# CLIPS Executive – Implementation

## Lab Course Winter Term 2021/2022

Till Hofmann, Tarik Viehmann

KB SG Knowledge-Based Systems Group | RWTH AACHEN UNIVERSITY

October 18, 2021

# Overview

# Production Systems

- Non-imperative
- First-Order Logic forward chaining
- Productions: condition-action rules
- A CLIPS program is mostly a set of rules
- If the condition holds, the actions are executed
- Working memory holds facts (‘’short-term memory”)
- Rules encode heuristic knowledge (‘’long-term memory”)

# CLIPS Terminology

**Facts** Information in Working Memory

```
(deftemplate machine
  (slot name (type SYMBOL))
  (slot team (type SYMBOL)
    (allowed-values nil CYAN MAGENTA))
  (slot mtype (type SYMBOL))
)
```

# CLIPS Terminology

**Facts** Information in Working Memory

```
(deftemplate machine
  (slot name (type SYMBOL))
  (slot team (type SYMBOL)
    (allowed-values nil CYAN MAGENTA))
  (slot mtype (type SYMBOL))
)
```

**Functions**

```
(deffunction get-output (?mps)
  "Return the navgraph point of the output side of the given mps"
  (return (str-cat ?mps "-O"))
)
```

# CLIPS Terminology

**Facts** Information in Working Memory

```
(deftemplate machine
  (slot name (type SYMBOL))
  (slot team (type SYMBOL)
    (allowed-values nil CYAN MAGENTA))
  (slot mtype (type SYMBOL))
)
```

**Functions**

```
(deffunction get-output (?mps)
  "Return the navgraph point of the output side of the given mps"
  (return (str-cat ?mps "-O"))
)
```

**Rules**

```
(defrule rule-name
  ?m <- (machine (name C-DS))
  =>
  (modify ?m (mtype DS))
)
```

# CLIPS Agenda

**Agenda:** Currently active rules

**Basic Execution Cycle:**

1. Top rule on agenda is selected
2. RHS of the rule is executed
3. Activate rules whose LHS is now satisfied, place on agenda according to salience and conflict resolution strategy
4. Deactivate rules whose LHS is no longer satisfied
5. Recompute dynamic saliences

**Salience:** highest salience is on top of the agenda

**Conflict Resolution:** depth first, breadth first, random, ...

# Overview

# Rule Components

## Rule Syntax

```
(defrule <rule-name> [<comment>]
  [<declaration>]          ; Rule Properties
  <conditional-element>* ; Left-Hand Side (LHS)
  =>
  <action>*                ; Right-Hand Side (RHS)
)
```

# Rule Components

## Rule Syntax

```
(defrule <rule-name> [<comment>]
  [<declaration>]          ; Rule Properties
  <conditional-element>* ; Left-Hand Side (LHS)
  =>
  <action>*                ; Right-Hand Side (RHS)
)
```

## Example

```
(defrule rule-name
  ?m <- (machine (name C-DS))
  =>
  (modify ?m (mtype DS))
)
```

# Binding Variables & Modifying Facts

**Bind variables in rules to**

- Match facts against each other
- Modify & retract fact
- Re-use their value

```
?m <- (machine (name C-RS2) (team CYAN))
```

# Asserting & Retracting Facts

- *Asserting* a fact adds it to the fact base

```
(defrule welcome
  (start)
  =>
  (assert (plan-action (id 1)
            (action-name say-hello))))
```

# Asserting & Retracting Facts

- *Asserting* a fact adds it to the fact base

```
(defrule welcome
  (start)
  =>
  (assert (plan-action (id 1)
            (action-name say-hello))))
```

- *Retracting* a fact removes it from the fact base

```
(defrule cancel-plan-actions
  (plan (id ?plan-id) (state CANCELED))
  ?a <- (plan-action (plan ?plan-id))
  =>
  (retract ?a))
```

# Asserting & Retracting Facts

- *Asserting* a fact adds it to the fact base

```
(defrule welcome
  (start)
  =>
  (assert (plan-action (id 1)
            (action-name say-hello))))
```

- *Retracting* a fact removes it from the fact base

```
(defrule cancel-plan-actions
  (plan (id ?plan-id) (state CANCELED))
  ?a <- (plan-action (plan ?plan-id))
  =>
  (retract ?a))
```

- *Modifying* a fact changes an existing fact

```
(defrule finish-plan
  ?p <- (plan (id ?plan-id))
  (not (plan-action (plan ?plan-id)
         (state ?state&~FINISHED)))
=>
  (modify ?p (state FINISHED)))
```

# Changing the World – Conditional changes

**The right-hand side may also contain simple conditions**

- If the additional conditional element is only a single value
- If additional facts have to be asserted in some cases
- Avoids clutter with a lot of similar rules

# Changing the World – Conditional changes

**The right-hand side may also contain simple conditions**

- If the additional conditional element is only a single value
- If additional facts have to be asserted in some cases
- Avoids clutter with a lot of similar rules

## Conditional Change

```
(if (not ?base) then
  (assert (holding NONE))
  (printout error ``Lost base during drive_to'' crlf)
)
```

