record.

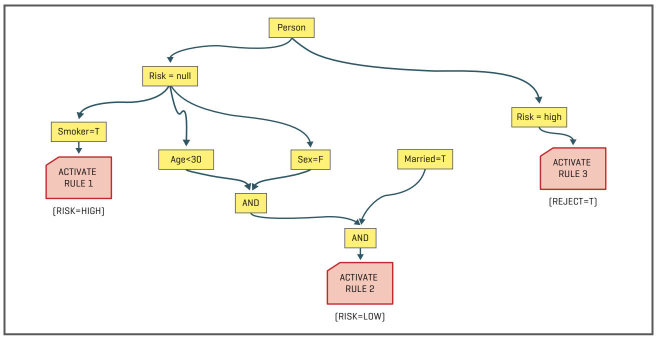
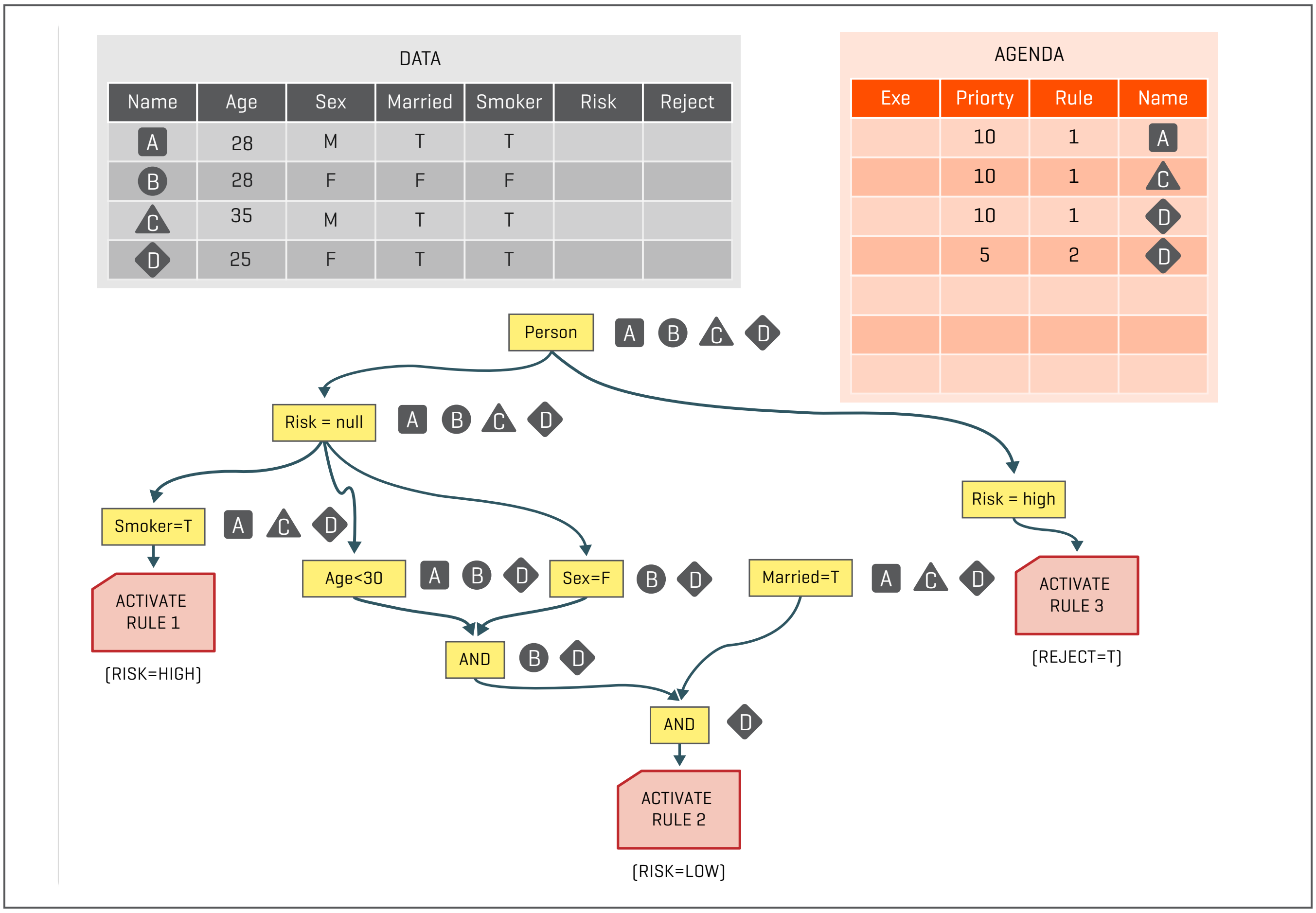


