Sistemas Baseados em Regras
Aula2: Conceituação

Profa. Patrícia Dockhorn Costa
pdcosta@inf.ufes.br
www.inf.ufes.br/~pdcosta/ensino

Um problema de matemática...

- 4 jogadores de golfe estão organizados em linha (da esquerda para direita), preparados para jogar. Cada jogador está vestindo calças com cores diferentes; um está vestindo uma calça vermelha; O jogador à direita de Fred está vestindo calças azuis.

Um problema de matemática... (cont.)

- Joe é o segundo da fila.
- Bob está vestindo calça xadrez.
- Tom não está na posição 1 ou 4; e ele não está vestindo a calça laranja.
- Em que ordem os 4 jogadores darão as primeiras tacadas, e quais são as cores de cada calça?

Um problema de matemática... (cont.)

- Não existe fórmula ou procedimento analítico para resolver este problema...
- Regras podem ajudar!
- Vamos encontrar uma solução em Jess! Não se preocupem com a linguagem, apenas com a abordagem!

Solução

- De forma geral, precisamos:
  - Escolher uma maneira de representar as possíveis combinações de nomes, posições e cores de calças.
  - Escrever a regra que descreve o problema.

Solução (em Jess)

- Precisamos definir as estruturas de dados que são adequadas para representar o conhecimento necessário para resolver o problema

```
(deftemplate cor-calca (slot of) (slot is))
(deftemplate posicao (slot of) (slot is))
```

Solução (em Jess)

- O deftemplate define um tipo, com propriedades (aqui chamados ***slots***)
- ***Fatos*** são as instâncias!
- Para o nosso exemplo, teremos 32 fatos, formando todas as combinações possíveis, por exemplo;

```
(cor-calca (of Bob) (is red))
(posicao (of Joe) (is 3))
```

Solução (em Jess)

- Podemos criar uma regra para inserir os 32 fatos na ***working memory***:

```
(defrule gera-possibilidades
=>
(foreach ?nome (create$ Fred Joe Bob Tom)
   (foreach ?cor (create$ red blue plaid orange)
       (assert (cor-calca (of ?nome) (is ?cor))))
   (foreach ?pos (create$ 1 2 3 4)
       (assert (posicao (of ?nome)(is ?pos)))))
```

Solução (em Jess)

- ... agora precisamos encontrar o subconjunto de fatos que realmente representam a solução.
  - **?c** para representar "alguma cor"
  - **?p** para representar "alguma posição"
  - **?c1...?c4** para representar as cores de Fred, Joe, Bob e Tom (respectivamente)
  - **?p1...?p4** para representar as posições de Fred, Joe, Bob e Tom (respectivamente)

... sobre Fred

- "O jogador à direita de Fred está vestindo calças azuis"

```
(defrule acha-solucao

(posicao (of Fred) (is ?p1))
(cor-calca (of Fred) (is ?c1))

(posicao (of ?n&~Fred) (is ?p&: (eq ?p (+
  ?p1 1))))

(cor-calca (of ?n&~Fred) (is blue&~?c1))
```

... sobre Joe

- “Joe é o segundo da fila.”

```
... Continuando
(posicao (of Joe) (is ?p2&2&~?p1))
(cor-calca (of Joe) (is ?c2&~?c1))
```

... sobre Bob

- “Bob está vestindo calça xadrez.”

```
... Continuando
(posicao (of Bob) (is ?p3&~?p1&~?p2&~?p))
(cor-calca (of Bob&~?n) (is
  plaid&?c3&~?c1&~?c2))
```

... sobre Tom

- “Tom não está na posição 1 ou 4; e ele não está vestindo a calça laranja.

```
(posicao (of Tom&~?n) (is
  ?p4&~1&~4&~?p3&~?p1&~?p2))
(cor-calca (of Tom) (is
  ?c4&~orange&~blue&~?c1&~?c2&~?c3))
```

O RHS da regra

```
=>
  (printout t Fred " " ?p1 " " ?c1 crlf)
  (printout t Joe" " ?p2 " " ?c2 crlf)
  (printout t Bob " " ?p3 " " ?c3 crlf)
  (printout t Tom " " ?p4 " " ?c4 crlf)
```

- Resposta:
  - Fred 1 orange
  - Joe 2 blue
  - Bob 4 plaid
  - Tom 3 red

Usando Informação Incompleta

- E se não tivéssemos a informação de que Joe é o segundo da fila?

```
(posicao (of Joe) (is ?p2&~?p1))
(cor-calca (of Joe) (is
  ?c2&~?c1))
```

Usando Informação Incompleta (cont.)

```
Fred 3 orange
Joe 4 blue
Bob 1 plaid
Tom 2 red
```

```
Fred 2 orange
Joe 1 blue
Bob 4 plaid
Tom 3 red
```

```
Fred 1 orange
Joe 3 blue
Bob 4 plaid
Tom 2 red
```

```
Fred 2 orange
Joe 4 blue
Bob 1 plaid
Tom 3 red
```

```
Fred 1 orange
Joe 2 blue
Bob 4 plaid
Tom 3 red
```

```
Fred 1 orange
Joe 4 blue
Bob 3 plaid
Tom 2 red
```

Refletindo...

