Forward Chaining using Rete

We already saw two ways to improve FC

- The two improvements we have seen before:
  - Conjunct ordering
  - Incremental Forward Chaining
- Rete takes incremental Forward Chaining to a whole new level
- Rete does not waste partial matches

Definite Clauses vs Production Rules

- All definite clauses can be expressed as production rules. But not the reverse.
- Example of a Definite clause:

  $Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$

- Equivalent Production Rule:
  - IF missile(x) AND Owns(Nono, x) THEN

    Assert(Sells(West, x, Nono))

Definite Clauses vs Production Rules- *Primary difference*

- Production rules supports multiple consequents
- It supports retract and modify
- Example:
  - IF missile(x) AND Owns(Nono, x) AND investigate (FBI, Nono) THEN

    Retract(Sells(West, x, Nono))

    Modify(Owns(Nono,x), Owns(Us, x))
- You just need to be familiar with production rules
  - All definite clauses are valid production rules and in this class, we will only deal with them

Rete Algorithm

- ‘Rete’ stands for ‘Network’ (of blood vessels) in latin
- Rete was designed for working on Production Systems
- it operates on production rules
- Invented by Charles Forgy (1978, 1979 and 1982) for OPS5 system

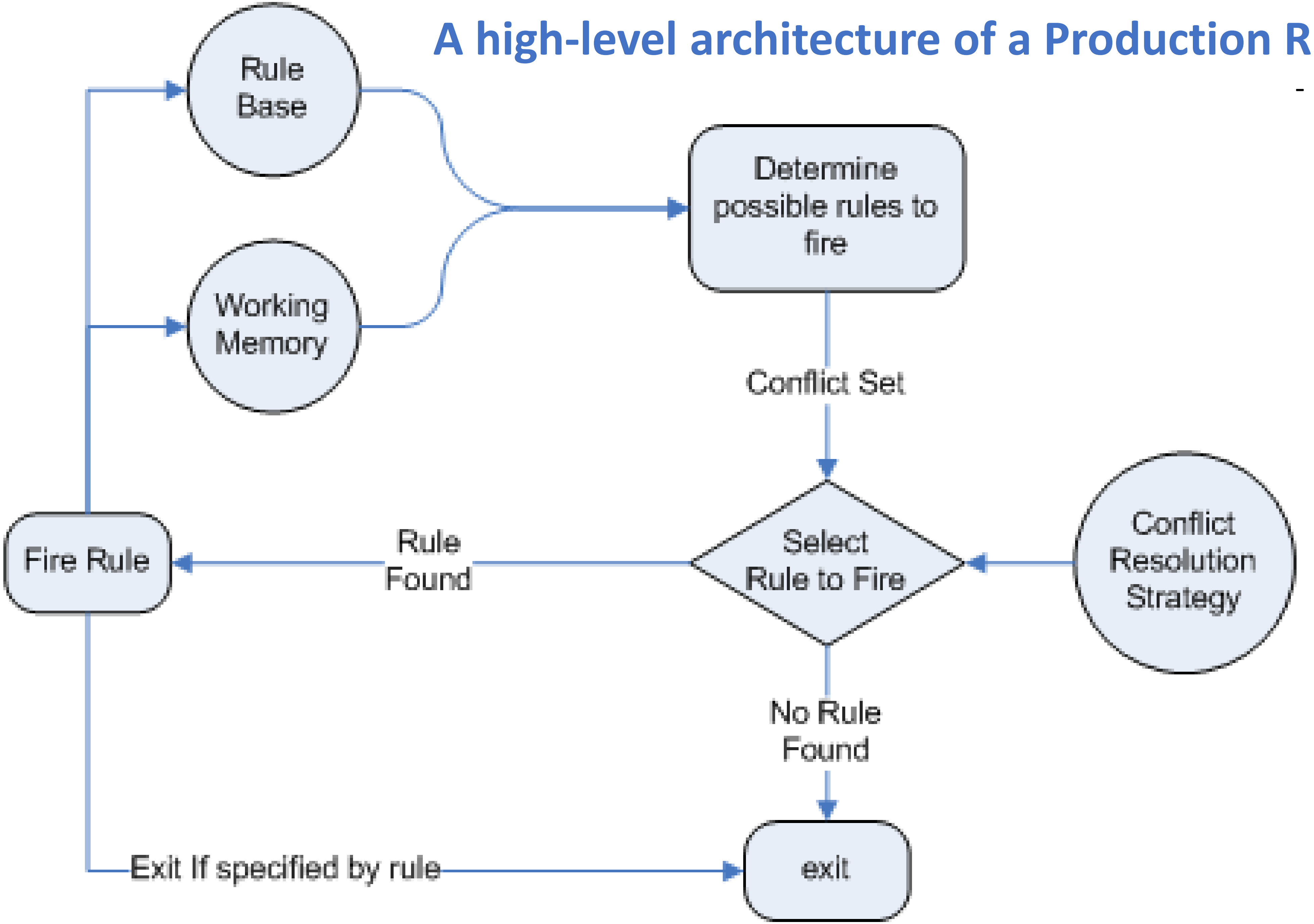
  “A fast algorithm for many pattern/many object pattern match problem”

Rete complexity

- P: Number of rules
- C: Average number of pattern/antecedents in a rule
- W: Number of facts
- The algorithmic complexity is:
  - Best case: O(Log(P))
  - Average Case O(PWC) – linear in the size of working memory
  - Worst Case: $O(PW^{C})$
- Proof/analysis is **left as a exercise**