Figure 21
Dependency network diagram for RETE scenario

Now consider four instances of data. The table at the top of figure 22 below shows the details for the four instances of data. On the first pass, all four instances of Person data (A, B, C, and D) have a null value for risk. Persons A, C, and D are smokers, which activates Rule #1 for A, C, and D (see the first three lines of the agenda table). Rule #2 requires a bit more complicated matching calculation. Each Person record is evaluated against the next level of nodes: Age<30, Sex=F, and Married=T. The results are then computed in a couple of "join" operations to determine the logical ANDs. Only record D activates Rule #2. This completes the first pass.



DATA

Name	Age	Sex	Married	Smoker	Risk	Reject
A	28	M	T	T		
B	28	F	F	F		
C	35	M	T	T		
D	25	F	T	T		

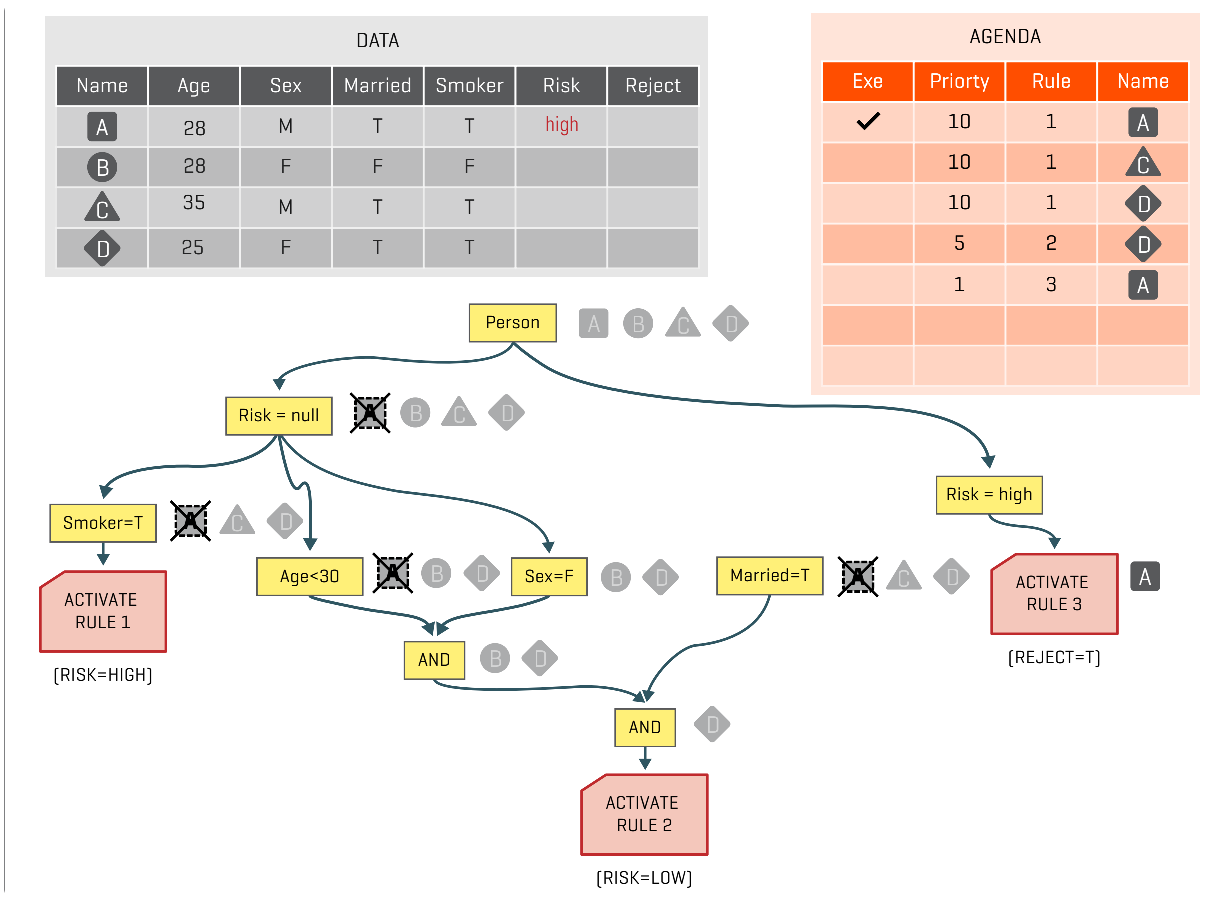
AGENDA

Exe	Priorty	Rule	Name
	10	1	A
	10	1	C
	10	1	D
	5	2	D

Figure 22
First pass through the RETE scenario

PROGRESS

With the completion of the first pass matches, it is time for the engine to select one rule to execute. The RETE engine takes the first rule-data instance with the highest priority. Rule #1 for data A is executed (noted by the check mark in the first row of the agenda table in Figure 23). The data is updated with the results from this rule execution, specifically Person A's risk has been set to High. With the data updated, it is time to make the second pass.



DATA

Name	Age	Sex	Married	Smoker	Risk	Reject
A	28	M	T	T	high	
B	28	F	F	F		
C	35	M	T	T		
D	25	F	T	T		