- Vocês acham que resolver “charadas de lógica” como esta não faz parte das tarefas de um programador?
- Problemas com informações incompletas são muito comuns em ambientes de negócio!
- Quando trabalhamos com “suposições” e “possibilidades”, soluções tradicionalmente algoritmicas não são as mais adequadas!

Programação Procedural

- Incluindo aqui programação OO
- Programador fala para o computador o que fazer, como fazer e em que ordem
- Funcionam bem para os problemas que têm entradas bem definidas, e que um conjunto conhecido de passos deve ser executado para resolver o problema
- E.g., Computações matemáticas

Programação Declarativa

- Programa declarativo descreve o que o computador deve fazer, porém omite as instruções de como fazer
- Devem ser executados por algum sistema *run-time* que entende como “preencher as lacunas” e resolver o problema usando as informações declarativas

Programação Declarativa (cont.)

- Solução natural para problemas nas áreas de:
  - Control
  - Diagnosis
  - Prediction
  - Classification
  - Pattern recognition
  - Situational awareness

Programa Baseado em Regras

- Precisamos escrever apenas as regras.
- Um outro programa (chamado *rule engine*) determina que regras são “aplicáveis” e as executa quando for apropriado.
- Ou seja, uma solução baseada em regra para um problema grande, pode ser simplificado!
- Podemos nos concentrar somente nas regras!

Regra

- “Uma instrução ou comando que se aplica em certas situações”
  - “Não masque chicletes na escola”
  - “Não corra enquanto segurar uma tesoura”
  - “Se pode dar errado, vai dar”

Regra (cont.)

- Podemos codificar em regras conhecimento (em termos de lógica) sobre o mundo!
- Regras são parecidas com expressões if-then

```
IF estou na escola                          LHS
AND estou mascando chiclete
THEN cuspa o chiclete                       RHS
END
```