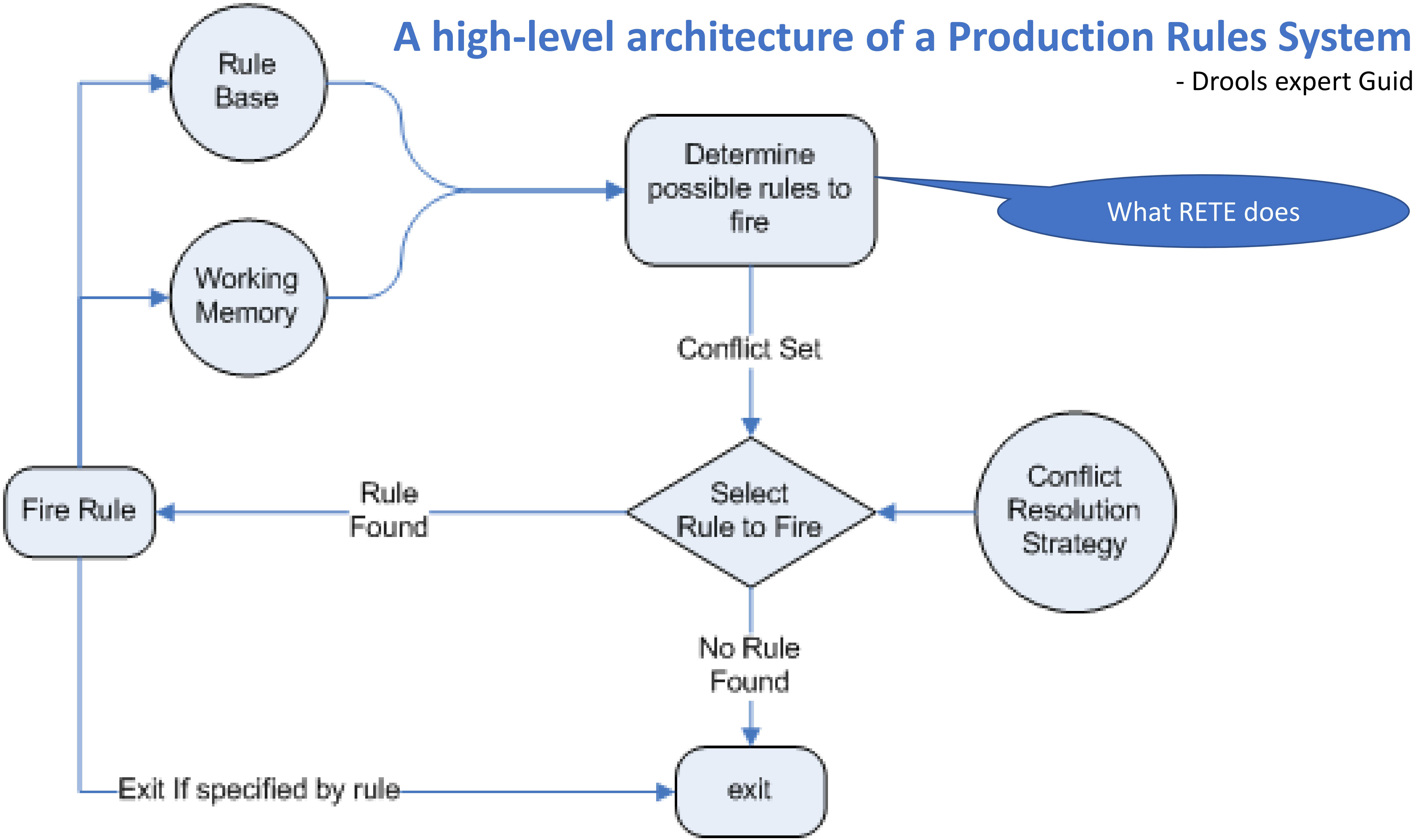
A high-level architecture of a Production Rules System

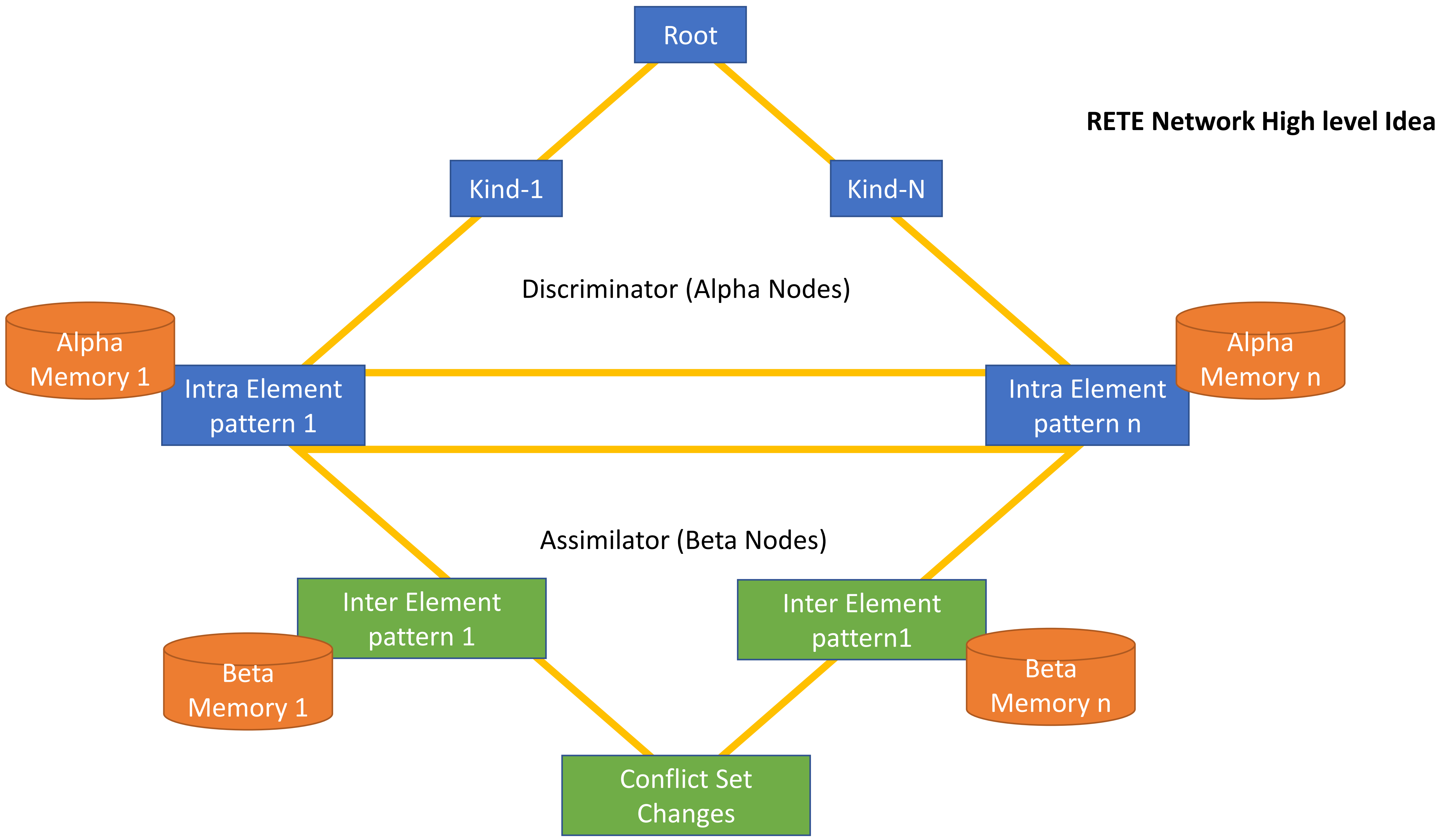
- Drools expert Guide



A high-level architecture of a Production Rules System

- Drools expert Guide





Root
RETE Network High level Idea
Kind-1
Kind-N
Discriminator (Alpha Nodes)
Alpha Memory 1
Intra Element pattern 1
Alpha Memory n
Intra Element pattern n
Assimilator (Beta Nodes)
Inter Element pattern 1
Beta Memory 1
Inter Element pattern1
Beta Memory n
Conflict Set Changes

