

Introduction to CLIPS



- Overview of CLIPS
- Facts
- Rules
- Rule firing
- Control techniques
- Example

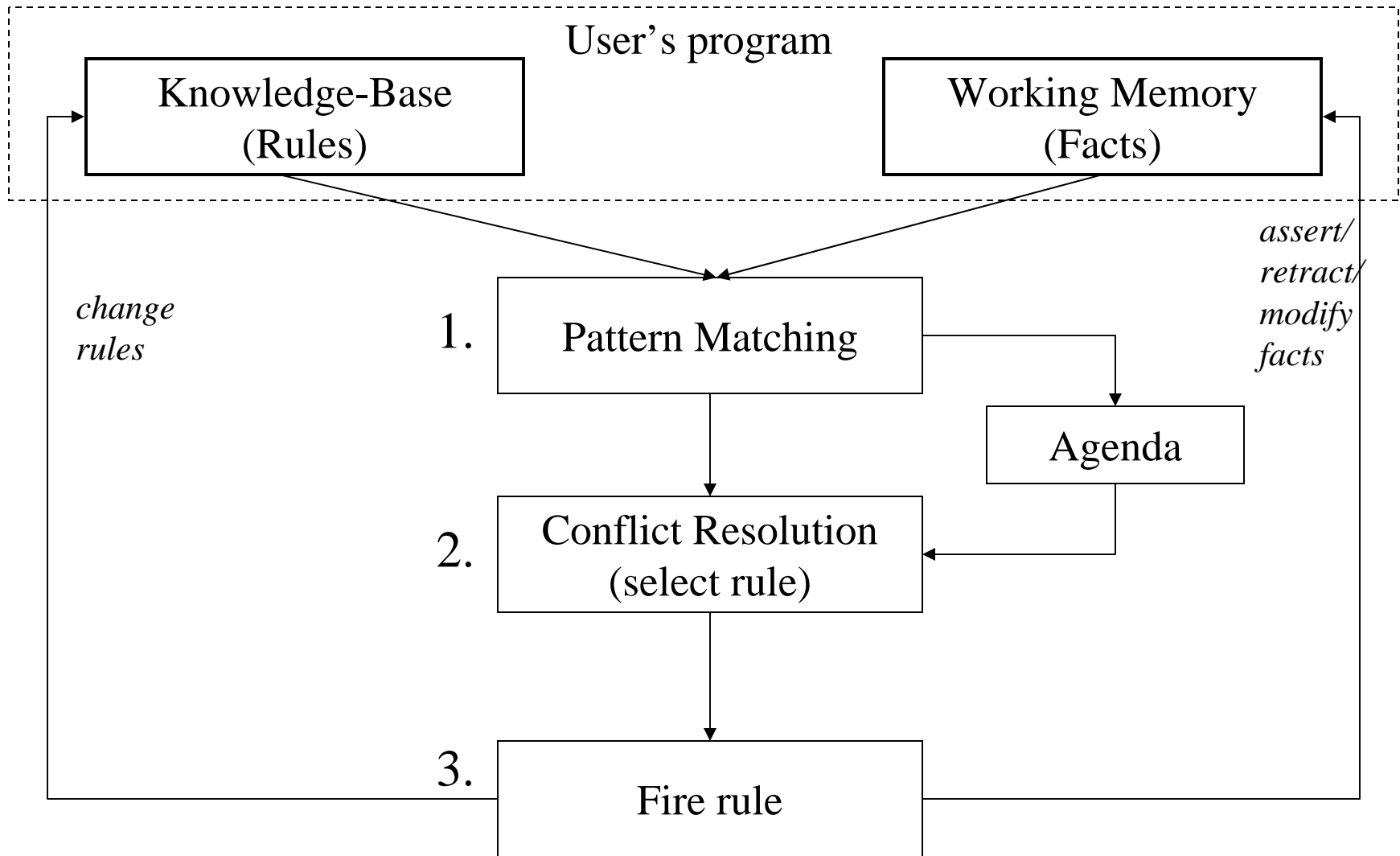
CLIPS basic elements



- **Fact-list:** global memory of data
- **Knowledge-base:** contain all the rules
- **Inference engine:** controls overall execution using forward chaining