- LHS ou premissas ou predicados
- RHS ou ações ou conclusões

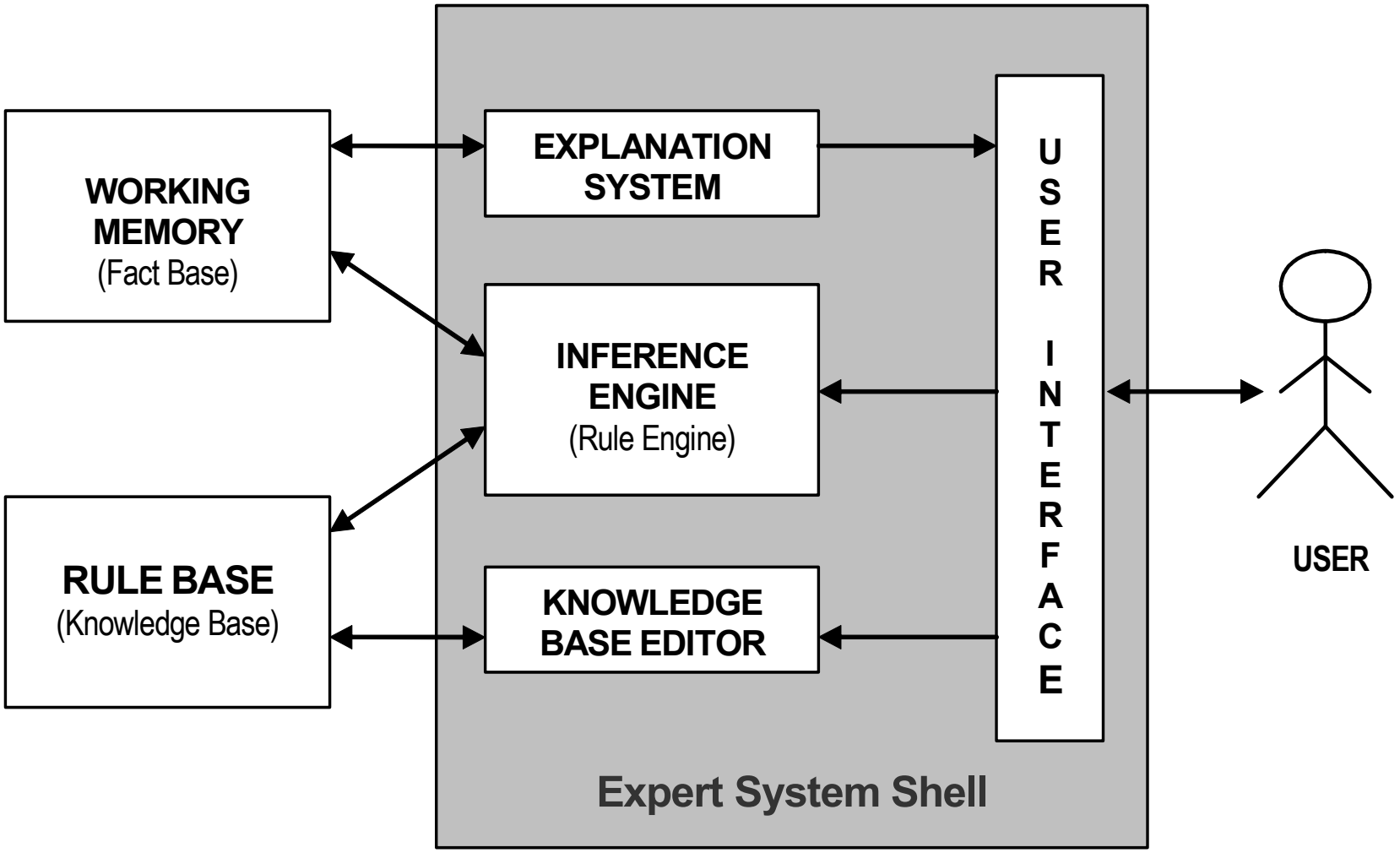
Sistema Baseado em Regra

- "... é um sistema que usa regras para derivar conclusões a partir de premissas"
- Consiste de um conjunto de regras que podem ser aplicadas repetidamte em uma coleção de fatos
  - Fatos representam as situações no mundo real. Na lógica são conhecidos como assertivas.
  - Regras são comandos ou ações que devem ser tomados em certas situações.

Strong and Loose Coupling

- Strongly coupled rules
  - A execução de uma regra resulta na execução de outra, que resulta na execução outra... (chain of logic)
  - Menos flexibilidade
  - Talvez possam ser implementadas com “decision trees” ou “hard coded”
- Loosly coupled rules
  - Oferecem mais flexibilidade (regras podem ser modificadas, removidas ou inseridas mais facilmente)

Arquitetura de um Sistema Baseado em Regra



The Inference Engine

- Aplica as regras a dados na working memory
- Geralmente funciona em ciclos discretos:
  - Regras são comparadas com a *working memory* (pelo *pattern matcher*). O conjunto de regras que devem ser ativadas é chamado de *conflict set*
  - O *confict set* é ordenado para formar a *agenda* usando um processo chamado *conflict resolution*
  - Primeira regra é executada (possivelmente mudando a working memory)

The Rule Base

- Contém as regras
- Normalmente em alguma estrutura de dados eficiente
- Em Jess e Drools: *rete network*

The Working Memory

- Contém as informações necessárias para o sistema de regras funcionar
- Podem conter premissas e/ou conclusões
- Tipicamente tem estrutura de índices como em BD para otimizar busca

Desenvolvimento de SBR's

- Engenharia de conhecimento
- Estruturação dos dados
- Testes
- Construção das interfaces
- Escrevendo as regras
- Desenvolvimento iterativo

Tipos de Regras (slides da Carol)

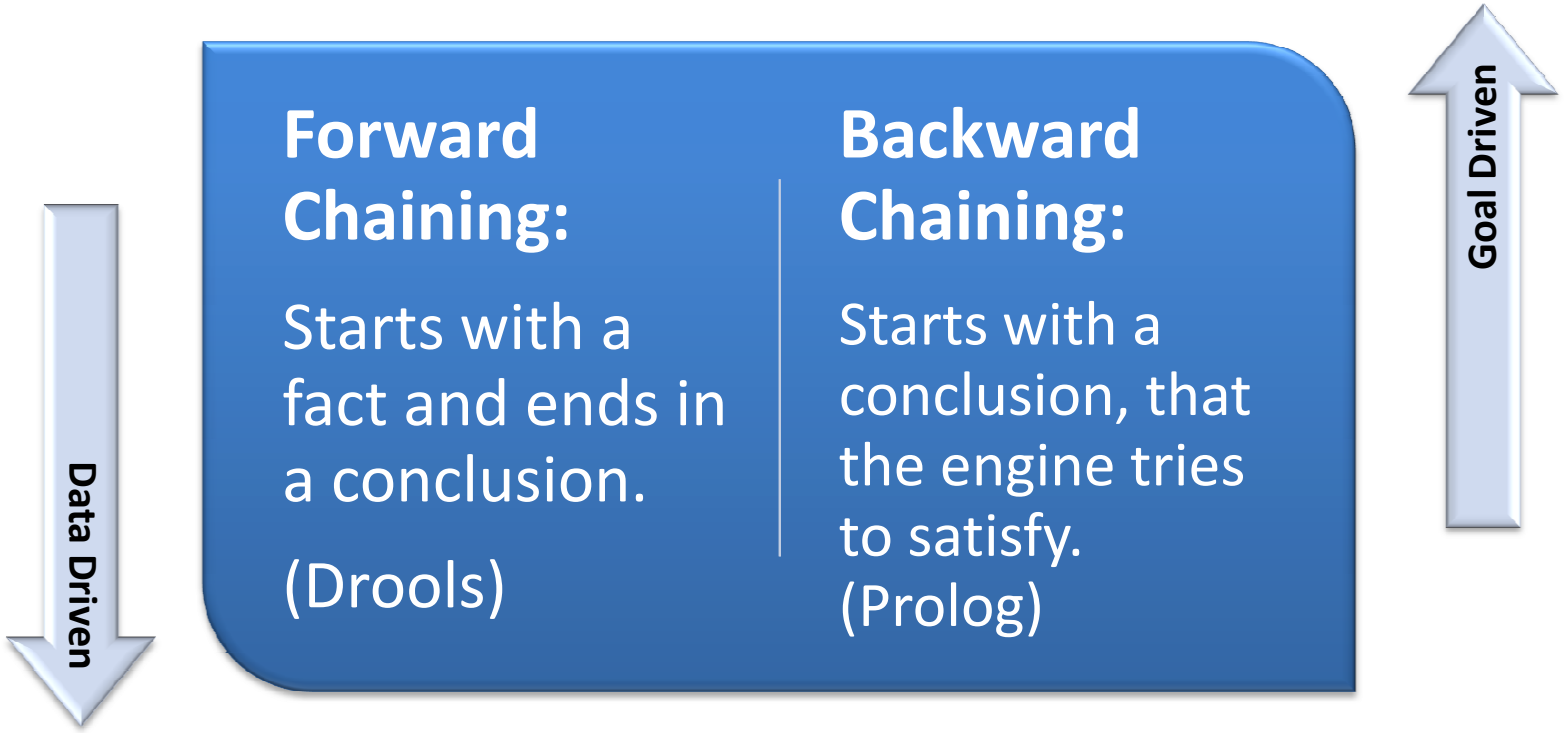
- Regra de Derivação
  - Deriva consequencias lógicas que são “asserted” e não executadas