Our Rule base and Facts

Rule 1: (has-goal ?x simplify)

(expression ?x 0 + ?y)

==> DO SOMETHING

Rule 2: (has-goal ?x simplify)

(expression ?x 0 * ?y)

==> DO SOMETHING

Fact 1: (has-goal e1 simplify)

Fact 3: (has-goal e2 simplify)

Fact 2: (expression e1 0 + 3)

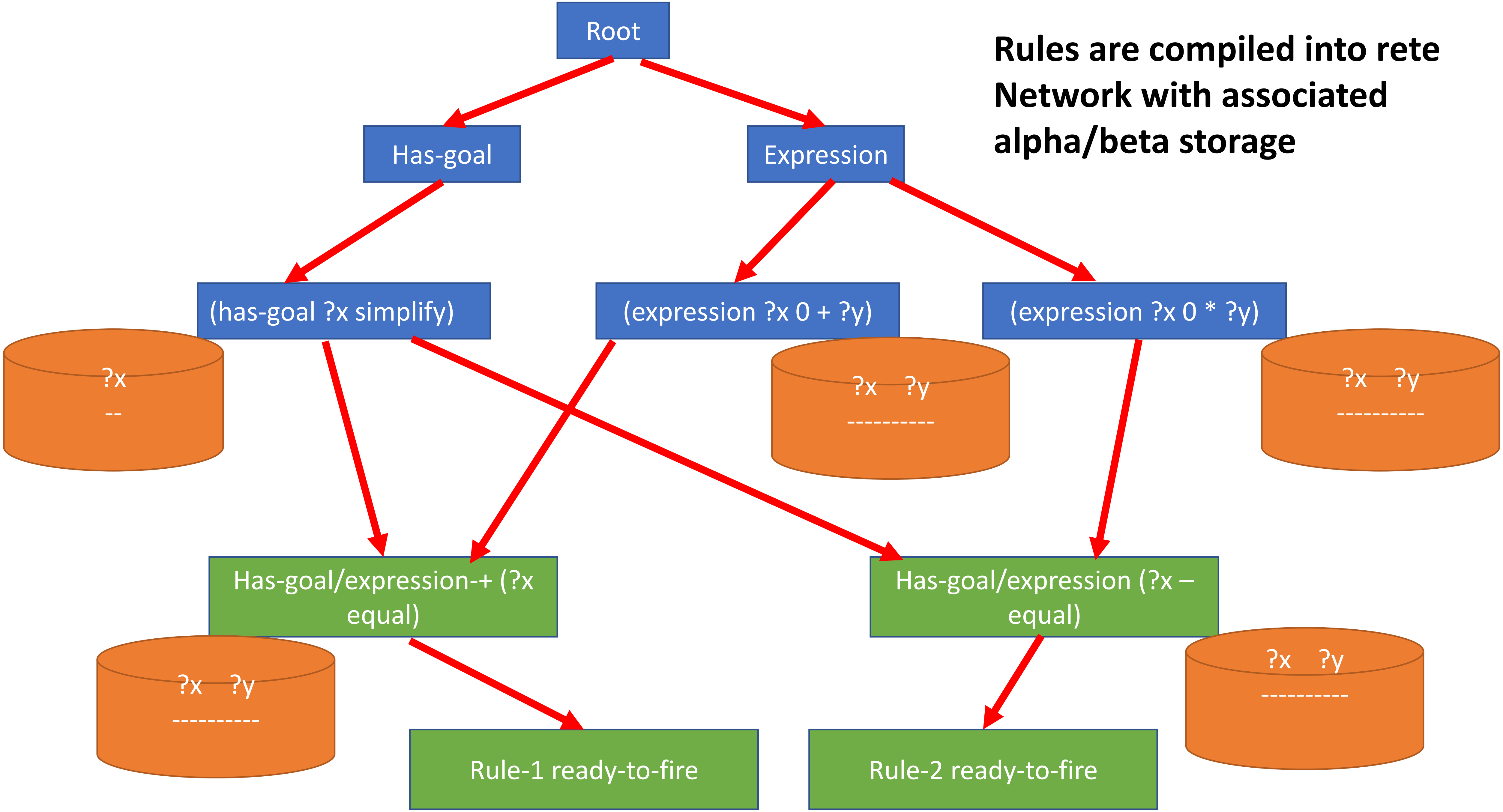
Fact 4: (expression e2 0 + 5)

