Programowanie w CLIPS

CLIPS - środowisko open source do tworzenia systemów ekspertowych. Stworzone w NASA - Johnson Space Center.

System ekspertowy - jest to program lub zestaw programów komputerowych wspomagający korzystanie z wiedzy i ułatwiający podejmowanie decyzji. Systemy ekspertowe mogą wspomagać bądź zastępować ludzkich ekspertów w danej dziedzinie, mogą dostarczać rad, zaleceń i diagnoz dotyczących problemów tej dziedziny.

Z czego składa się system ekspertowy?

1. **Explanation facility** – wyjaśnia proces wnioskowania, który doprowadził do danej konkluzji.
2. **Knowledge acquisition facility** – dostarcza sposobów do pozyskania i przechowywania wiedzy pozyskanej od eksperta w bazie wiedzy (knowledge base).
3. **Knowledge base** - przechowuje wiedzę w postaci reguł.
4. **Working memory** - baza faktów wykorzystywanych przez reguły.
5. **Inference engine** - decyduje które reguły, i w jakiej kolejności zostaną uruchomione.
6. **Agenda** - lista reguł, których warunki spełnione są przez fakty znajdujące się w pamięci roboczej (working memory)
7. **Pattern matcher** - porównuje reguły i fakty.

Z czego składa się system ekspertowy?

1. **Explanation facility** – wyjaśnia proces wnioskowania, który doprowadził do danej konkluzji.
2. **Knowledge acquisition facility** – dostarcza sposobów do pozyskania i przechowywania wiedzy pozyskanej od eksperta w bazie wiedzy (knowledge base).
3. **Knowledge base** - przechowuje wiedzę w postaci reguł.
4. **Working memory** - baza faktów wykorzystywanych przez reguły.
5. **Inference engine** - decyduje które reguły, i w jakiej kolejności zostaną uruchomione.
6. **Agenda** - lista reguł, których warunki spełnione są przez fakty znajdujące się w pamięci roboczej (working memory)
7. **Pattern matcher** - porównuje reguły i fakty.

Fakty i reguły

Fakt: „Stal jest stopem żelaza i węgla“
Reguła: „Jeśli w stali jest domieszka niklu to hartowanie jest ułatwione“

Fakt: „Adam jest studentem metalurgii“
Reguła: „Jeśli student jest studentem metalurgii to wie co to jest hartowanie“