If <*condition*> then <*conclusion*>

- Regra de Produção
  - Produz consequencias práticas, que são executadas
  - Podem implementar regras de derivação

If <*condition*> then <*action*>

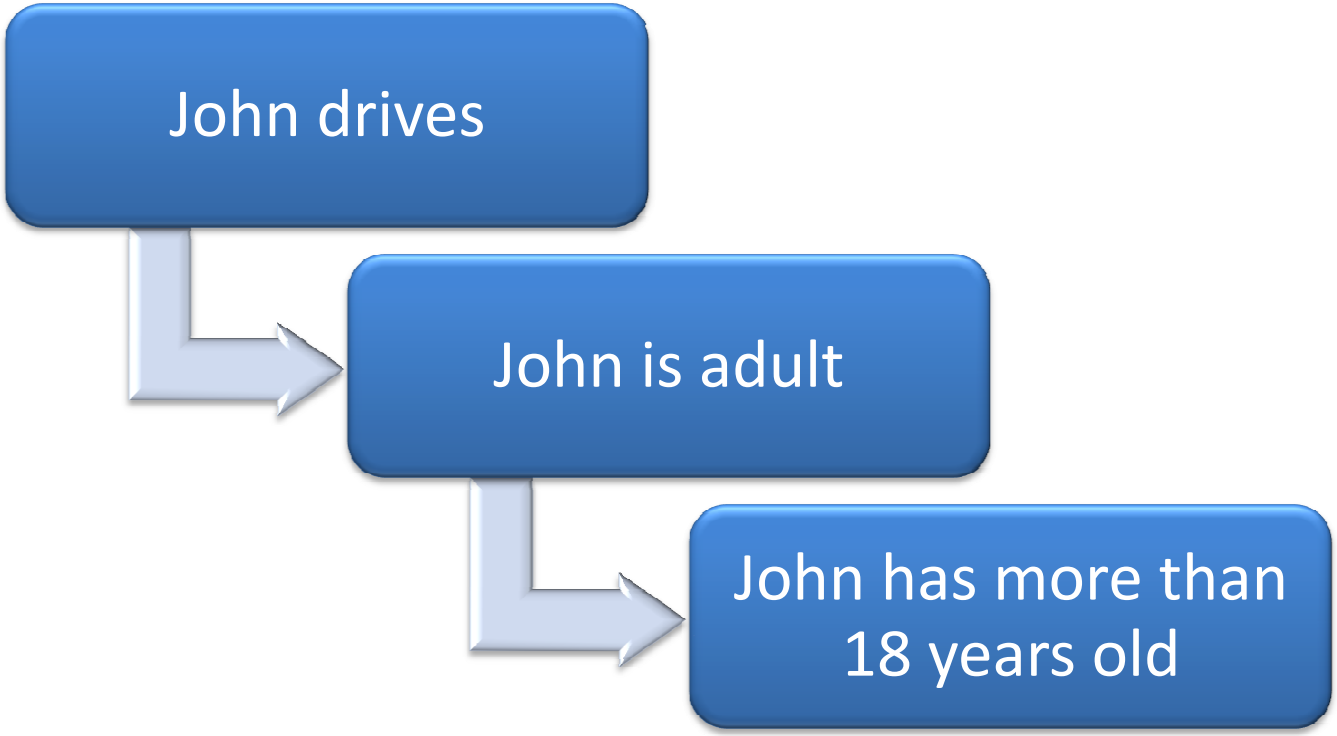
Métodos de Execução de Regras



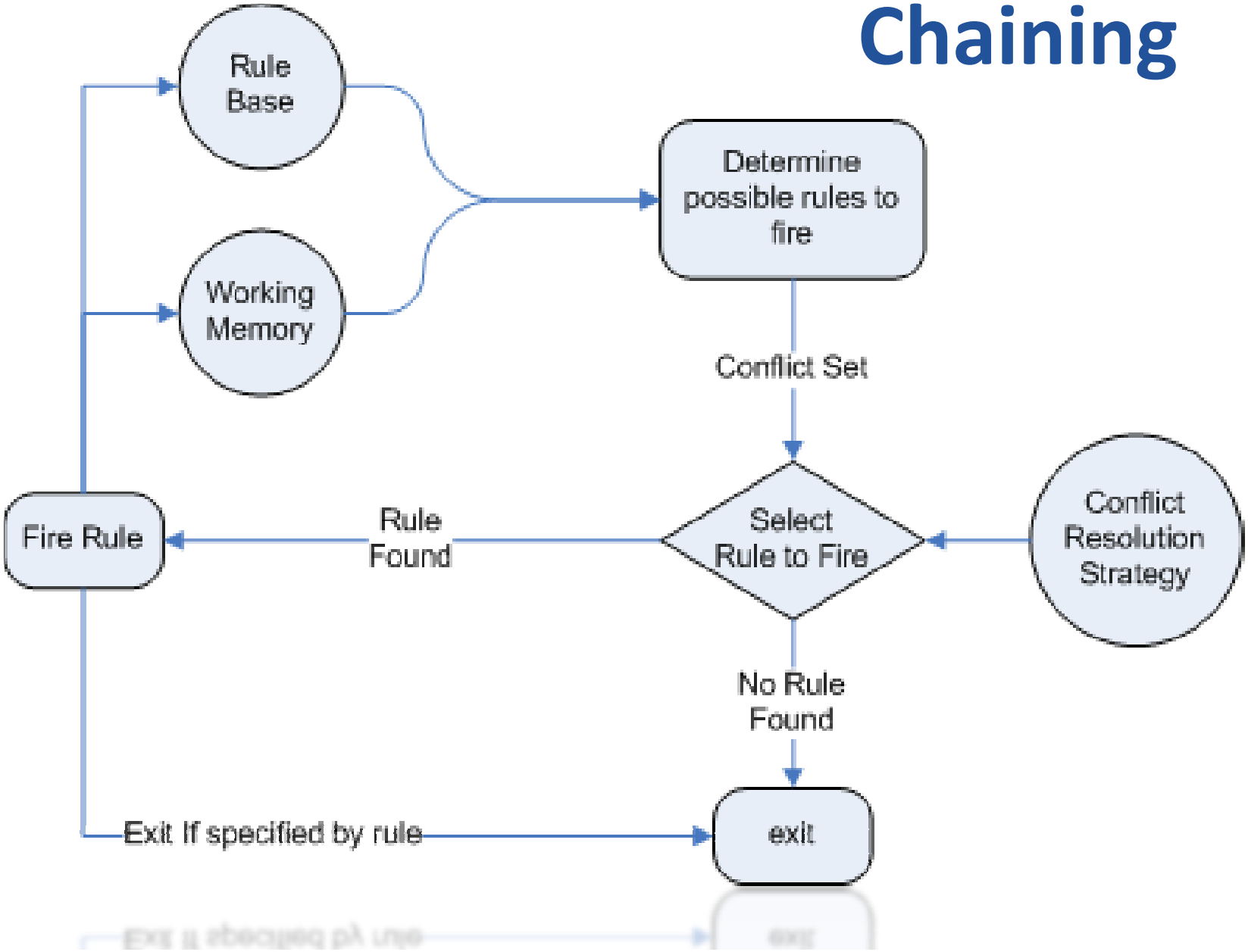
Forward Chaining

1. If <John drives> then <John is adult>