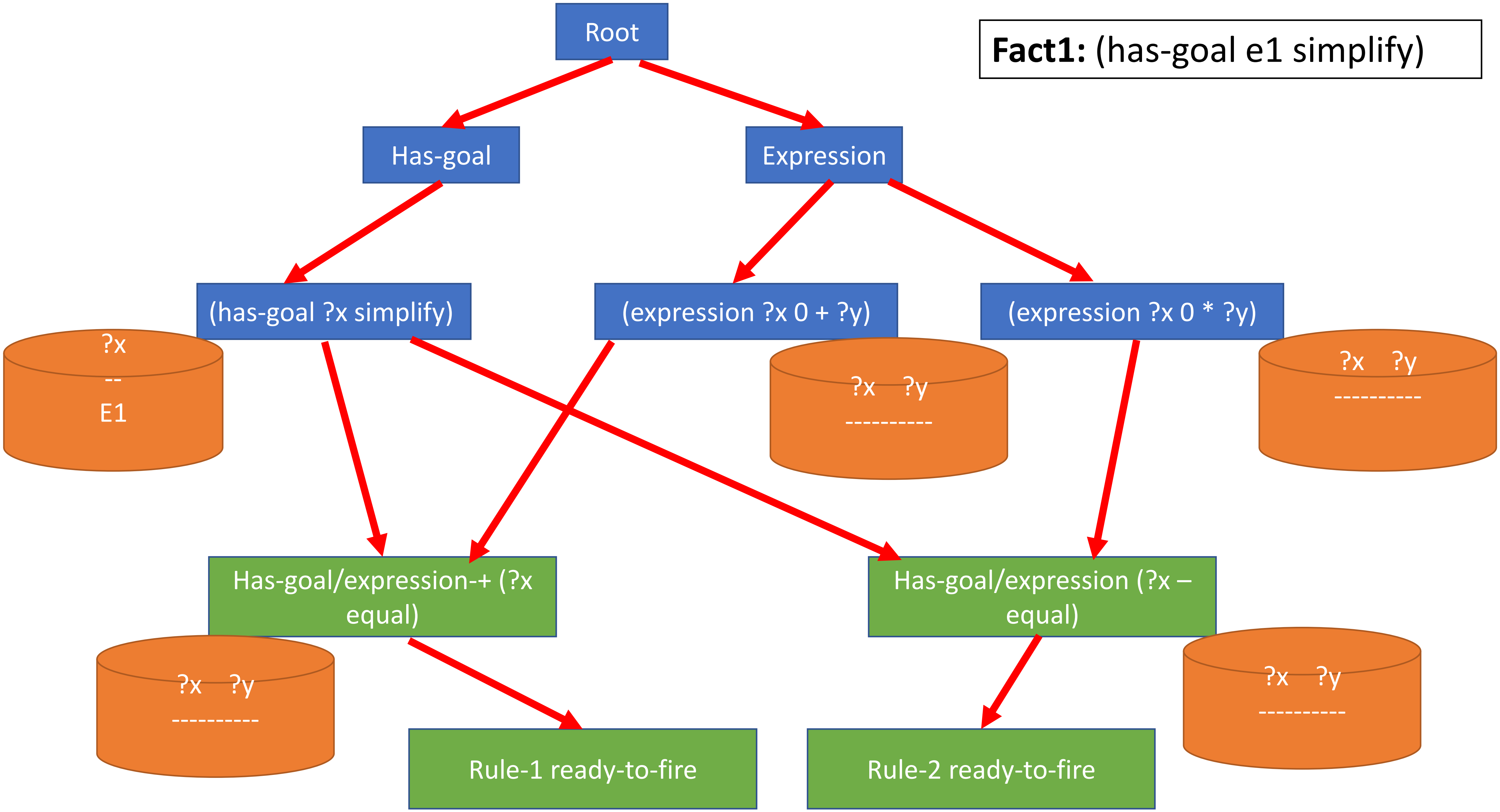
Fact 5: (has-goal e3 simplify)

Fact 6: (expression e3 0 * 2)

(assume that facts will be asserted one at a time)

Rules are compiled into rete Network with associated alpha/beta storage





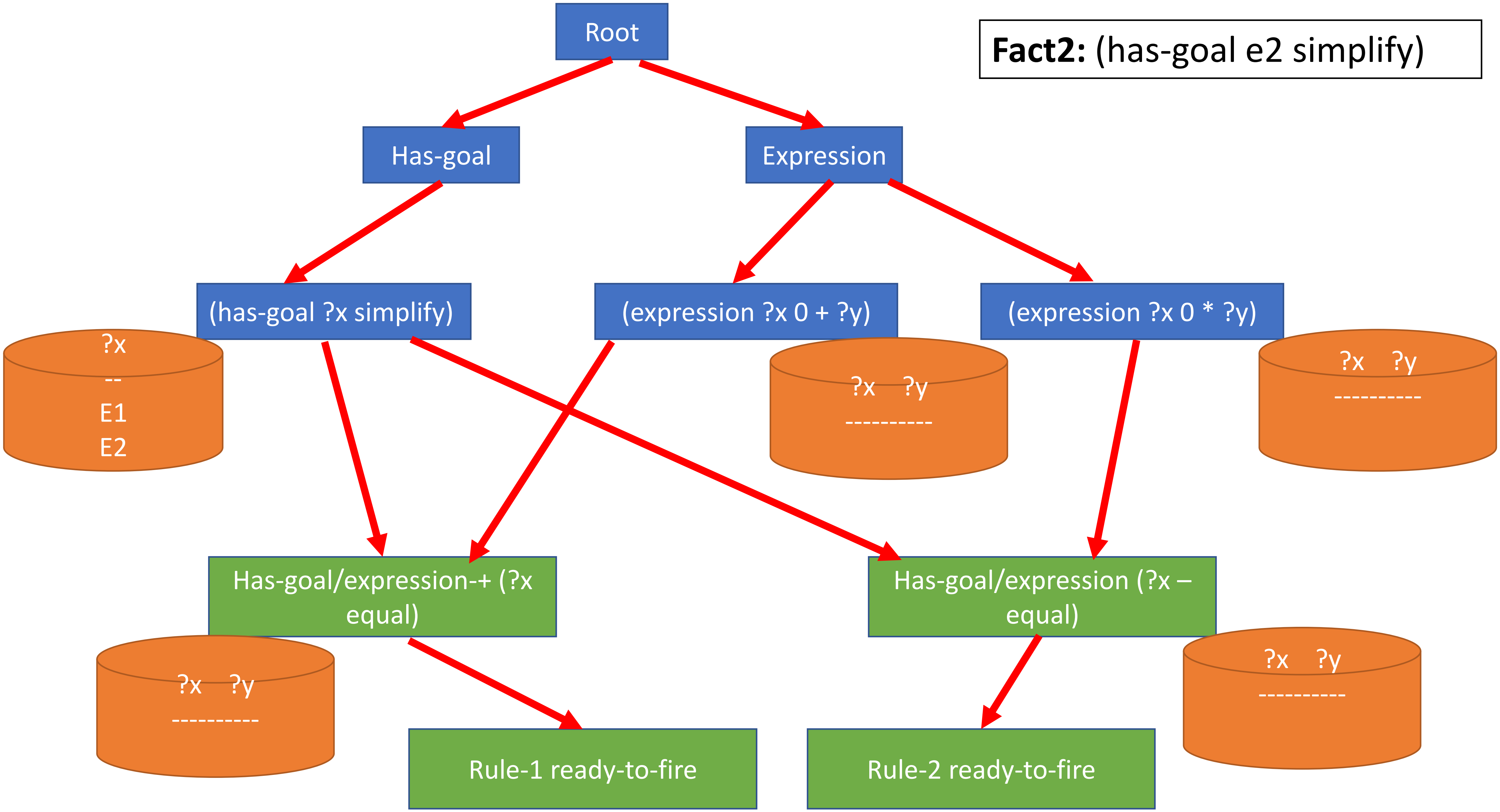
Fact1: (has-goal e1 simplify)
Root
Has-goal
Expression
(has-goal ?x simplify)
?x
--
E1
(expression ?x 0 + ?y)
?x ?y

