Recap (Lecture 1): Propositional logic program

A *definite clause* is a formula of one of the two shapes below

$$q \qquad \text{(a Prolog } \textit{fact}\ q\,.\ \text{)}$$

$$p_1 \wedge \cdots \wedge p_k \ \rightarrow q \qquad \text{(a Prolog } \textit{rule}\ q :\!-\ p_1, \ldots, p_k.\text{)}$$

where $p_1, \ldots, p_k, q$ are all *atoms* (that is, atomic statements).

A *logic program* is a list $F_1, \ldots, F_n$ of definite clauses

A *goal* is a list $g_1, \ldots, g_m$ of atoms.

The job of the system is to ascertain whether the logical consequence below holds.

$$F_1, \ldots, F_n \models g_1 \wedge \cdots \wedge g_m \ .$$

Example logic program (Lecture 1)

$$
\begin{array}{rcl}
\texttt{chicago} & \longrightarrow & \texttt{windy} \\
\texttt{edinburgh} & \longrightarrow & \texttt{windy} \\
\texttt{edinburgh} & \longrightarrow & \texttt{scotland} \\
\texttt{scotland} & \longrightarrow & \texttt{rainy} \\
\texttt{windy} \wedge \texttt{rainy} & \longrightarrow & \texttt{insideOutUmbrella} \\
 & & \texttt{edinburgh}
\end{array}
$$

The Prolog search procedure

The search starts with the goal $g_1, \ldots, g_m$, given by the user to the system, as the initial (current) goal.

At every stage in the search, we have an associated list of atoms: the *current goal*.

If we ever obtain the *empty goal* (notation ϵ) as the current goal then the search is successful, and Prolog returns the answer `yes`.

If the search terminates without obtaining the empty goal then Prolog returns the answer `no`.

We illustrate the procedure, using our running example.

The initial goal is thus `insideOutUmbrella`.

Current goal: `insideOutUmbrella`.

The only clause in the program that can be used to derive `insideOutUmbrella` is

$$\texttt{windy} \wedge \texttt{rainy} \rightarrow \texttt{insideOutUmbrella}$$

Replace the current goal with

$$\texttt{windy}, \texttt{rainy}$$

Current goal: $\texttt{windy}, \texttt{rainy}$

The clauses in the program that might be used to derive `windy` are:

$$\texttt{chicago} \rightarrow \texttt{windy}$$
$$\texttt{edinburgh} \rightarrow \texttt{windy}$$

Try the first rule first, so replace the current goal with

$$\texttt{chicago}, \texttt{rainy}$$

Current goal: `chicago, rainy`

There are no clauses in the program that can be used to derive `chicago`.

(This is the point at which Sicstus Prolog complains with the original Prolog formulation of the example. The error message states that no rule is found with `chicago` as conclusion.)

We thus *backtrack* to the last point at which there remains an unexplored search option.

We thus restore the current goal to

`windy, rainy`

Current goal: $\texttt{windy}, \texttt{rainy}$

We have already seen that the first matching clause

$$\texttt{chicago} \rightarrow \texttt{windy}$$

does not produce a derivation of $\texttt{windy}$.

We thus try the next clause in the program that matches $\texttt{windy}$:

$$\texttt{edinburgh} \rightarrow \texttt{windy}$$

So replace the current goal with

$$\texttt{edinburgh}, \texttt{rainy}$$

Current goal: `edinburgh, rainy`

The first atom in the goal, `edinburgh`, is an axiom.

We thus remove `edinburgh` from the goal list.

So replace the current goal with

`rainy`

Continuing the same way, we obtain successively:

Current goal: `rainy`

Current goal: `scotland`

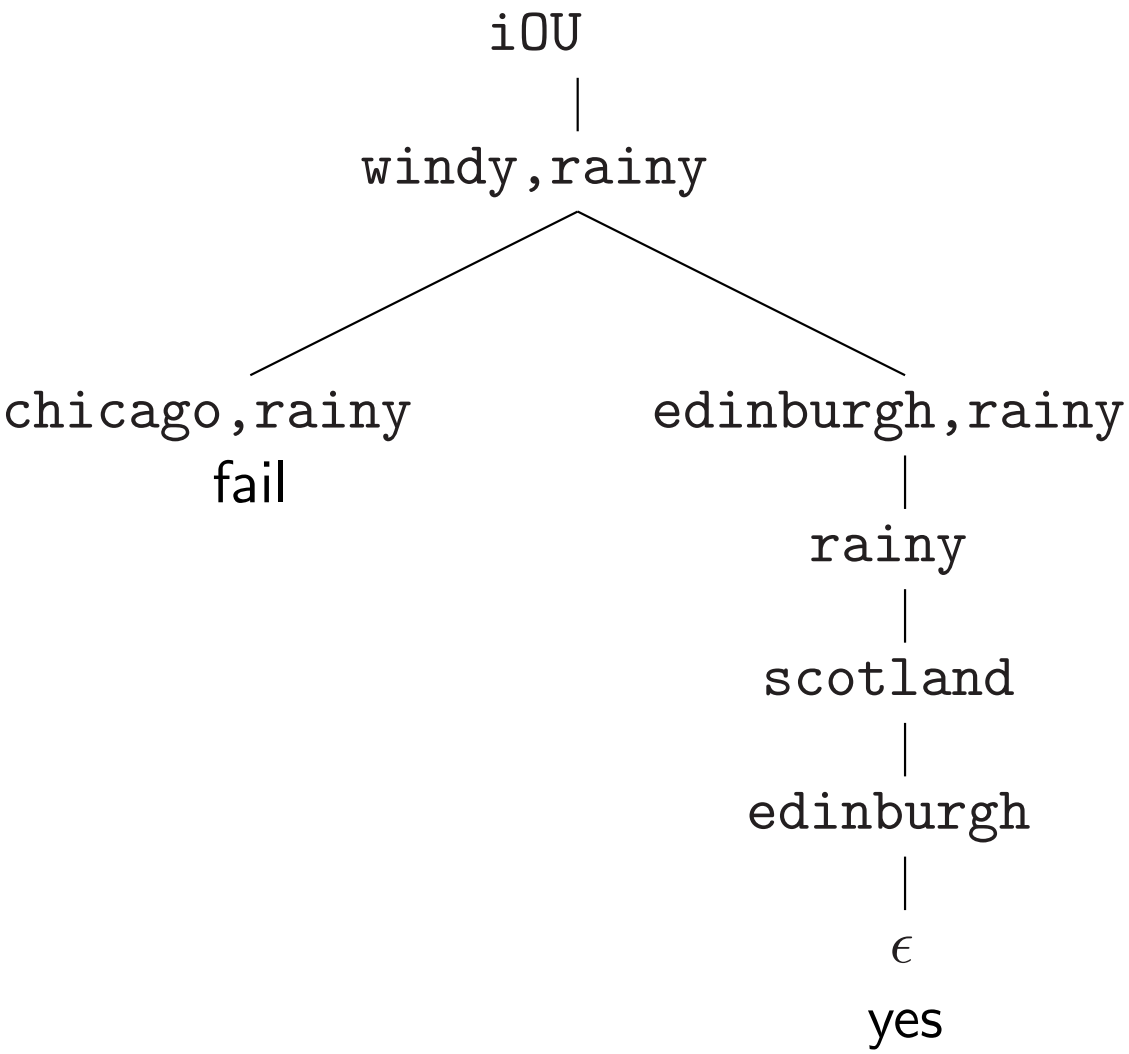
Current goal: `edinburgh`

Current goal: ϵ

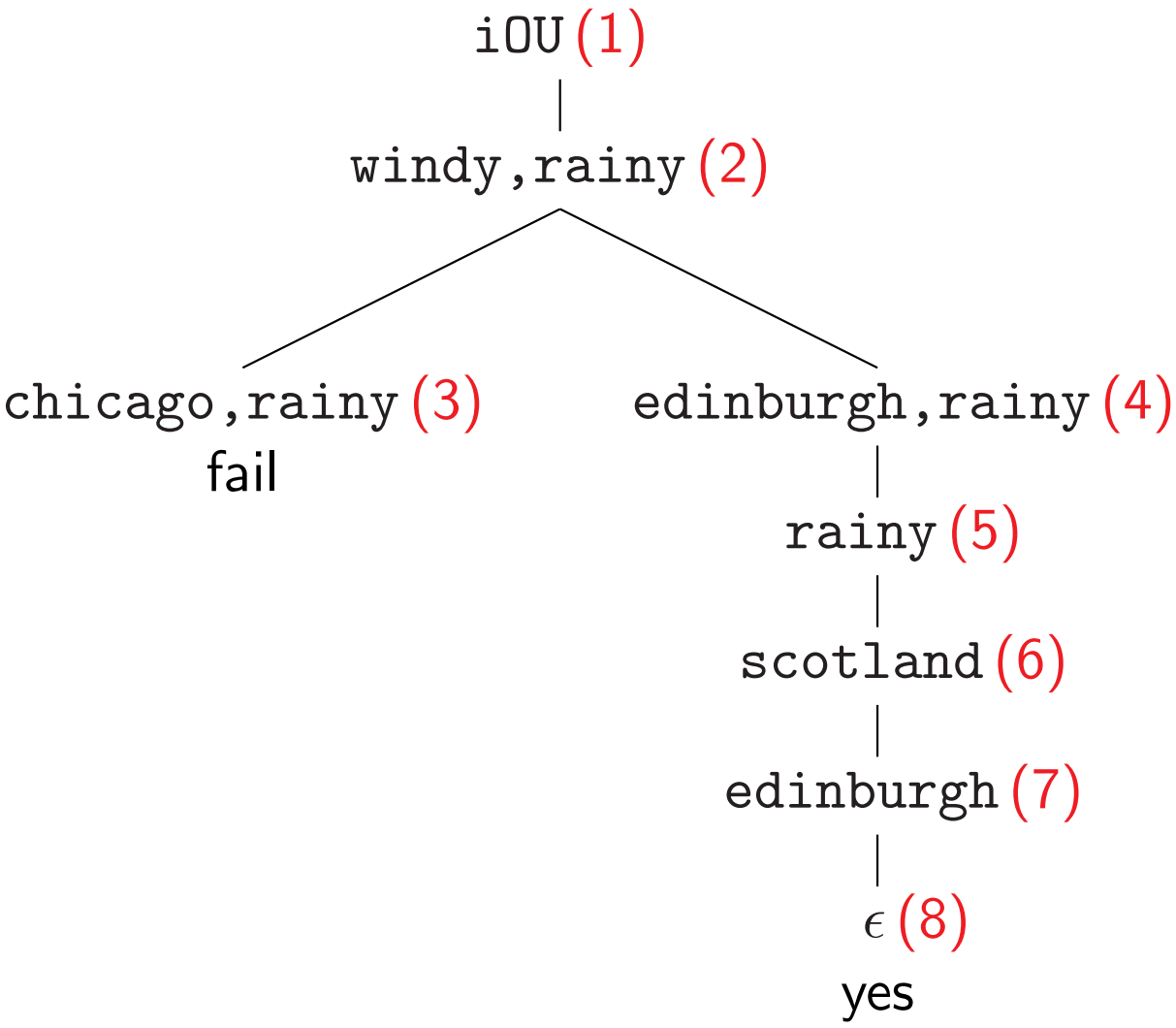
Now the search has produced an empty list for the current goal. This means the search for a derivation has been successful.

So (idealized) Prolog search thus returns the answer `yes` to the query `insideOutUmbrella`

Prolog search tree for goal `insideOutUmbrella`



Prolog search tree for goal `insideOutUmbrella`



Order in which the nodes are visited.

What does Prolog search do?

Why is Prolog search correct? Where does it come from?

The theoretical explanation is that (propositional) Prolog is searching for a *formal derivation* of the goal from *axioms* given by the program in an *inference system* (a.k.a. *proof system*) for definite clause (propositional) logic.

A derivation can be directly read off from a successful branch in the search tree.

The next slide shows the derivation that is found by proof search in the case of the running example.