AGENDA

Exe	Priorty	Rule	Name
✓	10	1	A
	10	1	C
	10	1	D
	5	2	D
	1	3	A

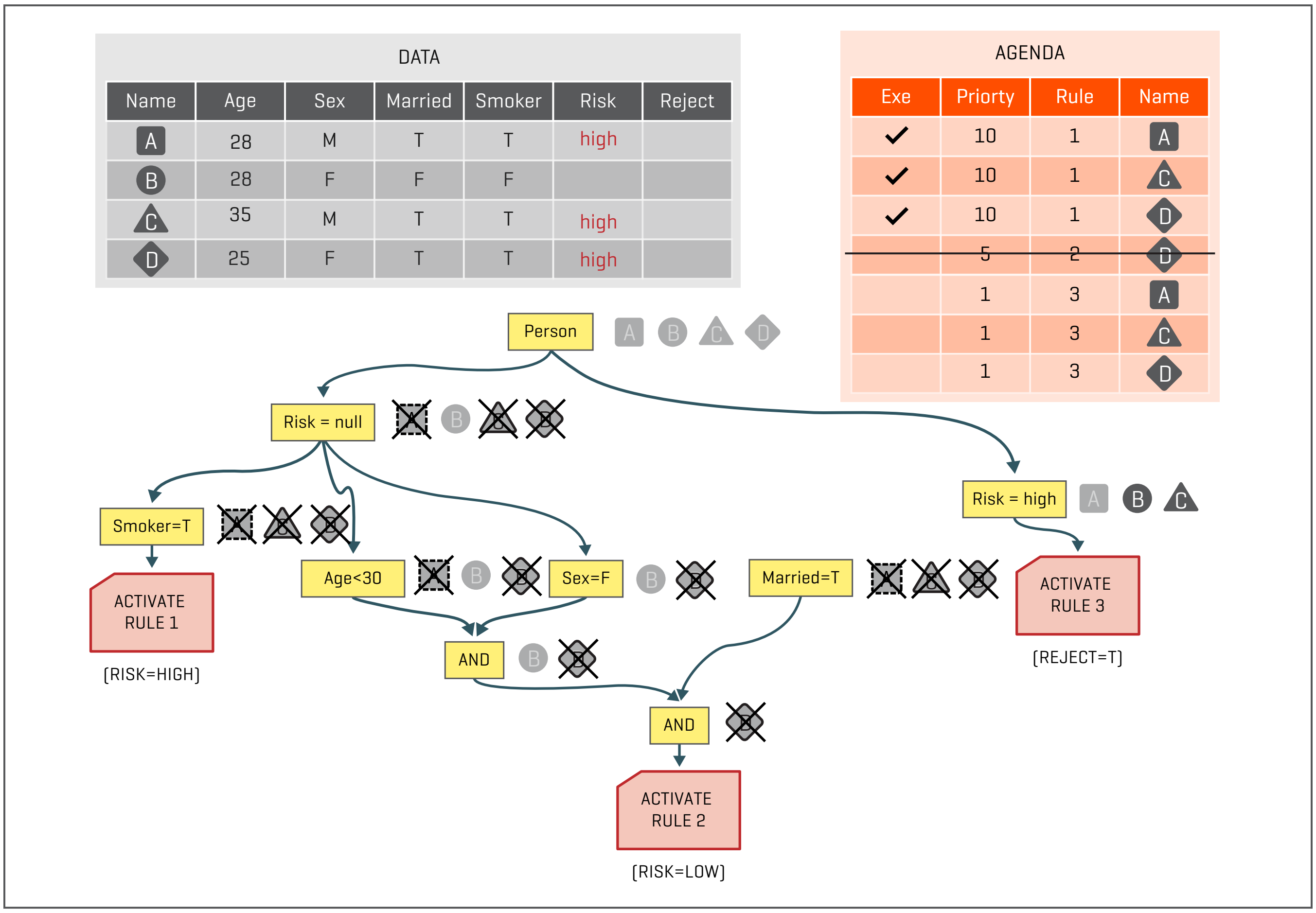
Figure 23
Second pass of RETE scenario

PROGRESS

On the second pass, A no longer satisfies the node Risk = null and, as a result, is "removed" from other "downstream" nodes. Person A, however, does satisfy the node Risk = High, and activates Rule #3. This concludes the second pass.

Completion of the second pass matching cycle places one more active rule on the agenda, and the engine executes the next highest priority rule, Rule #1 for C. The result is to set Person C's risk to high.

A third pass yields similar results for D (risk is set to high, and Rule #3 activated). A curious thing happens on the subsequent matching cycle: with risk no longer null for D, Rule #2 is removed from the agenda (see Figure 24). At this stage of processing, the only active rule instances are Rule #3 for A, C, and D.



DATA

Name	Age	Sex	Married	Smoker	Risk	Reject
A	28	M	T	T	high	
B	28	F	F	F		
C	35	M	T	T	high	
D	25	F	T	T	high	

