Implementing Knowledge-based Systems

To build an interpreter for a language, we need to distinguish

- Base language the language of the RRS being implemented.
- Metalanguage the language used to implement the system.

They could even be the same language!

Implementing the base language

Let's use the definite clause language as the base language and the metalanguage.

- We need to represent the base-level constructs in the metalanguage.
- We represent base-level terms, atoms, and bodies as meta-level terms.
- We represent base-level clauses as meta-level facts.
- In the non-ground representation base-level variables are represented as meta-level variables.

Representing the base level constructs

- Base-level atom $p(t_1, \ldots, t_n)$ is represented as the meta-level term $p(t_1, \ldots, t_n)$.
- Meta-level term $oand(e_1, e_2)$ denotes the conjunction of base-level bodies e_1 and e_2.
- Meta-level constant *true* denotes the object-level empty body.
- The meta-level atom $clause(h, b)$ is true if "h if b" is a clause in the base-level knowledge base.

Example representation

The base-level clauses

$$connected\_to(l_1, w_0).$$

$$connected\_to(w_0, w_1) \leftarrow up(s_2).$$

$$lit(L) \leftarrow light(L) \wedge ok(L) \wedge live(L).$$

can be represented as the meta-level facts

$$clause(connected\_to(l_1, w_0), true).$$

$$clause(connected\_to(w_0, w_1), up(s_2)).$$

$$clause(lit(L), oand(light(L), oand(ok(L), live(L)))).$$

Making the representation pretty

- Use the infix function symbol "&" rather than *oand*.
  - instead of writing $oand(e_1, e_2)$, you write $e_1 \,\&\, e_2$.
- Instead of writing $clause(h, b)$ you can write $h \Leftarrow b$, where $\Leftarrow$ is an infix meta-level predicate symbol.
  - Thus the base-level clause "$h \leftarrow a_1 \wedge \cdots \wedge a_n$" is represented as the meta-level atom $h \Leftarrow a_1 \,\&\, \cdots \,\&\, a_n$.

Example representation

The base-level clauses

$$connected\_to(l_1, w_0).$$

$$connected\_to(w_0, w_1) \leftarrow up(s_2).$$

$$lit(L) \leftarrow light(L) \wedge ok(L) \wedge live(L).$$

can be represented as the meta-level facts

$$connected\_to(l_1, w_0) \Leftarrow true.$$

$$connected\_to(w_0, w_1) \Leftarrow up(s_2).$$

$$lit(L) \Leftarrow light(L) \,\&\, ok(L) \,\&\, live(L).$$

Vanilla Meta-interpreter

$prove(G)$ is true when base-level body G is a logical consequence of the base-level KB.

$$prove(true).$$

$$prove((A \,\&\, B)) \leftarrow$$
$$\quad prove(A) \wedge$$
$$\quad prove(B).$$

$$prove(H) \leftarrow$$
$$\quad (H \Leftarrow B) \wedge$$
$$\quad prove(B).$$

Example base-level KB

$$live(W) \Leftarrow$$
$$\quad connected\_to(W, W_1) \;\&$$
$$\quad live(W_1).$$
$$live(outside) \Leftarrow true.$$
$$connected\_to(w_6, w_5) \Leftarrow ok(cb_2).$$
$$connected\_to(w_5, outside) \Leftarrow true.$$
$$ok(cb_2) \Leftarrow true.$$
$$?prove(live(w_6)).$$

Expanding the base-level

Adding clauses increases what can be proved.

- Disjunction Let $a; b$ be the base-level representation for the disjunction of a and b. Body $a; b$ is true when a is true, or b is true, or both a and b are true.
- Built-in predicates You can add built-in predicates such as N is E that is true if expression E evaluates to number N.

Expanded meta-interpreter

$prove(true).$

$prove((A \,\&\, B)) \leftarrow$

$\quad prove(A) \wedge prove(B).$

$prove((A; B)) \leftarrow prove(A).$

$prove((A; B)) \leftarrow prove(B).$

$prove((N \text{ is } E)) \leftarrow$

$\quad N \text{ is } E.$

$prove(H) \leftarrow$

$\quad (H \Leftarrow B) \wedge prove(B).$

Depth-Bounded Search

- Adding conditions reduces what can be proved.

% *bprove*(G, D) is true if G can be proved with a proof tree of depth less than or equal to number D.

$$bprove(true, D).$$

$$bprove((A \,\&\, B), D) \leftarrow$$
$$\quad bprove(A, D) \wedge bprove(B, D).$$

$$bprove(H, D) \leftarrow$$
$$\quad D \geq 0 \wedge D_1 \text{ is } D - 1 \wedge$$
$$\quad (H \Leftarrow B) \wedge bprove(B, D_1).$$