After that we define the inference system rigorously

Example derivation

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\epsilon \quad \texttt{edin}}{\texttt{edin}} \qquad \texttt{edin} \rightarrow \texttt{scot}}{\texttt{scot}} \qquad \texttt{scot} \rightarrow \texttt{rain}}{\texttt{rain}} \qquad \texttt{edin}}{\texttt{edin}, \texttt{rain}} \qquad \texttt{edin} \rightarrow \texttt{wind}}{\texttt{wind}, \texttt{rain}} \qquad \texttt{wind} \wedge \texttt{rain} \rightarrow \texttt{iOU}}{\texttt{insideOutUmbrella}}
$$

The inference system

To derive new goals starting with the empty goal ϵ, which we view as trivially established.

We then use a single inference rule to derive a new *established goal* from a goal already established

$$\frac{g_1, \ldots, g_{l-1}, h_1, \ldots, h_k, g_{l+1}, \ldots, g_m \qquad h_1 \wedge \cdots \wedge h_k \rightarrow g_l}{g_1, \ldots, g_{l-1}, g_l, g_{l+1}, \ldots, g_m}$$

where $1 \leq l \leq m$ and $k \geq 0$.

The rule says: if we have already established goal $g_1, \ldots, g_{l-1}, h_1, \ldots, h_k, g_{l+1}, \ldots, g_m$ and we have $h_1 \wedge \cdots \wedge h_k \rightarrow g_l$ as one of the program clauses, then we can obtain $g_1, \ldots, g_{l-1}, g_l, g_{l+1}, \ldots, g_m$ as a new established goal.

Constructing a derivation

Derivations can be constructed in two directions.

- Bottom up: To prove a goal $g_1, \ldots, g_m$, start from the empty goal ϵ and apply the inference rules until $g_1, \ldots, g_m$ is reached.
- Top down: To prove $g_1, \ldots, g_m$, apply the inference rules backwards, starting from $g_1, \ldots, g_m$, until ϵ is produced.

Constructing a derivation

Derivations can be constructed in two directions.

- Bottom up: To prove a goal $g_1, \ldots, g_m$, start from the empty goal ϵ and apply the inference rules until $g_1, \ldots, g_m$ is reached.
- Top down: To prove $g_1, \ldots, g_m$, apply the inference rules backwards, starting from $g_1, \ldots, g_m$, until ϵ is produced.
- Counterintuitively, "bottom up" starts at the top of the paper and proceeds downwards, whereas "top down" starts at the bottom and builds upwards. The terminology "top down" and "bottom up" is motivated by the approach to problem solving rather than the direction the process takes on paper.

Constructing a derivation

Derivations can be constructed in two directions.

- Bottom up: To prove a goal $g_1, \ldots, g_m$, start from the empty goal ϵ and apply the inference rules until $g_1, \ldots, g_m$ is reached.
- Top down: To prove $g_1, \ldots, g_m$, apply the inference rules backwards, starting from $g_1, \ldots, g_m$, until ϵ is produced.
- Counterintuitively, "bottom up" starts at the top of the paper and proceeds downwards, whereas "top down" starts at the bottom and builds upwards. The terminology "top down" and "bottom up" is motivated by the approach to problem solving rather than the direction the process takes on paper.
- It makes no difference to the derivation how it is put together. However the top down approach is (usually) the one best suited to searching for a derivation.

Constructing a derivation

Derivations can be constructed in two directions.

- Bottom up: To prove a goal $g_1, \ldots, g_m$, start from the empty goal ϵ and apply the inference rules until $g_1, \ldots, g_m$ is reached.
- Top down: To prove $g_1, \ldots, g_m$, apply the inference rules backwards, starting from $g_1, \ldots, g_m$, until ϵ is produced.
- Counterintuitively, "bottom up" starts at the top of the paper and proceeds downwards, whereas "top down" starts at the bottom and builds upwards. The terminology "top down" and "bottom up" is motivated by the approach to problem solving rather than the direction the process takes on paper.
- It makes no difference to the derivation how it is put together. However the top down approach is (usually) the one best suited to searching for a derivation. Prolog proof search is top down.

Top-down proof search

A top-down search for a derivation maintains a list $g_1, \ldots, g_n$ of atoms, the *current goal*. The procedure is:

- For l with $1 \leq l \leq n$, find an axiom (program clause) of form

$$h_1 \wedge \cdots \wedge h_k \;\rightarrow\; g_l$$

  Replace g_l with $h_1, \ldots, h_k$. So the current goal becomes

$$g_1, \ldots, g_{l-1}, h_1, \ldots, h_k, g_{l+1}, \ldots, g_n$$

  This includes, with $k = 0$, the case in which g_l is itself an axiom, in which case it simply gets removed from the list.