(expression ?x 0 * ?y)
?x ?y

Has-goal/expression-+ (?x equal)
?x ?y

Has-goal/expression (?x – equal)
?x ?y

Rule-1 ready-to-fire
Rule-2 ready-to-fire

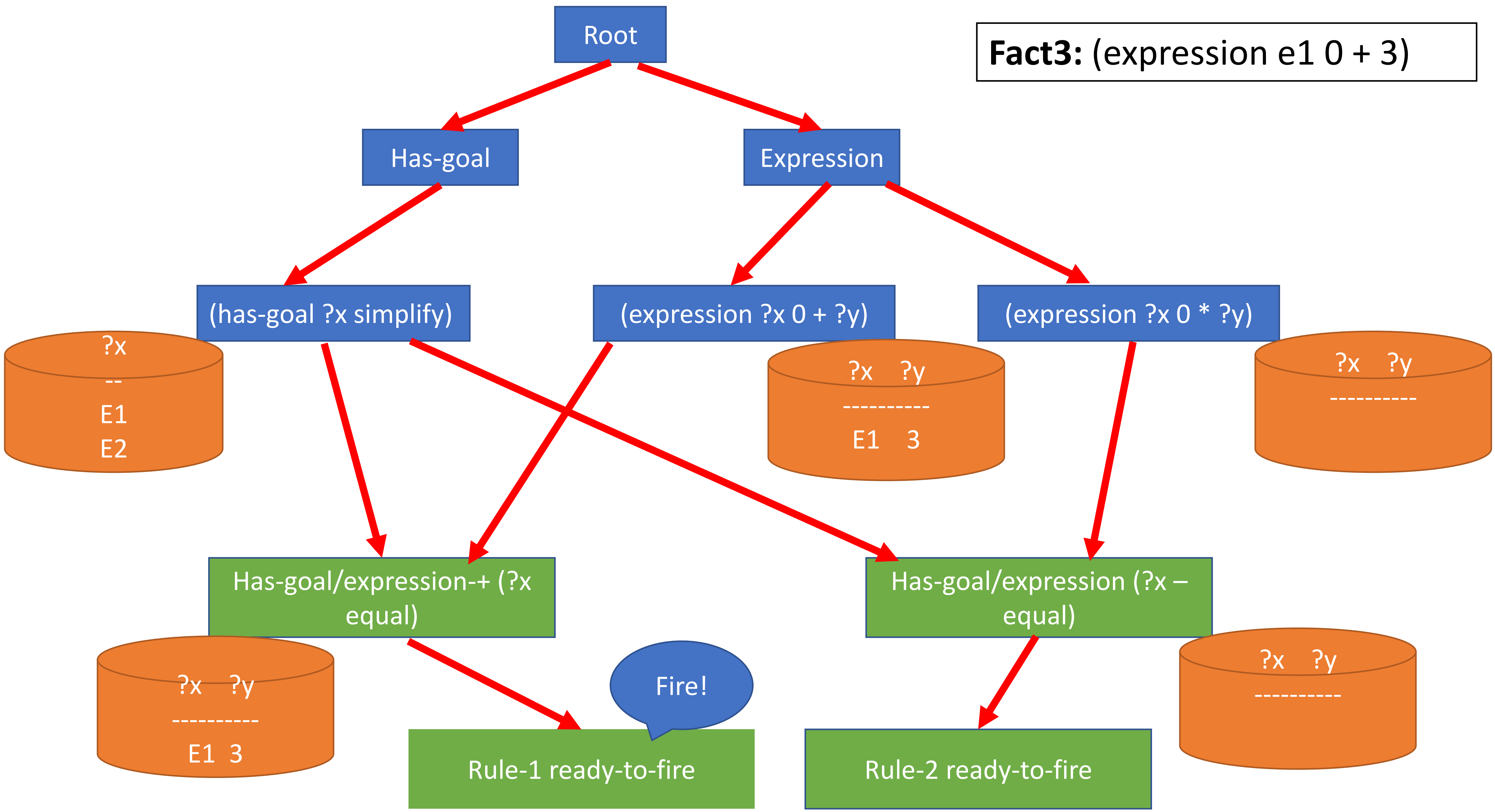

Fact2: (has-goal e2 simplify)
Root
Has-goal
Expression
(has-goal ?x simplify)
(expression ?x 0 + ?y)
(expression ?x 0 * ?y)
?x
--
E1
E2
?x ?y