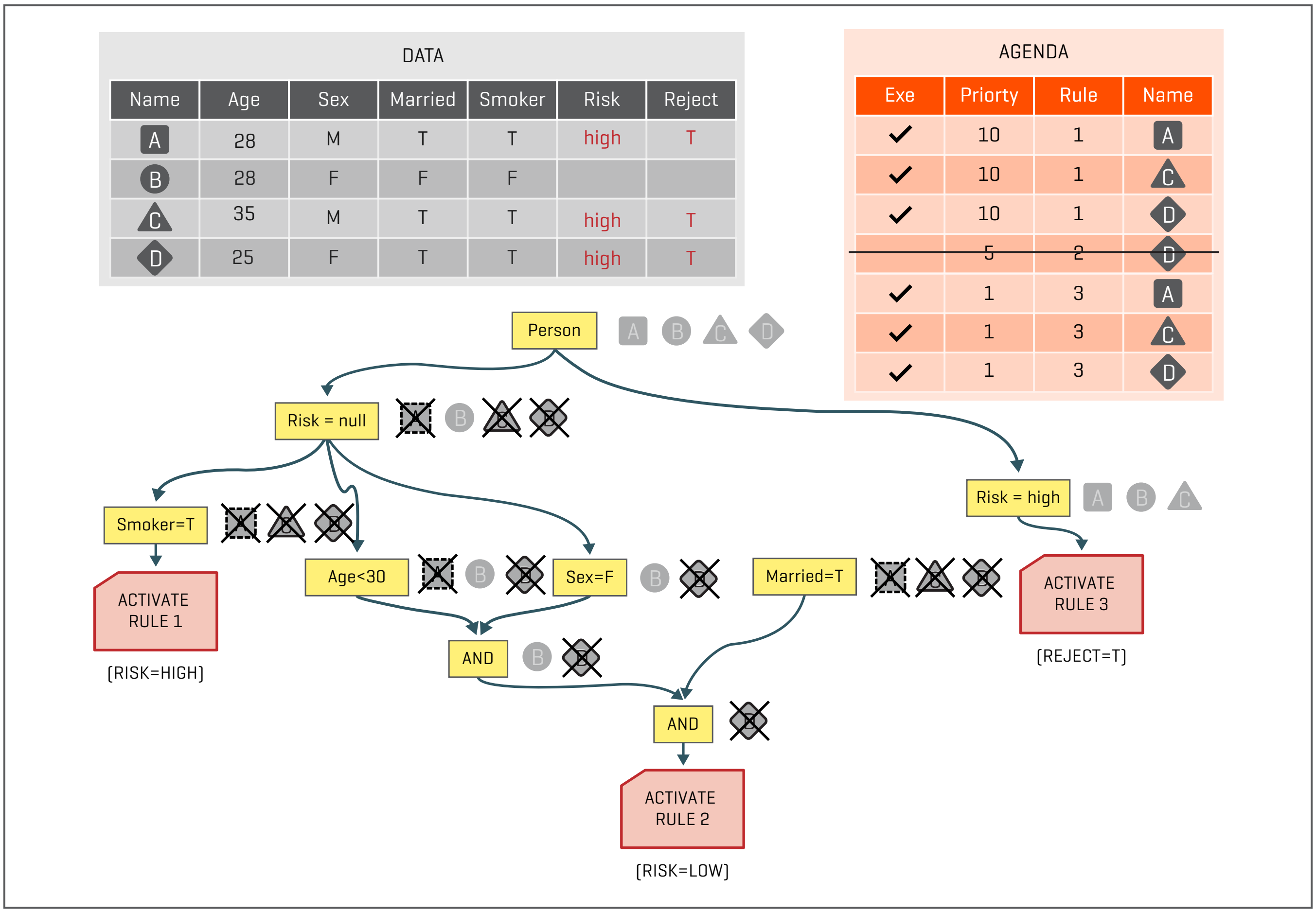
AGENDA

Exe	Priorty	Rule	Name
✓	10	1	A
✓	10	1	C
✓	10	1	D
	~~5~~	~~2~~	~~D~~
	1	3	A
	1	3	C
	1	3	D

Figure 24
Through pass 3 of the RETE scenario

To complete this scenario, let's fast forward through the execution of each instance of Rule #3 to reject A, C, and D because they are high risk. The final results are shown in Figure 25 below, with the ending data values for A, C, and D (B did not satisfy any of the rules in this set).