Programowanie w CLIPS

Fakty + Reguły

Programowanie w CLIPS

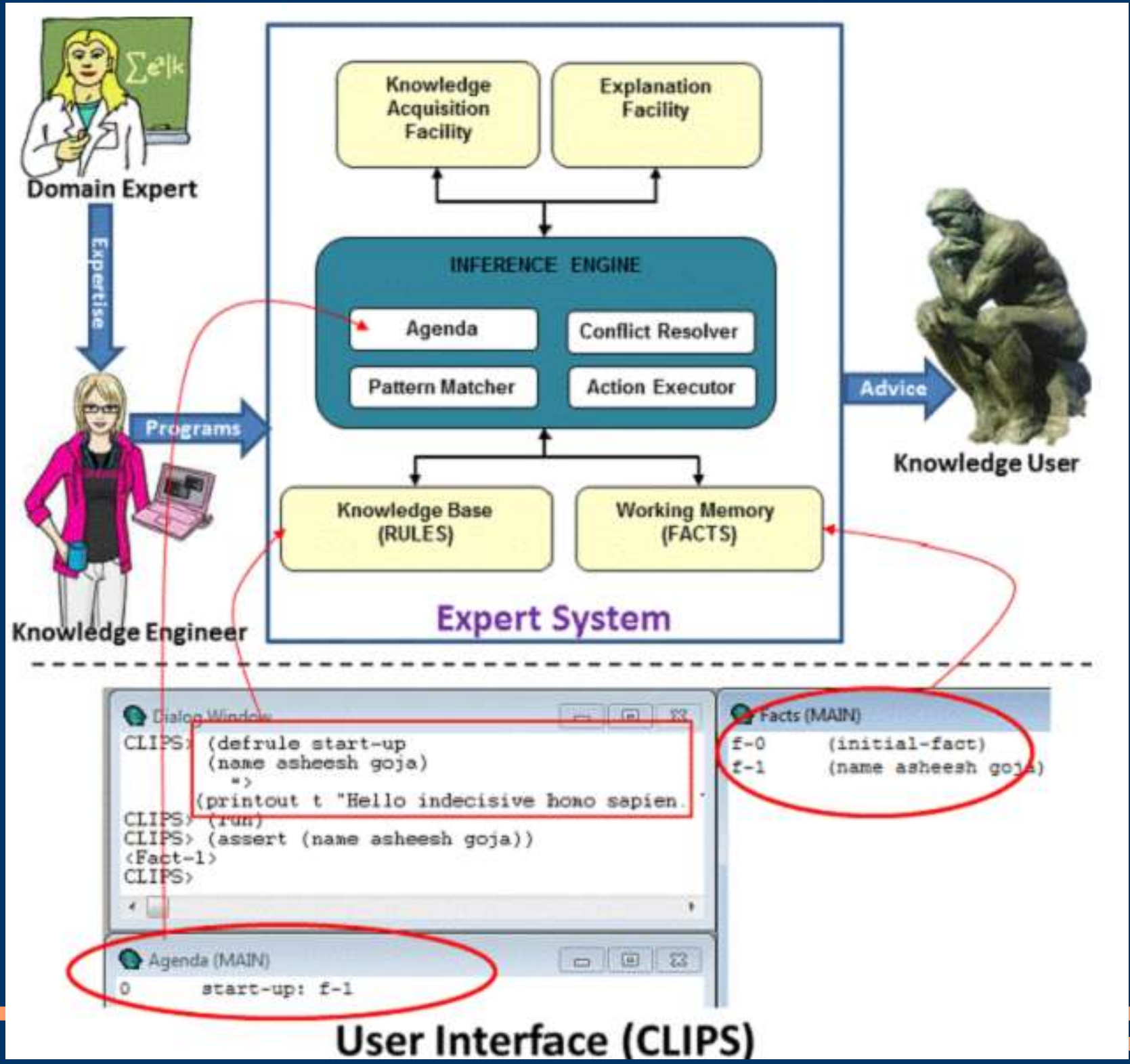
Fakty + Reguły

Fakt:

```
(assert (light-color green))
```

Reguła:

```
(defrule is-driving-possible
    (light-color green)
    =>
    (assert (driving possible))
```



User Interface (CLIPS)

Programowanie w CLIPS

Fakty:

- niemodyfikowalne (modyfikacja faktu oznacza jego usunięcie i stworzenie nowego)
- mogą być tworzone
- mogą byś usuwane

Reguły:

- składają się z „poprzedników“ (LHS) i „następstw“ (RHS)
- reguła zostaje uruchomiona gdy spełniony jest jest poprzednik/poprzednicy (LHS)
- następstwem (RHS) reguły jest najczęściej modyfikacja lub dodanie nowych faktów do pamięci roboczej (working memory)

Programowanie w CLIPS

Produkcja faktów:

Fakty uruchamiają reguły, które produkują nowe fakty, które uruchamiają reguły, które... itd. itd.

Przypomnienie:
Agenda - lista reguł, których warunki spełnione są przez fakty znajdujące się w pamięci roboczej (working memory).

Program kończy się gdy na agendzie nie ma już reguł. Wynikiem działania programu jest lista faktów w pamięci roboczej.

Programowanie w CLIPS

Wiedza reprezentowana jest przez:

- unordered facts (fakty nieuporządkowane)
- ordered facts (fakty uporządkowane)
- rules (reguły)

Fakty

Fakty uporządkowane (ordered facts):

(jan kowalski)
(pogoda słoneczna)
(stal niklowana)
(data-godzina 12.12.2011 19:30)
(lista-numerow 2 32 3 5 6 7 34 32 4)

Fakty nieuporządkowane (unordered facts):

```
(person  (name Jan Kowalski)
         (age 23 )
         (eye-color blue)
         (hair-color brown))
```

Fakty

W przypadku faktów uporządkowanych by odnieść się do danego pola/symbolu należy znać jego pozycję.