?x ?y

Has-goal/expression-+ (?x equal)
Has-goal/expression (?x – equal)
?x ?y

?x ?y

Rule-1 ready-to-fire
Rule-2 ready-to-fire

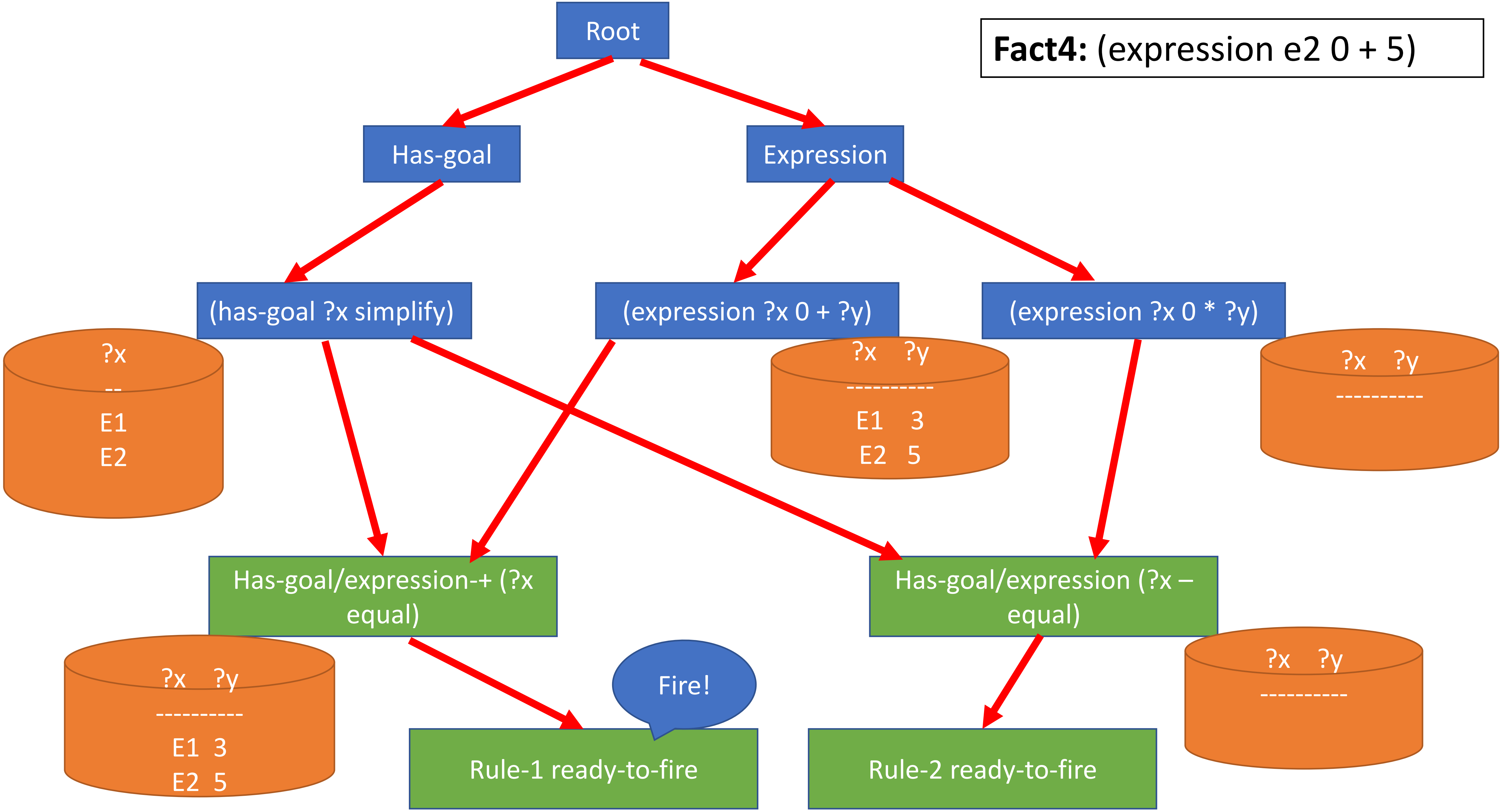

Fact3: (expression e1 0 + 3)
Root
Has-goal
Expression
(has-goal ?x simplify)
(expression ?x 0 + ?y)
(expression ?x 0 * ?y)
?x
--
E1
E2
?x ?y

E1 3
?x ?y

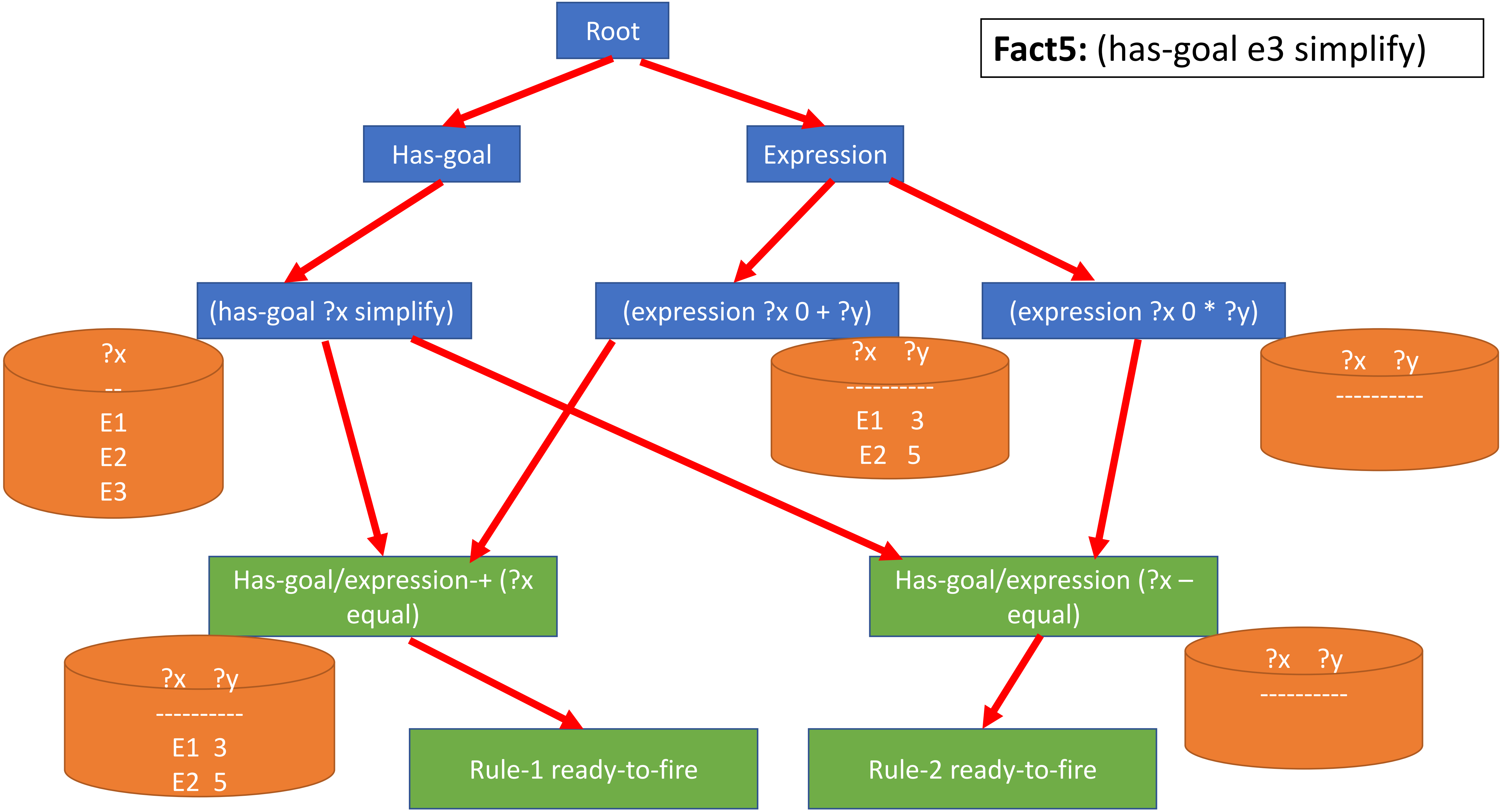
Has-goal/expression-+ (?x equal)
Has-goal/expression (?x – equal)
?x ?y