With the help of the additional constructs (priorities and null check as the mechanism to implement a logical override), the RETE algorithm was able to solve this scenario. But how efficient is the algorithm?



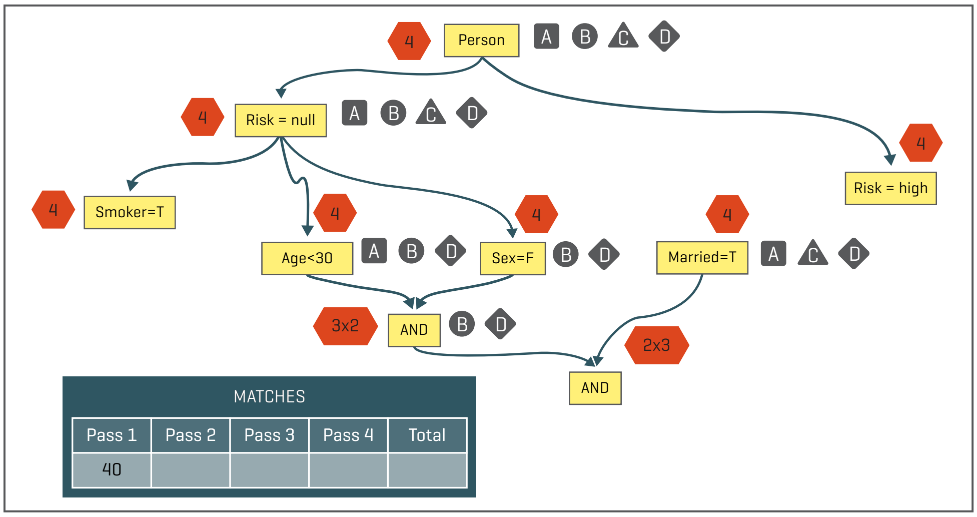
DATA

Name	Age	Sex	Married	Smoker	Risk	Reject
A	28	M	T	T	high	T
B	28	F	F	F		
C	35	M	T	T	high	T
D	25	F	T	T	high	T