- <http://www.ghg.net/clips/CLIPS.html>

Inference cycle



Antecedent Matching



1. matches facts in working memory against antecedents of rules
2. each combination of facts that satisfies a rule is called an instantiation
3. each matching rule is added to the agenda

Selection of a rule from the Agenda



Some selection strategies:

- Recency (most recent first)
triggered by the most recent facts
- Specificity (most specific first)
rules prioritized by the number of condition elements
- Random
choose a rule at random from the agenda

Execution of the rule



- Can modify working memory
 - add facts
 - remove facts
 - alter existing facts
- Alter rules
- Perform an external task (read sensors, control actuator)

Control mechanism



- Consider the following rule-base:
 - (1) Car won't start → check battery
 - (2) Car won't start → check gas
 - (3) Check battery AND battery bad → replace battery
- If the fact "*car won't start*" is asserted, then which of the rules (1) and (2) should be placed on the agenda? (1), (2), or both?
- We need a mechanism to place instantiations of rules on the agenda.

Control mechanisms



- **Markov algorithms:**

Approach: Apply rule with highest priority, if not applicable then take the next one etc.

Problem: inefficient for systems with many (1000s of) rules.
Has to do pattern matching on every rule in each cycle.

- **Rete algorithm:**

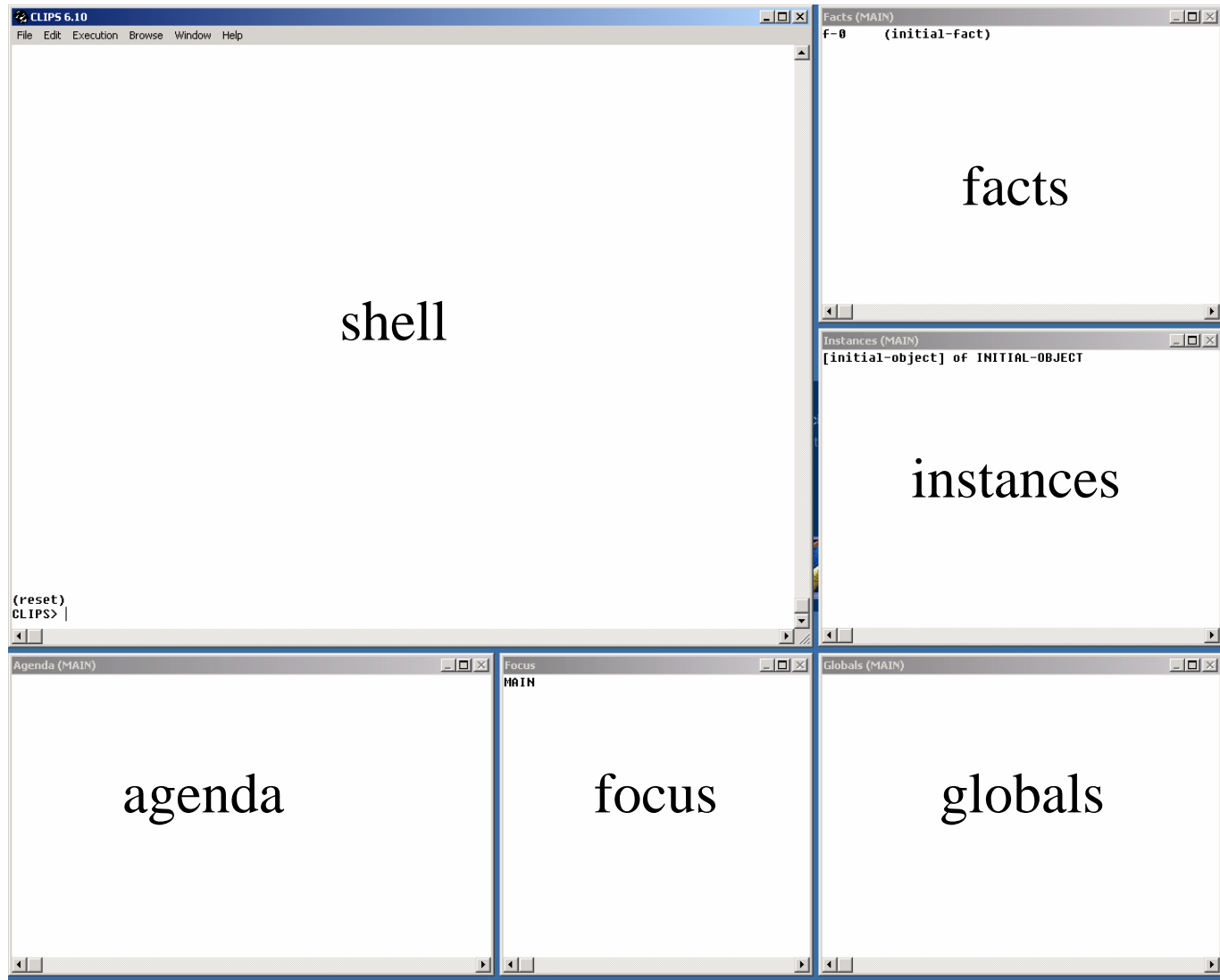
Fast pattern matching that obtains speed by storing information about all rules in a network. Only looks for changes in pattern matches in every cycle.