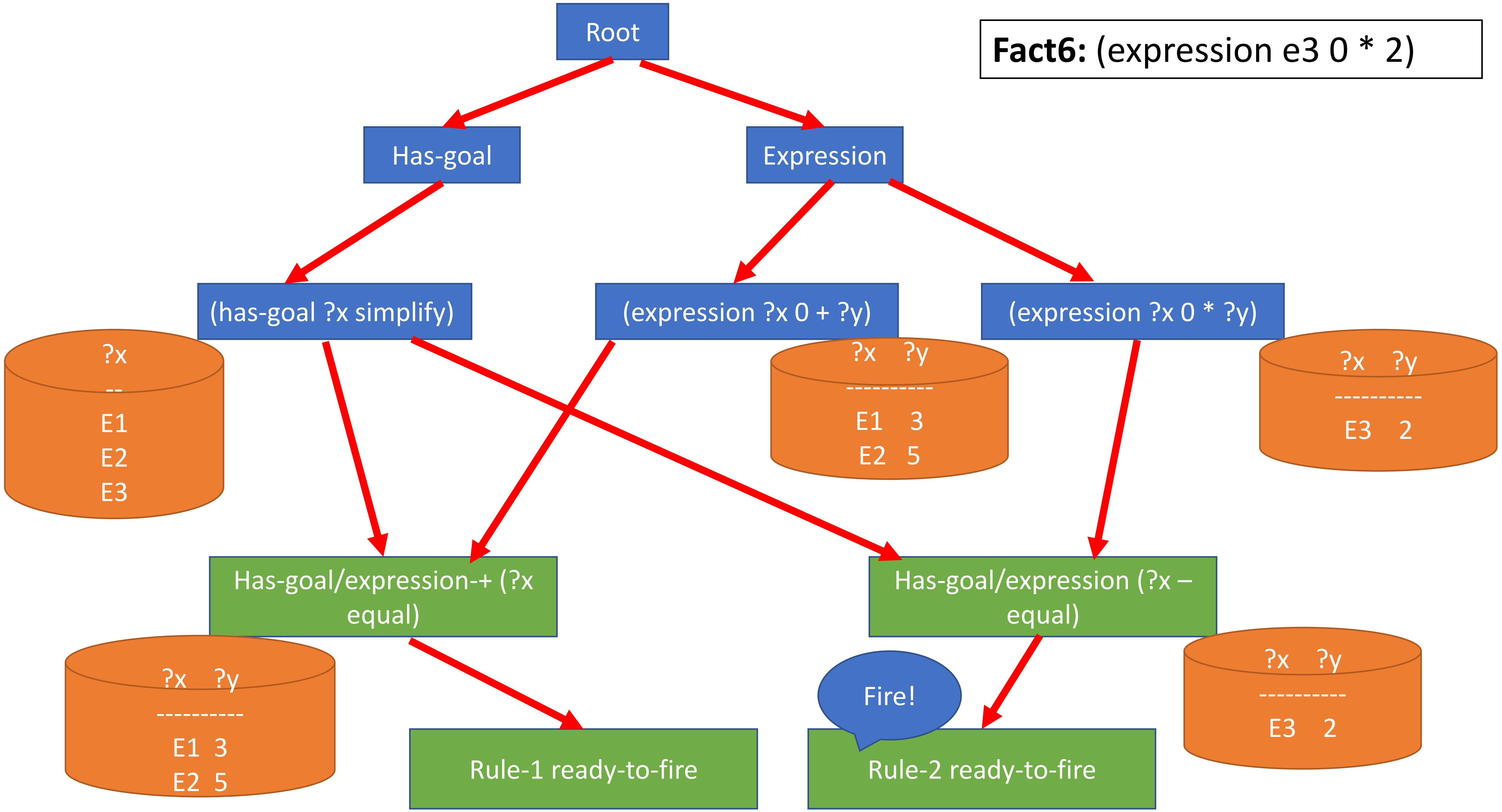
E1 3
Fire!
Rule-1 ready-to-fire
Rule-2 ready-to-fire
?x ?y

Fact4: (expression e2 0 + 5)



Fact5: (has-goal e3 simplify)





Fact6: (expression e3 0 * 2)
Root
Has-goal
Expression
(has-goal ?x simplify)
(expression ?x 0 + ?y)
(expression ?x 0 * ?y)
?x
--
E1
E2
E3
?x ?y

E1 3
E2 5
?x ?y

E3 2
Has-goal/expression-+ (?x equal)
Has-goal/expression (?x – equal)
?x ?y

E1 3
E2 5
Rule-1 ready-to-fire
Fire!
Rule-2 ready-to-fire
?x ?y

E3 2

FOL Resolution

Substitutions

- A substitution is a finite set $\{t_1 / V_1, \cdot\cdot\cdot, t_n / V_n\}$
  - V_i is a variable
  - t_i is a term, different from vi
- No two elements in the set have the same variable after the '/' symbol.
- Please remember many books seem to contradict on whether its 't/v' or 'v/t' . There is a lack of uniformity.

Example Substitutions

- f (z)/x, y/z} is a substitution
- {a/x, g(y)/y, f (g(b))/z} is a substitution
- {y/x, g(b)/y} is a substitution
- {a/x, g(y)/x, f (g(b))/z} is *not* a substitution
- {g(y)/x, z/f (g(b))} is *not* a substitution

Most general unifier

- A *most general unifier (mgu)* of a set S of expressions is a unifier of θ of S such that any other unifier σ of S can be written as $\sigma = \theta\alpha$ for some substitution α.
- *Example*. Let $S = \{p(x, a), p(y, z)\}$. The unifiers of S are

  $\{x/y, z/a\}$ and $\{y/x, z/a\}$ and $\{x/t, y/t, z/a\}$ for any term t.
- The unifier $\{x/y, z/a\}$ is an mgu for S because

  $\{y/x, z/a\} = \{x/y, z/a\}\{y/x\}$ and $\{x/t, y/t, z/a\} = \{x/y, z/a\}\{y/t\}$.

Most general unifier

- A *most general unifier (mgu)* of a set S of expressions is a unifier of θ of S such that any other unifier σ of S can be written as $\sigma = \theta\alpha$ for some substitution α.
- *Example*. Let $S = \{p(x, a), p(y, z)\}$. The unifiers of S are

  $\{x/y, z/a\}$ and $\{y/x, z/a\}$ and $\{x/t, y/t, z/a\}$ for any term t.
- The unifier $\{x/y, z/a\}$ is an mgu for S because

  $\{y/x, z/a\} = \{x/y, z/a\}\{y/x\}$ and $\{x/t, y/t, z/a\} = \{x/y, z/a\}\{y/t\}$.