# Example: Using CLIPS to Solve Towers of Hanoi

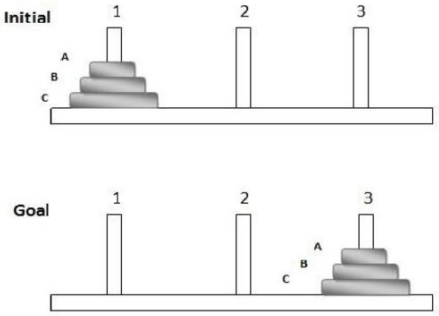


Figure: Towers of Hanoi[1]

[1] Shekoyan, Vazgen, Using multiple-possibility physics problems in introductory physics courses

# Overview

# Example: Hello World

- Simple domain with two actions:
  **say-hello** greets the user
  **say-goodbye** farewell, must happen after the greeting
- Single goal `TESTGOAL`
- Action sequence is hard-coded
- No monitoring

# Hello World in More Detail I

1. Goal creation (`goal-reasoner.clp`)

```
(defrule goal-reasoner-create
  (not (goal))
  (not (goal-already-tried))
  =>
  (assert (goal (id TESTGOAL)))
  (assert (goal-already-tried))
)
```

2. Goal selection (`goal-reasoner.clp`)

```
; We can choose one or more goals for expansion
(defrule goal-reasoner-select
  ?g <- (goal (id ?goal-id) (mode FORMULATED))
  =>
  (modify ?g (mode SELECTED))
  (assert (goal-meta (goal-id ?goal-id)))
)
```

# Hello World in More Detail II

3. Goal expansion (`fixed-sequence.clp`)

```
(defrule goal-expander-create-sequence
  ?g <- (goal (mode SELECTED) (id TESTGOAL))
  =>
  (assert
    (plan (id TESTGOAL-PLAN) (goal-id TESTGOAL)))
  (assert
    (plan-action (id 1) (plan-id TESTGOAL-PLAN)
      (duration 4.0) (action-name say-hello)))
  (assert
    (plan-action (id 2) (plan-id TESTGOAL-PLAN)
      (duration 4.0) (action-name say-goodbye)))
  (modify ?g (mode EXPANDED))
)
```

4. Goal commitment (`goal-reasoner.clp`)

```
(defrule goal-reasoner-commit
  ?g <- (goal (mode EXPANDED))
  =>
  (modify ?g (mode COMMITTED))
)
```

# Hello World in More Detail III

5. Goal dispatch (`goal-reasoner.clp`)

```
(defrule goal-reasoner-dispatch
  ?g <- (goal (mode COMMITTED))
  =>
  (modify ?g (mode DISPATCHED))
)
```

# Hello World in More Detail IV

6. Action selection (`action-selection.clp`)

```
(defrule action-selection-select
  ?pa <- (plan-action
           (plan-id ?plan-id) (id ?id)
           (status FORMULATED)
           (action-name ?action-name)
           (executable TRUE))
  (plan (id ?plan-id) (goal-id ?goal-id))
  (goal (id ?goal-id) (mode DISPATCHED))
  (not (plan-action
         (status PENDING|WAITING|RUNNING|FAILED)))
  (not (plan-action (status FORMULATED)
         (id ?oid&:(< ?oid ?id))))
  =>
  (modify ?pa (status PENDING))
)
```

7. Action execution (`plan-exec.clp`)

# Overview: Files

General CLIPS Executive
(`fawkes/src/plugins/clips-executive/clips/`):

```
action-selection/       # Select the next action to execute
  sequential.clp        # Sequential plans
  temporal.clp          # Temporal plans
coordination-mutex.clp  # Low-level mutex mechanism
domain.clp              # Domain representation
execmon/                # General execution monitoring
goal.clp                # Goal representation
goals/                  # Pre-defined goal types for goal trees
  retry.clp             # Retry a goal (tree)
  run-all.clp           # Run all sub-goals in a sequence
  run-one.clp           # Run one sub-goal
  try-all.clp           # Run all sub-goals until first one succeeds
```

# Overview: Files

General CLIPS Executive
(`fawkes/src/plugins/clips-executive/clips/`):

```
lock-actions.clp    # Locking mechanism on action level
pddl.clp            # Use a PDDL planner to create a plan
plan.clp            # Plan representation
resource-locks.clp  # Locking mechanism on goal level
skills.clp          # Skiller handling
skills-actions.clp  # Execute actions with the Skiller
wm-domain-sync.clp  # Synchronize domain model and world model
wm-robmem-sync.clp  # Synchronize world model with database
worldmodel.clp      # World model representation
```

# Overview: Files

RCLL Agent (`src/clips-specs/rcll`):

```
action-selection.clp # Domain-specific action selection
domain.clp # Domain initialization
domain.pddl # PDDL domain for the RCLL
execution-monitoring.clp # RCLL Execution monitoring
exploration.clp # Exploration game phase
fixed-sequence.clp # Pre-defined plans for goal expansion
goal-production.clp # Goals for the production phase
goal-reasoner.clp # Domain-specific goal reasoning
lock-actions.clp # Domain-specific lock actions
noop-actions.clp # Pseudo actions without actual execution
refbox-actions.clp # Interact with the refbox
refbox-worldmodel.clp # Update the world model with refbox info
```