W przypadku faktów nieuporządkowanych używa się nazwy slotu.

Fakty

Fakty nieuporządkowane (unordered facts):

Struktura (definicja faktu):

Typ faktu: person

```
(deftemplate person "An example deftemplate"
        (slot name)
        (slot age)
        (slot eye-color)
        (slot hair-color))
```

Fakty

Fakty nieuporządkowane (unordered facts):

Tworzenie (dodawanie) faktów:

```
(assert (person (name John Q. Public)
                (age 23 )
                (eye-color blue)
                (hair-color brown)))
```

```
(deftemplate question
        (slot factor (default none))
        (slot question-to-ask (default none))
        (slot has-pre-condition (type SYMBOL) (default no))
        (multislot choices (default yes no))
        (multislot range (type INTEGER)))
```

Słowa kluczowe:

- **deftemplate**: definicja faktu
- **slot**: pojedynczy symbol
- **multislot**: wiele symboli
- **type**: typ symbolu
- **default**: wartość domyślna

Fakty

Dodawanie faktów:
(assert <*fakt*>)

Usuwanie faktów:
(retract <*indeks-faktu*>)

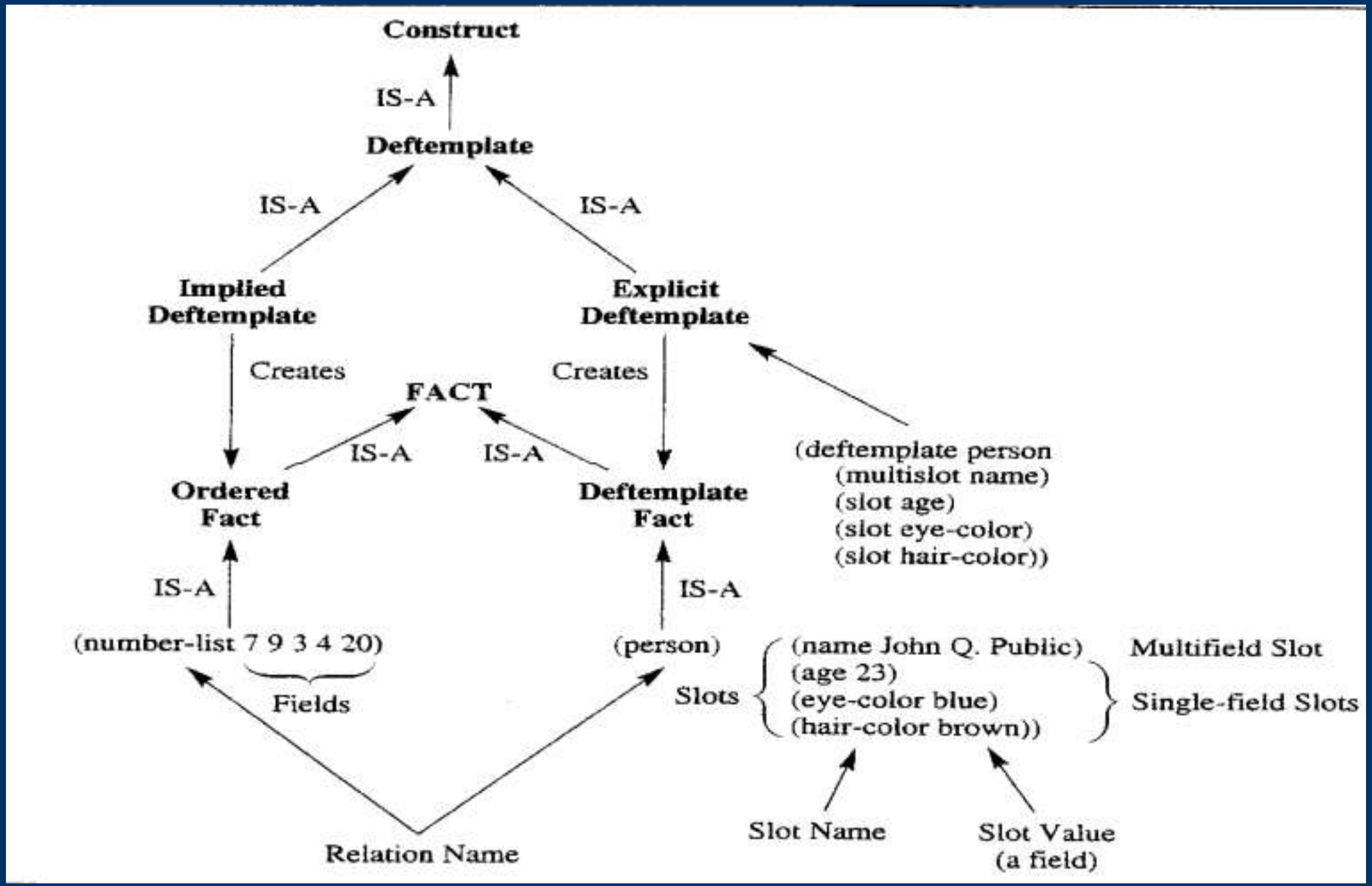
Modyfikowanie (uwaga: = usunięcie i dodanie nowego)
(modify <indeks-faktu> <slot-modifier>+)
gdzie <slot-modifier> to:
(<slot-name> <slot-value>)
np. (modify 0 (age 23) (name Adam Kowalski))