AGENDA

Exe	Priorty	Rule	Name
✓	10	1	A
✓	10	1	C
✓	10	1	D
	~~5~~	~~2~~	~~D~~
✓	1	3	A
✓	1	3	C
✓	1	3	D

Figure 25
Final state of the RETE scenario



MATCHES

Pass 1	Pass 2	Pass 3	Pass 4	Total
40				

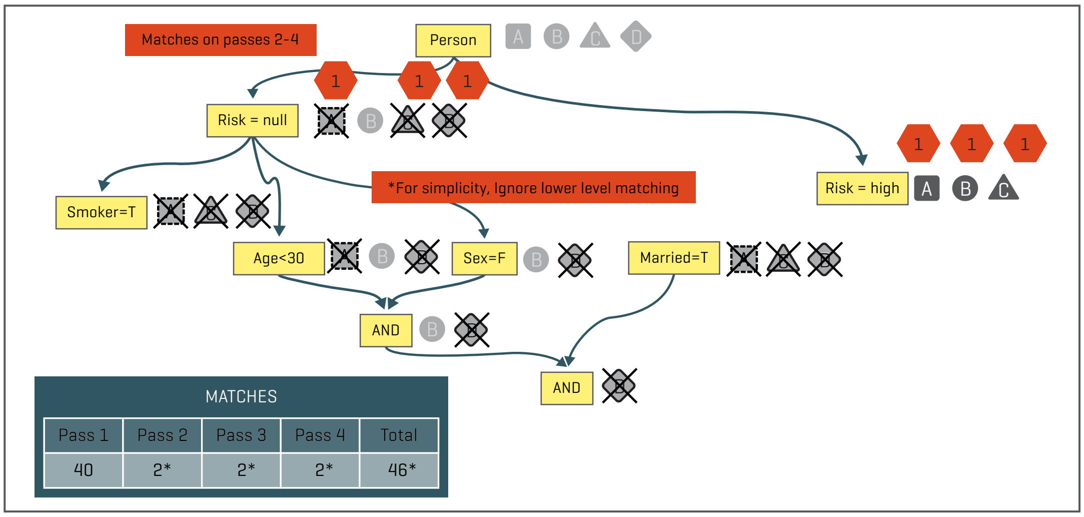
Figure 26
First pass tally of RETE evaluations for matches

Let's review the engine processing and tally up the number of matches that had to be processed (see the number at each node noted within the green octagon in Figure 26 above). This will give us some indication of the burden on the engine since pattern matching accounts for 90% of processing time. In the first pass, all four data instances were evaluated for possible matches at the first few levels of nodes, including Person, Risk=null, Smoker=T, and so on (even if some of the results were "no match," the evaluation had to be performed). Because the AND nodes are essentially joins, the number of instances from the nodes that are to be joined are multiplied together to determine the number of match evaluations for the AND node. For instance, the AND of Age<30 and Sex=F takes three instances satisfying the first condition (A, B, and D) and performs a join with the two instances satisfying the second condition (B and D), for a total of 3x2, or 6. The final tally of evaluations for matches is 40.

As we know from walking through the complete scenario of RETE processing, it requires more than one "pass" in RETE to solve this problem. On the second "pass," data for A was updated, so it is re-evaluated for a match at the Risk=null node and the Risk=high node. Similar reasoning can be applied to "passes" three and four, making the final tally 46 (40 for the first pass, plus 2 each for the passes 2-4). Figure 27 below diagrams the actions taken and evaluation of matches for passes 2-4.

This detailed view of how RETE processes a simple rule example shows the brute force required to solve relatively simple logic. Imagine the amount of processing required for a scenario of 100s of rules and 100s of data records.

Figure 27
Accounting for passes 2-4 of RETE scenario



MATCHES

Pass 1	Pass 2	Pass 3	Pass 4	Total
40	2*	2*	2*	46*

A BUSINESS SCENARIO IN DETI

To better appreciate the difference between Corticon BRMS and first-generation BRMS, let's return to the scenario used to illustrate how RETE works. The processing flow in Corticon for the three simple rules looks much simpler (see Figure 28). Note that this is not an in-memory dependency network like RETE. However, during decision service deployment, the same dependency graph is used by the Corticon optimizing compiler to determine the most efficient processing flow, accounting for all logical overrides used to resolve any rule conflicts.