Install and run



- Access to CLIPS:
 - **On aludra:** at ~csci561a/clips
 - **In Windows:** install
<http://www.ghgcorp.com/clips/download/executables/pc>
- Running Clips
 - On aludra: > clips
 - In Windows: run clips.exe

Overview



Getting started



- Shell commands: (<command>)
 - (help)
 - (reset) → places (initial-fact) on factlist
 - (run) → runs till completion of program
 - (run 1) → runs 1 step
 - (facts) → shows the factlist
 - (assert (fact)) → puts (fact) on factlist
 - (retract 0) → removes fact with ID 0 from factlist
 - (defrule myrule ...) → defines a rule named *myrule*
 - (clear) → removes all facts from factlist

Facts



- (field1 field2 ... fieldN) an ordered, flat list
- E.g.,
(Hans 561a) is not equal to (561a Hans)
- (Hans (561a 561b)) is illegal
- Common to start with the relation that fact describes
e.g.,
(class Hans 561b)
- Keyword nil: used to indicate that a field has no value
- deftemplates to have names for each field

Field types



- **Types:**
 - Float: 1.34
 - Integer: 1, 2, 10, 20
 - Symbol: alkflksjfd
 - String: "duck/soup"
 - external-address:
 - fact-address:
 - instance-name:
 - instance-address:
- The type of each field is determined by the type of value stored in the field.
- In **deftemplates**, you can *explicitly* declare the type of value that a field can contain.

Deffacts



- (deffacts <deffacts name> [<optional comment>]
<<facts>>)

used to automatically assert **a set of facts**

- (deffacts status "some facts about emergency"
(emergency fire)
(fire-class A))
- Are asserted after a (reset) command

Adding and removing facts

- `(assert <<<fact>>>)` used to assert multiple facts
- `(retract <<<fact-index>>>)` removes a number of facts

e.g., `(assert (fact1)
(fact2))`

`(retract 1)`

- Is assigned a unique **fact identifier**: (e.g., f-1) starts with 'f' and followed by an integer called the **fact-index**
- **Fact-index**: can be used to refer to that fact (e.g., retract it)
- **Fact-list**: can be viewed in the fact-list window or using the `(facts)` command.

`(facts [<start> [<end> [<maximum>]])`

Components of a rule

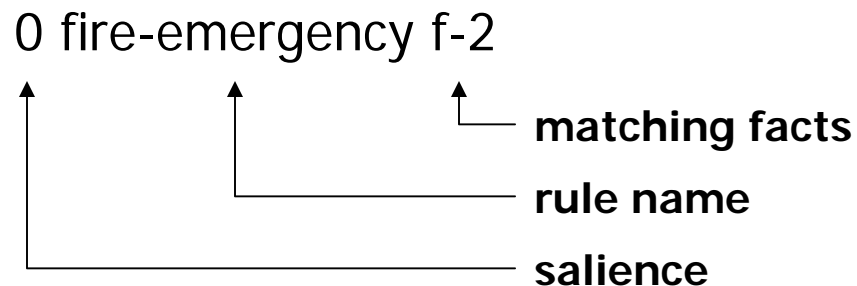
- (defrule <rule name> [<optional comment>]
 <<<patterns>>>
 =>
 <<<actions>>>)
- (defrule fire-emergency "An example rule"
 (emergency fire)
 =>
 (assert (action activate-sprinkler-system)))
- Rules can be inserted into the shell or loaded from a file using the (load) command