Fakty

Wyświetlenie faktów:
(facts)

Usunięcie wszystkich faktów i ponowne dodanie tych z deffacts (o tym zaraz):
(reset)

Usunięcie faktów oraz reguł!:
(clear)


Construct
IS-A
Deftemplate
IS-A
IS-A
Implied Deftemplate
Explicit Deftemplate
Creates
Creates
FACT
IS-A
IS-A
Ordered Fact
Deftemplate Fact
(deftemplate person
(multislot name)
(slot age)
(slot eye-color)
(slot hair-color))
IS-A
IS-A
(number-list 7 9 3 4 20)
Fields
(person)
Slots
(name John Q. Public)
(age 23)
(eye-color blue)
(hair-color brown))
Multifield Slot
Single-field Slots
Slot Name
Slot Value
(a field)
Relation Name

Fakty

„Hurtowe“ dodawanie faktów danego typu (unordered facts):

```
(deffacts people "Znajomi"
    (person (name "Adam Kowalski") (age 24)
            (eye-color blue) (hair-color black))
    (person (name "Jan Kowalski") (age 24)
            (eye-color blue) (hair-color black))
    (person (name "Katarzyna Nowak") (age 36)
            (eye-color green) (hair-color red)))
```

Przydatne, gdy chcemy na początku programu dodać zbiór faktów, o których wiemy, że są prawdziwe (tzw. Initial knowledge)

Reguły

Składnia:

```
(defrule nazwa
    Left-Hand-Side (LHS) (antecedent/poprzednik)
=>
    Right-Hand-Side (RHS) (consequent/konsekwencja)
)
```

LHS – musi być spełnione (true), by reguła została umieszczona na agendzie.
RHS – określa czynności wykonywane przez regułę.

Reguły

```
IF
   LHS
THEN
   RHS
```

Reguły

IF alarm pożarowy
THEN podejmij-akcje uaktywnij zraszacze

Uwaga: przed zdefiniowaniem reguły wszystkie używane przez nią fakty nieuporządkowane (deftemplate) muszą być zdefiniowane.

```
(deftemplate alarm (slot typ))
(deftemplate wykonaj-akcje (slot akcja))

(defrule akcja-p.pożarowa “Przykład reguły"
    (alarm (typ pożarowy))
=>
    (assert (podejmij-akcje (akcja uaktywnij-zraszacze)))
)
```