2. If <John is a adult> then <John has more than 18 years old>



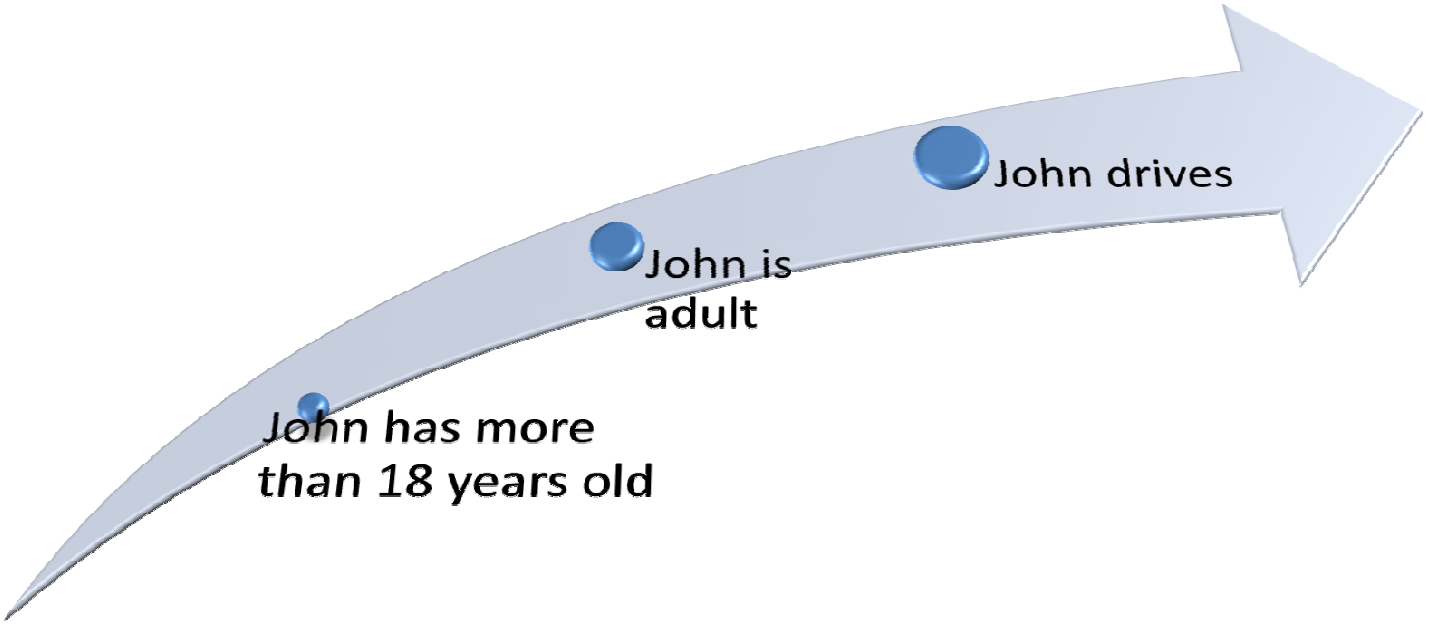
Forward Chaining



Backward Chaining

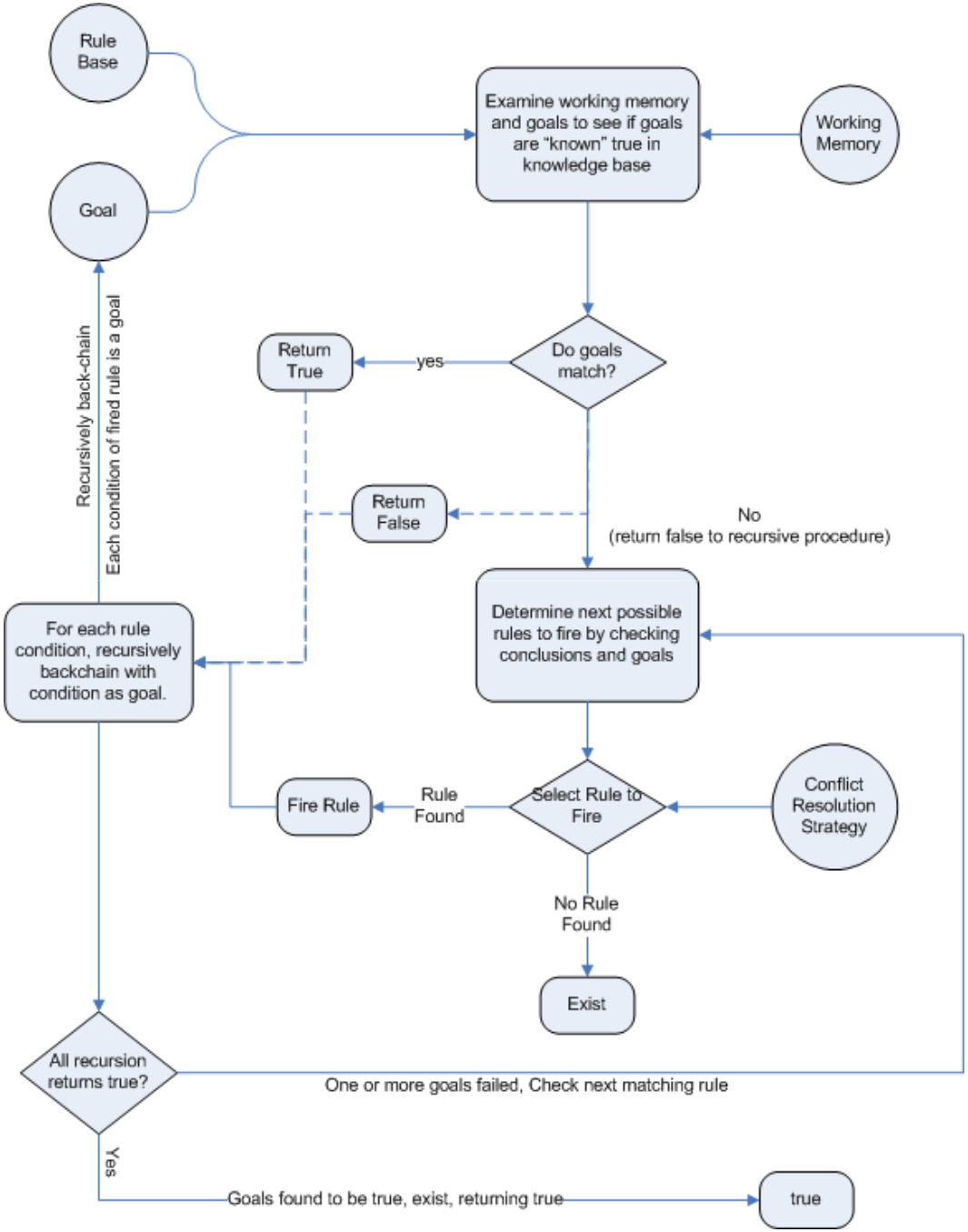
1. If <John drives> then <John is adult>

2. If <John is a adult> then <John has more than 18 years old>



John has more than 18 years old given he drives.