The agenda and activation

- (run [<limit>])

runs a CLIPS program,
<limit> is the number of rules to fire

- **Activating a rule:** requires that all its patterns on LHS (Left-Hand-Side) are matched. Asserting an existing fact has no effect.
- **List of activated rules:** can be seen in the agenda window or listed using (agenda)



Rule firing and refraction



- (run) will cause the most salient rule on the agenda to fire
- What if the run command is issued again?

Rule firing and refraction



- (run) will cause the most salient rule on the agenda to fire
- What if the run command is issued again?

There are no rules on the agenda so nothing will happen.

- **Refraction:** CLIPS rule firing models the refraction effect of a neuron to avoid endless loops

Commands used with rules



- **(rules)** displays the rules in the knowledge-base
- **(pprule <rule-name>)** displays a rule
- **(load <file-name>)** loads rules described in a file
- **(save <file-name>)** saves the stored rules into a file
- Comments: start with the character ";"

Multiple rules



- (defrule fire-emergency
 (emergency fire)
 =>
 (assert (action activate-sprinkler-system)))
- (defrule flood-emergency
 (emergency flood)
 =>
 (assert (action shut-down-electrical-equipment)))
- Asserting (emergency fire) will fire rule 1
 asserting (emergency flood) will activate rule 2

Rules with multiple patterns

- (defrule class-A-fire-emergency
 (emergency fire)
 (fire-class A)
 =>
 (assert (action activate-sprinkler-system)))
- (defrule class-B-fire-emergency
 (emergency fire)
 (fire-class B)
 =>
 (assert (action activate-carbon-dioxide-extinguisher)))
- All patterns must be matched for the rule to fire

Removing rules



- **(clear)** removes all rules from the knowledge-base
- **(excise <rule-name>)** removes rule

Debugging



- (watch {facts, rules, activations, all})
is used to provide the information about facts, rules, activations
- (unwatch {facts, rules, activations, all})
undoes the a (watch) command
- (matches <rule-name>)
indicates which patterns in a rule match facts
- (set-break <rule-name>)
allows execution to be halted before a rule
- (remove-break [<rule-name>])
removes all or a given breakpoint
- (show-breaks)
lists all breakpoints

Variables



- ?speed
- ?sensor
- ?value

```
(defrule grandfather
  (is-a-grandfather ?name) ?name bound to the 2nd field of fact
  =>
  (assert (is-a-man ?name)))
```

E.g: (is-a-grandfather John) → ?name = John
(is-a-grandfather Joe) → ?name = Joe

Wildcards



```
(person <name> <eye-color> <hair-color>)
```

```
(person John brown black)
```

```
(person Joe blue brown)
```

```
(defrule find-brown-haired-people
```

```
  (person ?name ?brown)
```

```
  =>
```

```
  (printout t ?name " has brown hair"))
```

States that eye color doesn't matter.

Control techniques



- Using **control facts**
- Using **salience**
- Using **control rules**

Example



```
CLIPS> (clear)
CLIPS> (assert (animal-is duck))
<Fact-0>
CLIPS> (assert (animal-sound quack))
<Fact-1>
CLIPS> (assert (The duck says "Quack."))
<Fact-2>
CLIPS> (facts)
f-0      (animal-is duck)
f-1      (animal-sound quack)
f-2      (The duck says "Quack.")
For a total of 3 facts.
CLIPS>
```

Example



```
CLIPS> (clear)
CLIPS> (assert (animal-is duck))
<Fact-0>
CLIPS> (defrule duck
  (animal-is duck)
=>
  (assert (sound-is quack)))
CLIPS>
```