Change FOL to CNF

- Change to CNF (But need to handle quantifiers)
- Standardize Variables
- Universal quantifiers can be left alone
- Existential quantifiers need to be ***skolemized*** (examples in Book)

$$\forall x \;\; American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \;\Rightarrow\; Criminal(x)$$

becomes, in CNF,

$$\neg American(x) \vee \neg Weapon(y) \vee \neg Sells(x, y, z) \vee \neg Hostile(z) \vee Criminal(x)\,.$$

Change FOL to CNF

$$\forall x\ \ [\exists y\ \ Animal(y) \land \neg Loves(x,y)] \lor [\exists z\ \ Loves(z,x)]\ .$$

$$\forall x\ \ [Animal(A) \land \neg Loves(x,A)] \lor Loves(B,x)\ ,$$

which has the wrong meaning entirely: it says that everyone either fails to love a particular animal A or is loved by some particular entity B. In fact, our original sentence allows each person to fail to love a different animal or to be loved by a different person. Thus, we want the Skolem entities to depend on x and z:

$$\forall x\ \ [Animal(F(x)) \land \neg Loves(x,F(x))] \lor Loves(G(z),x)\ .$$

Resolution inference rule

$$\frac{\ell_1 \vee \cdots \vee \ell_k, \qquad m_1 \vee \cdots \vee m_n}{\textsc{Subst}(\theta, \ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n)}$$

where $\textsc{Unify}(\ell_i, \neg m_j) = \theta$. For example, we can resolve the two clauses

$$[Animal(F(x)) \vee Loves(G(x), x)] \quad \text{and} \quad [\neg Loves(u, v) \vee \neg Kills(u, v)]$$

by eliminating the complementary literals $Loves(G(x), x)$ and $\neg Loves(u, v)$, with unifier $\theta = \{u/G(x), v/x\}$, to produce the **resolvent** clause

$$[Animal(F(x)) \vee \neg Kills(G(x), x)] .$$

Resolution example

a. $\forall x$: food(x) $\rightarrow$ likes(John, x)

b. food(Apple) $\wedge$ food(vegetables)

c. $\forall x\ \forall y$: eats(x, y) $\wedge \neg$ killed(x) $\rightarrow$ food(y)

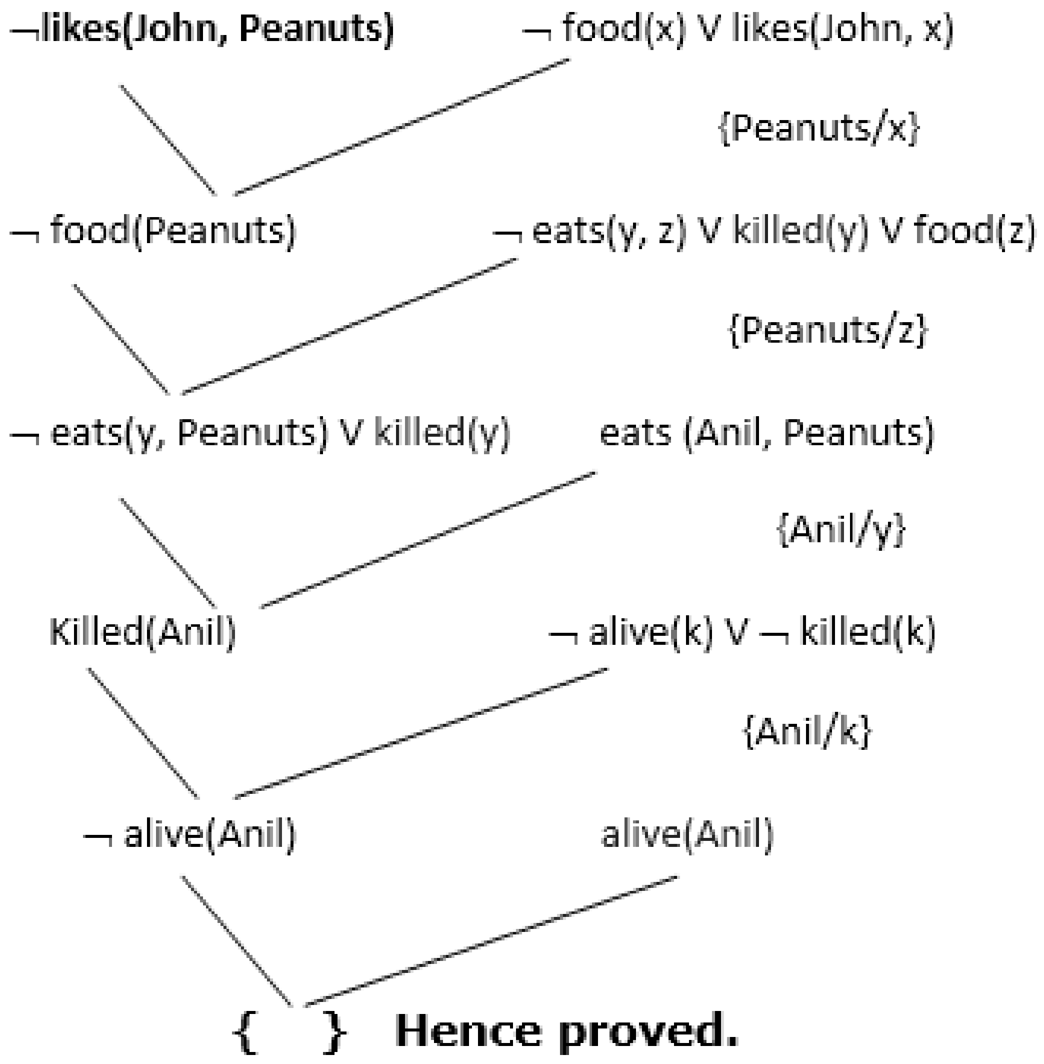
d. eats (Anil, Peanuts) $\wedge$ alive(Anil).

e. $\forall x$: eats(Anil, x) $\rightarrow$ eats(Harry, x)

f. $\forall x$: $\neg$ killed(x) $\rightarrow$ alive(x) **added predicates.**

g. $\forall x$: alive(x) $\rightarrow \neg$ killed(x)

Resolution example



Resolution example

$$\neg American(x) \vee \neg Weapon(y) \vee \neg Sells(x, y, z) \vee \neg Hostile(z) \vee Criminal(x)$$
$$\neg Missile(x) \vee \neg Owns(Nono, x) \vee Sells(West, x, Nono)$$
$$\neg Enemy(x, America) \vee Hostile(x)$$
$$\neg Missile(x) \vee Weapon(x)$$
$$Owns(Nono, M_1) \qquad Missile(M_1)$$
$$American(West) \qquad Enemy(Nono, America) .$$

Resolution example