Backward Chaining

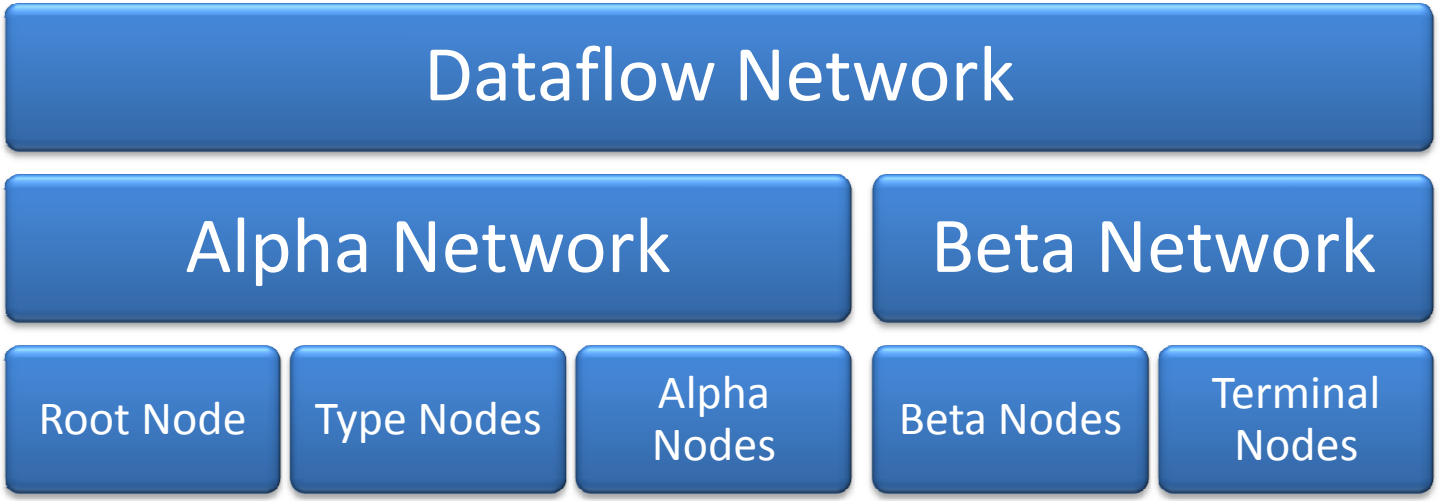


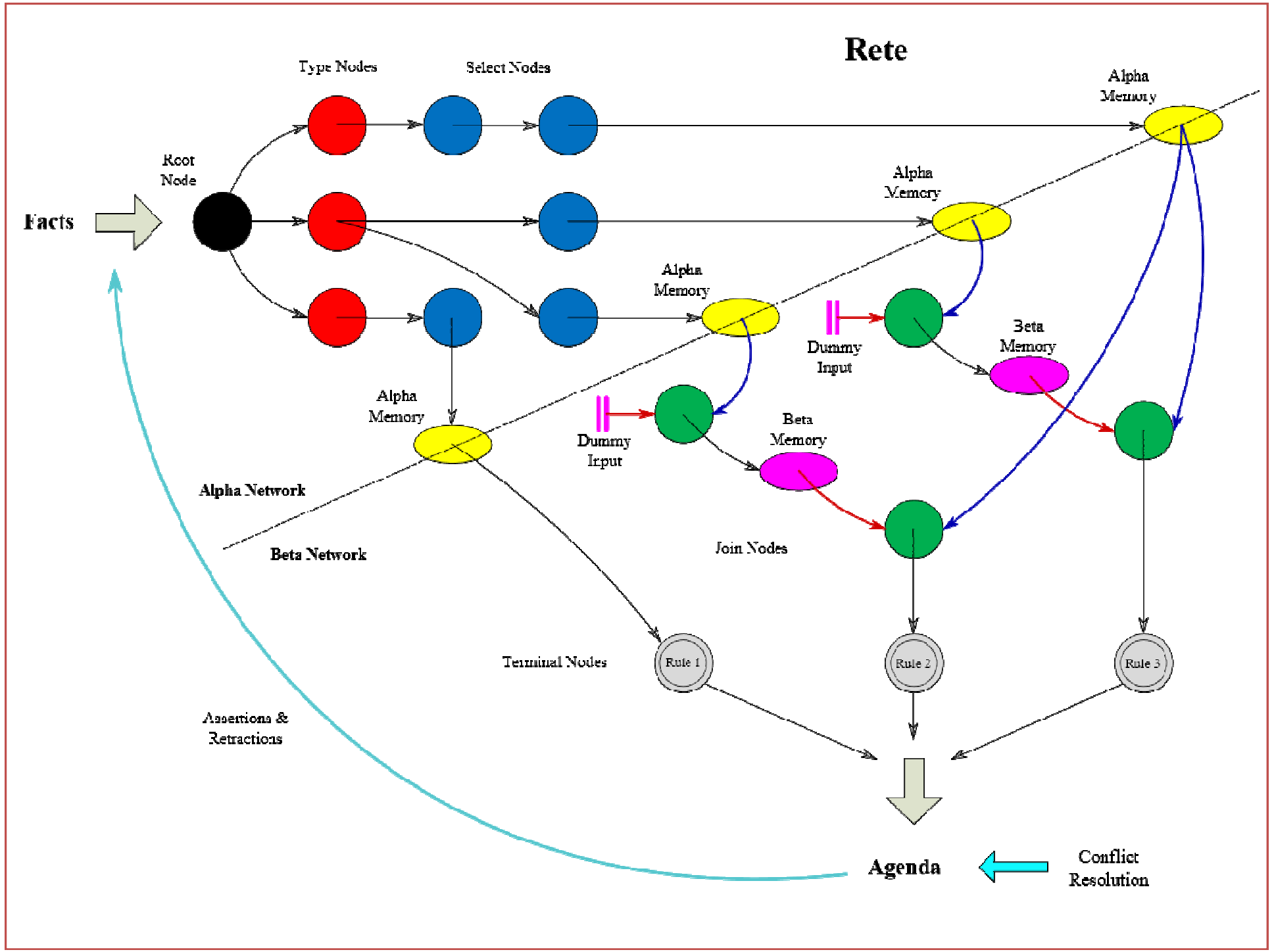
Casamento de Padrões

- Deve ser eficiente para garantir a velocidade na execução das regras
- Muitos algorítmos foram propostos, tais como:
  - Liner
  - Rete
  - Treat
  - Leaps
- CLIPS, BizTalk, Rules Engine, Soar, Jess e Drools usam o Rete

Rete

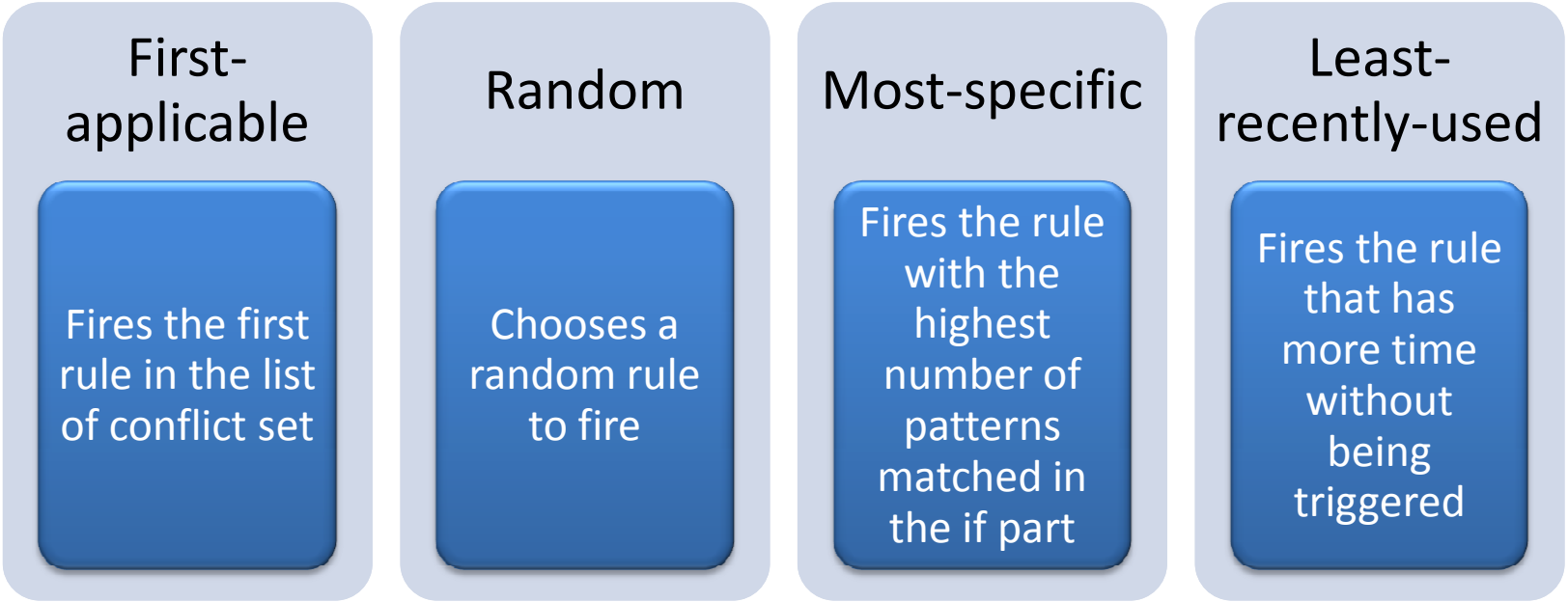
- Grafo direcionado acíclico
- Armazena casamento parcial de padrões
- Compartilha padrões




Rete
Type Nodes
Select Nodes
Alpha Memory
Root Node
Facts
Alpha Memory
Alpha Memory
Dummy Input
Beta Memory
Alpha Memory
Dummy Input
Beta Memory
Alpha Network
Beta Network
Join Nodes
Terminal Nodes
Rule 1
Rule 2
Rule 3
Assertions & Retractions
Agenda
Conflict Resolution

Resolução de Conflitos

- Método utilizado para escolher a regra que deve ser executada primeiro



Drools Conflict Resolution

Salience

- Is the default conflict resolution method.
- The user can specify a priority (or salience) to the rules. The rule with the highest priority is fired first.

LIFO

- The order is given by the assigned Working Memory Action counter value.
- That is, the rules matched due the latest action in the WM are fired first.

The execution order of firings with the same priority is arbitrary.

Próxima aula vamos aprender...

```
rule "AccountUnderObservation"
when

   $osw:OngoingSuspiciousWithdrawal($account:account)
    not exists (CurrentSituation (situation == $osw))
    Account(this==$account)

   exists(DeactivationSituationEvent(situation==$osw)
   over window:time(30d))
then
     SituationHelper.activateSituation(drools,
       kcontext, AccountUnderObservation.class);
```