The choice of l and choice of axiom means that the search space branches, producing a *search tree*.

The goal of the proof search procedure is to find a branch in the search tree ending in a leaf labelled with the empty goal ϵ.

Prolog search tree

Prolog search specializes the above by always considering the individual goal atoms in *goal order*.

This means that given a current goal

$$g_1, \ldots, g_n$$

Prolog always looks for an axiom (program clause) of form

$$h_1 \wedge \cdots \wedge h_k \;\rightarrow\; g_1$$

That is l on the previous slide is always chosen to be 1.

This policy is built into the Prolog search tree for a query. In a Prolog search tree, the branching reflects only the choice involved in selecting which axiom to use.

Prolog proof search

Prolog proof search is *depth first*:

- The search always moves to an *unvisited* (i.e., not previously visited) *child* (i.e., immediately below) node of the current node, whenever such a node exists.
- If there is no such node, the search backtracks to the most recently visited node from which such an unvisited child node is available.

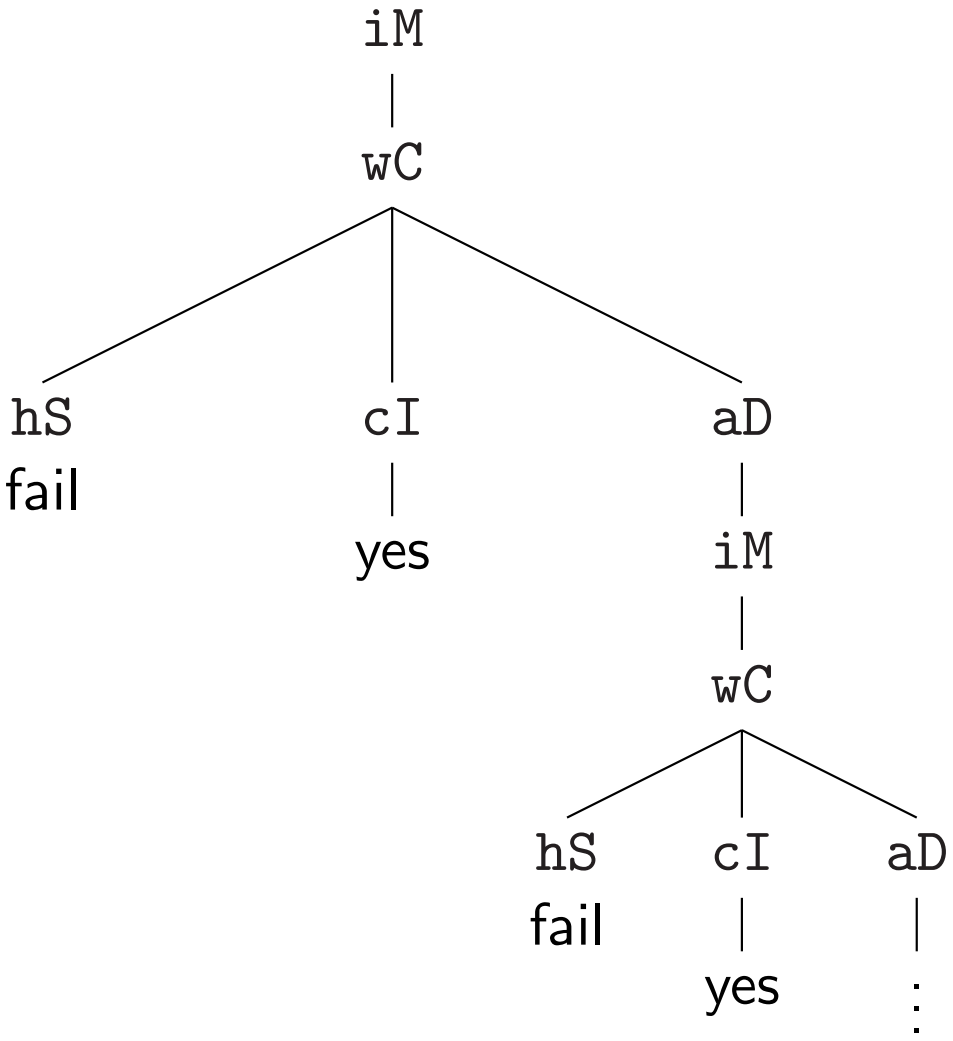
Prolog proof search follows *program order*.