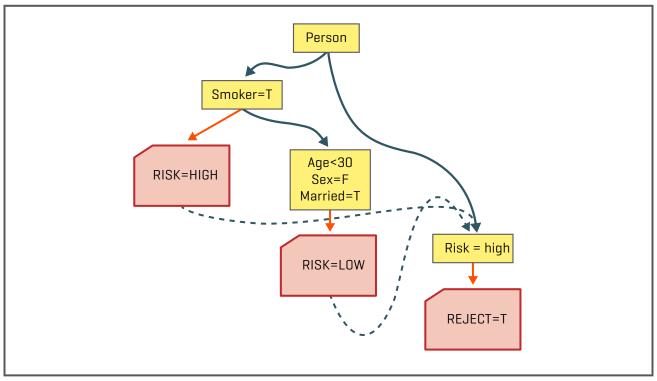


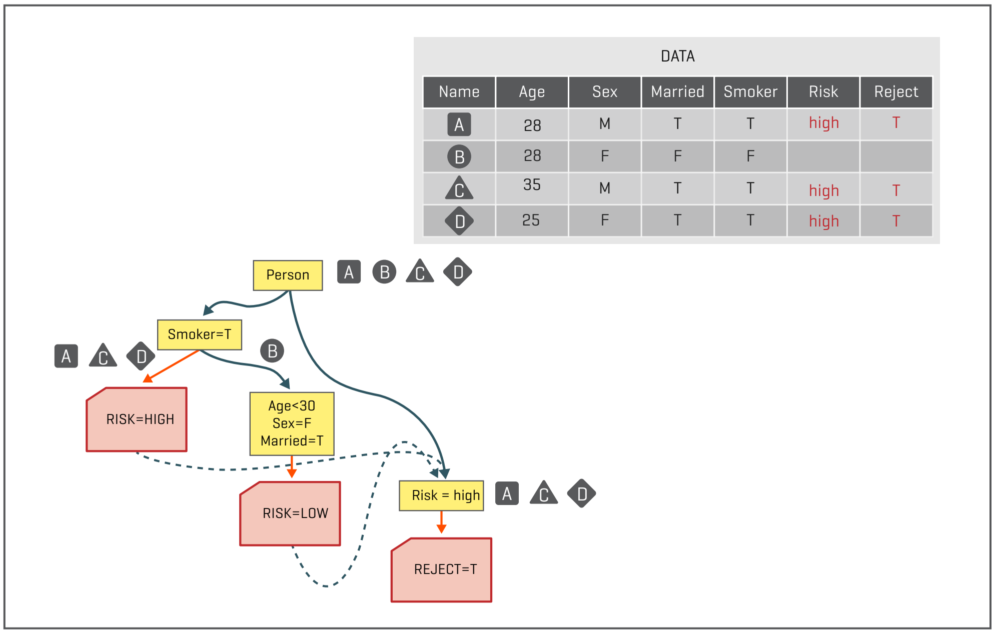
Figure 28
Dependency network in Corticon

With Corticon, the logical conflict between Rule #1 and Rule #2 (it is possible to be a married female younger than 30 who is also a smoker) can be automatically identified and resolved with an override. This results in checking the Smoker condition first. Data instances that satisfy it are processed for Rule #1, and only data instances that do not satisfy it are evaluated in Rule #2. Note also that the DeTI algorithm is more efficient in processing multi-conditional rules, such as Rule #2.

The DeTI algorithm also figures out that Rule #3 must be processed after Rules #1 and #2 since Rule #3's condition is dependent on the result of the actions of Rules #1 and #2.

Applying the same data (see Figure 29), we see that Rule #1 fired for A, C, and D, setting risk to high, and, subsequently, Rule #3 fired for A, C, and D, setting reject to true. While B did not qualify for Rule #1, it also did not qualify for Rule #2 (B is not married) and did not end up with a risk rating, resulting in Rule #3 not firing either.

