Often you want to assume that your knowledge is complete.

**Example:** you can state what switches are up and the agent can assume that the other switches are down.

**Example:** assume that a database of what students are enrolled in a course is complete.

The definite clause language is **monotonic:** adding clauses can’t invalidate a previous conclusion.

Under the complete knowledge assumption, the system is **non-monotonic:** adding clauses can invalidate a previous conclusion.
Completion of a knowledge base

- Suppose the rules for atom $a$ are
  
  $a \leftarrow b_1$.

  \vdots

  $a \leftarrow b_n$.

  equivalently $a \leftarrow b_1 \lor \ldots \lor b_n$.

- Under the Complete Knowledge Assumption, if $a$ is true, one of the $b_i$ must be true:

  $a \rightarrow b_1 \lor \ldots \lor b_n$.

- Under the CKA, the clauses for $a$ mean **Clark’s completion:**

  $a \leftrightarrow b_1 \lor \ldots \lor b_n$
Clark’s completion of a knowledge base consists of the completion of every atom.

If you have an atom \( a \) with no clauses, the completion is \( a \leftrightarrow \text{false} \).

You can interpret negations in the body of clauses. \( \sim a \) means that \( a \) is false under the complete knowledge assumption. This is called negation as failure.
Bottom-up negation as failure interpreter

\[ C := \{\}; \]
repeat
\[ \text{either} \]
\[ \text{select } r \in KB \text{ such that} \]
\[ r \text{ is } "h \leftarrow b_1 \land \ldots \land b_m" \]
\[ b_i \in C \text{ for all } i, \text{ and} \]
\[ h \notin C; \]
\[ C := C \cup \{h\} \]
\[ \text{or} \]
\[ \text{select } h \text{ such that for every rule } "h \leftarrow b_1 \land \ldots \land b_m" \in KB \]
\[ \text{either for some } b_i, \sim b_i \in C \]
\[ \text{or some } b_i = \sim g \text{ and } g \in C \]
\[ C := C \cup \{\sim h\} \]
until no more selections are possible
Negation as failure example

\[ p \leftarrow q \land \sim r. \]
\[ p \leftarrow s. \]
\[ q \leftarrow \sim s. \]
\[ r \leftarrow \sim t. \]
\[ t. \]
\[ s \leftarrow w. \]
Top-Down negation as failure proof procedure

- If the proof for $a$ fails, you can conclude $\sim a$.
- Failure can be defined recursively:
  Suppose you have rules for atom $a$:
  
  $$ a \leftarrow b_1 $$
  $$ \vdots $$
  $$ a \leftarrow b_n $$
  
  If each body $b_i$ fails, $a$ fails.
  A body fails if one of the conjuncts in the body fails.
  Note that you need finite failure. Example $p \leftarrow p$. 