- Child nodes are visited in the order that the axioms (Prolog rules) that determine the child node appear in the program (i.e., from left to right in the trees as we are drawing them).

Another example logic program

$$
\begin{aligned}
\texttt{hotterSun} &\rightarrow \texttt{warmerClimate} \\
\texttt{carbonIncrease} &\rightarrow \texttt{warmerClimate} \\
\texttt{warmerClimate} &\rightarrow \texttt{iceMelts} \\
\texttt{iceMelts} &\rightarrow \texttt{albedoDecrease} \\
\texttt{albedoDecrease} &\rightarrow \texttt{warmerClimate} \\
& \texttt{carbonIncrease}
\end{aligned}
$$

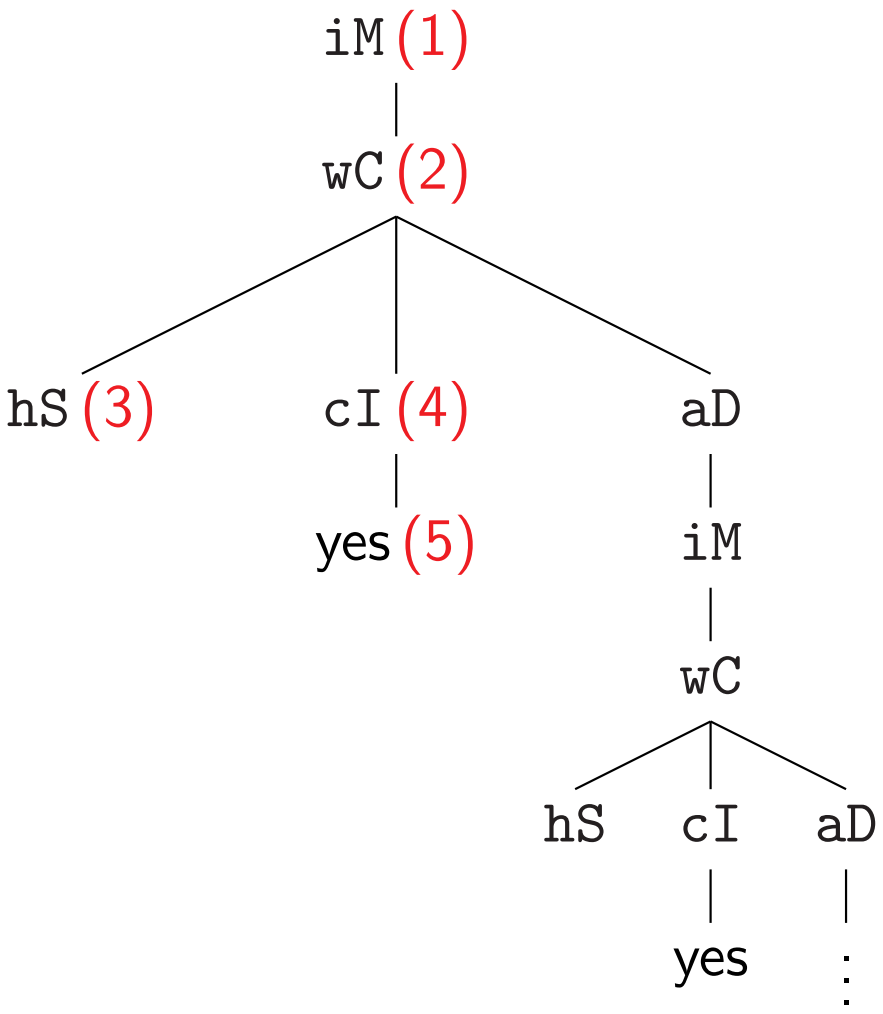
Prolog search tree for goal `iceMelts`



A derivation of the goal `iceMelts`

$$
\dfrac{\dfrac{\dfrac{\epsilon \quad \mathtt{cI}}{\mathtt{cI}} \qquad \mathtt{cI} \longrightarrow \mathtt{wC}}{\mathtt{wC}} \qquad \mathtt{wC} \longrightarrow \mathtt{iM}}{\mathtt{iM}}
$$