Figure 29
DeTI example scenario



DATA

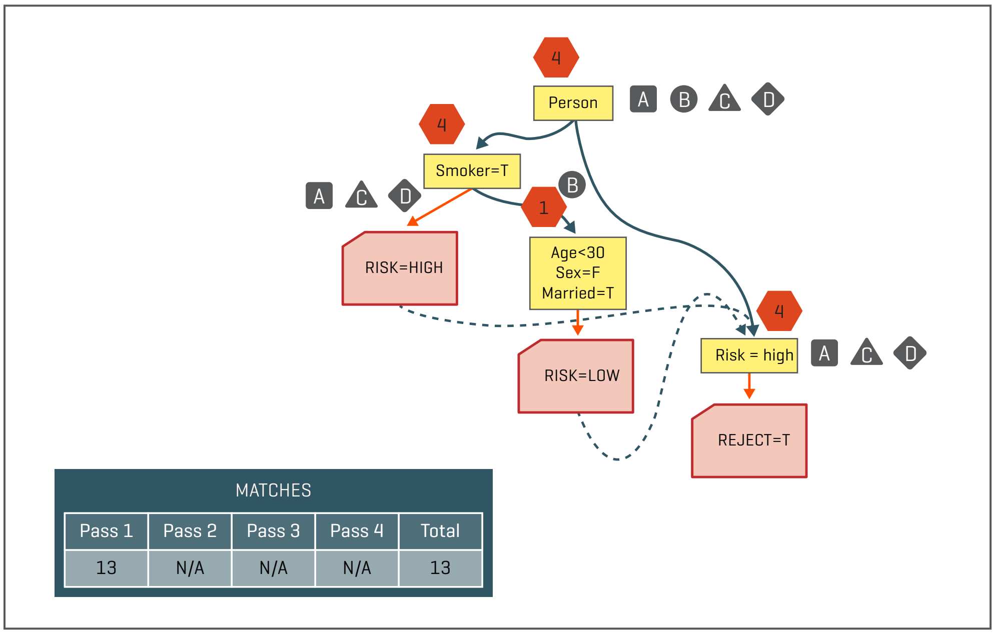
Name	Age	Sex	Married	Smoker	Risk	Reject
A	28	M	T	T	high	T
B	28	F	F	F		
C	35	M	T	T	high	T
D	25	F	T	T	high	T

To complete the analysis, let's tally up the number of matches (see Figure 30). All four instances of data were evaluated for matches at the root level (Person) and for Rule #1 (Smoker=T). Notice that the RETE approach performed a lot of unnecessary evaluations on all Person instances to activate B for Rule #2 and then subsequently deactivated it as a side-effect of Rule #1 execution. Since the DeTI algorithm considers rule overrides during deployment, it really cut down on the number of evaluations required for Rule #2, as only B was even considered for it.

All four instances were evaluated for Rule #3, even though only A, C, and D satisfied the conditions to trigger Rule #3 to fire. That puts the grand total at 13 matches, all performed in a single pass—which compares favorably to the 46 matches required in the RETE approach.

ABOUT PROGRESS CORTICON

Progress Corticon BRMS enables organizations to make better, faster decisions by automating business rules. Its patented "no-coding" rules engine is used by over 450 customers to automate their most sophisticated decision processes, reducing development and change cycles by 90%. Automated decision management with Corticon empowers organizations to improve productivity and customer service, and adapt quickly to changing market conditions.



MATCHES				
Pass 1	Pass 2	Pass 3	Pass 4	Total
13	N/A	N/A	N/A	13

Figure 30
DeTI evaluation of matches

PROGRESS SOFTWARE

Progress Software Corporation (NASDAQ: PRGS) is a global software company that simplifies the development, deployment and management of business applications on-premise or in the cloud, on any platform or device, to any data source, with enhanced performance, minimal IT complexity and low total cost of ownership.

WORLDWIDE HEADQUARTERS

Progress Software Corporation, 14 Oak Park, Bedford, MA 01730 USA Tel: +1 781 280-4000 Fax: +1 781 280-4095 On the Web at: www.progress.com

Find us on facebook.com/progresssw twitter.com/progresssw youtube.com/progresssw

For regional international office locations and contact information, please go to www.progress.com/worldwide

Progress and Corticon are trademarks or registered trademarks of Progress Software Corporation or one of its affiliates or subsidiaries in the U.S. and other countries. Any other marks contained herein may be trademarks of their respective owners. Specifications subject to change without notice.

© 2011-2012, 2014 Progress Software Corporation. All rights reserved.

Rev. 8/14 | 120103-0117

PROGRESS