Reguły

```
(defrule akcja-p.pożarowa “Przykład reguły"
    (alarm (typ pożarowy))
=>
    (assert (wykonaj-akcje (akcja uaktywnij-zraszacze)))
)
```

Czy tak stworzona reguła zostanie umieszczona na agendzie (wykonana)? Nie. Stanie się to dopiero, gdy dodamy fakt:

```
(assert alarm (typ pożarowy))
```

Rezultat wykonania reguły? Dodanie (produkcja) nowego faktu:

```
(wykonaj-akcje (akcja uaktywnij-zraszacze))
```

Reguły

Składnia LHS:

```
(defrule nazwa
    (and
        (warunek 1)
        (warunek 2) …
    )
=>
    …

(defrule nazwa
    (or
        (warunek 1)
        (warunek 2) …
    )
=>
    ...
```

Reguły

Składnia RHS:

```
(defrule nazwa
    ...
=>
    (tworzenie/usuwanie/modyfikowanie faktu; wypisanie; ..)
    (tworzenie/usuwanie/modyfikowanie faktu; wypisanie; ..)
    ...
```

Reguły

Ciąg zdarzeń:

```
(defrule akcja-p.pożarowa “Przykład reguły"
    (alarm (typ pożarowy))
=>
    (assert (wykonaj-akcje (akcja uaktywnij-zraszacze)))
)

(defrule uruchamianie-zraszaczy
    (wykonaj-akcje (akcja uaktywnij-zraszacze)
    (prąd wyłączony)
=>
    (assert (uruchom dopływ wody))
    (assert (odblokuj zraszacze))
```

Reguły

Zmienne reguł i zakresy:

```
(defrule start-up
   (osoba (wiek ?a&:(> ?a 30)&:(< ?a 40)) (imie $?n) )
    =>
   (printout t "Witaj " ?n ". Masz" ?a "lat" crlf))
```

Zmienne:

?a – jeden slot, ?n – wiele slotów (multislot), bo zdefiniowana jako $?n (pole imie w deftemplate osoba jest multislot – może mieć wiele symboli)

Zakres:

?a&:(> ?a 30)&:(< ?a 40) – zmienna ?a, taka że ?a > 30 oraz ?a < 40

Reguły

```
(defrule start-up
   (osoba (wiek ?a&:(> ?a 30)&:(< ?a 40)) (imie $?n) )
    =>
   (printout t "Witaj " ?n ". Masz" ?a "lat" crlf))
```

Reguła nie zadziała:

```
(assert (osoba (wiek 22) (imie Jan Kowalski))
```

Reguła zadziała:

```
(assert (osoba (wiek 34) (imie Jan Kowalski))
```

Reguły

Modyfikowanie i usuwanie faktów:

(retract <*indeks-faktu*>)
(modify <*indeks-faktu*> …)

Jak wewnątrz reguły otrzymać indeks faktu?

Reguły

```
(deftemplate parametry (slot temperatura) (slot czas))

(defrule obniz-temperature-hartowania
    ?p <- (parametry (temperatura ?temperatura) (czas ?czas) )
    (stop-to stal niklowana)
=>
    (modify ?p (temperatura (- ?temperatura 200) ) )
    (assert (hartowanie))
)
```

Pobranie indeksu faktu: ?p <- (parametry …)

Analogicznie usuwanie faktu:

```
    (retract ?p)
```

Uwaga na notację!!!:
NIE : (a + b)
TAK : (+ a b)

Skąd CLIPS wie, które reguły w danym momencie ma uruchomić?

Przypomnienie:

3. **Knowledge base** - przechowuje wiedzę w postaci reguł.
4. **Working memory** - baza faktów wykorzystywanych przez reguły.
5. **Inference engine** - decyduje które reguły, i w jakiej kolejności zostaną uruchomione.
6. **Agenda** - lista reguł, których warunki spełnione są przez fakty znajdujące się w pamięci roboczej (working memory)
7. **Pattern matcher** - porównuje reguły i fakty.

Uruchamianie programu:

(run)