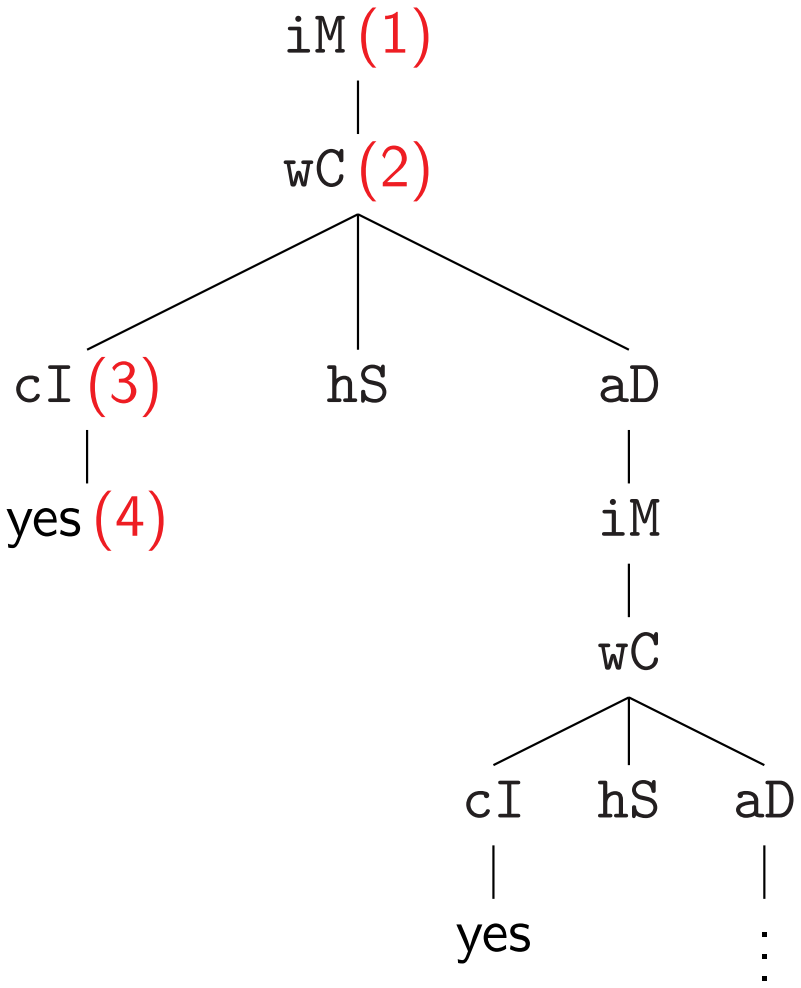
Prolog search for goal `iceMelts`



Reordering program can improve efficiency . . .

carbonIncrease	$\longrightarrow$	warmerClimate
hotterSun	$\longrightarrow$	warmerClimate
warmerClimate	$\longrightarrow$	iceMelts
iceMelts	$\longrightarrow$	albedoDecrease
albedoDecrease	$\longrightarrow$	warmerClimate
		carbonIncrease

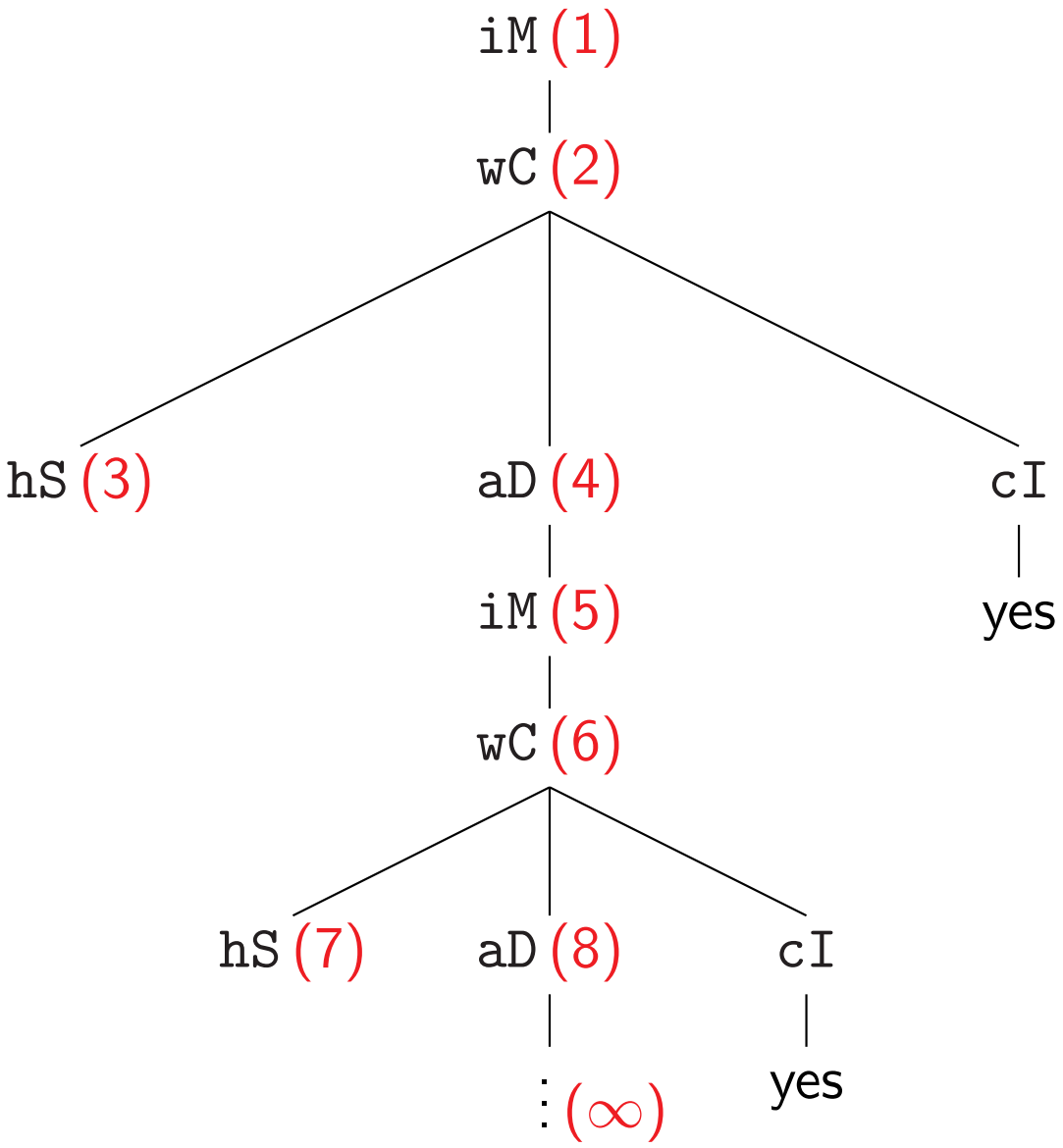
Resulting Prolog search



. . . or reduce efficiency . . .

`hotterSun`	$\longrightarrow$	`warmerClimate`
`albedoDecrease`	$\longrightarrow$	`warmerClimate`
`carbonIncrease`	$\longrightarrow$	`warmerClimate`
`warmerClimate`	$\longrightarrow$	`iceMelts`
`iceMelts`	$\longrightarrow$	`albedoDecrease`
		`carbonIncrease`

. . . drastically!



In Sicstus Prolog

Program:

```
warmerClimate :- hotterSun.
warmerClimate :- albedoDecrease.
warmerClimate :- carbonIncrease.
iceMelts :- warmerClimate.
albedoDecrease :- iceMelts.
carbonIncrease.
hotterSun :- false.
```

Query:

```
| ?- iceMelts.
!  Resource error:  insufficient memory
```

Incompleteness of Prolog proof search

The correct answer to the query `iceMelts` is `yes`, because `iceMelts` is a logical consequence of the program (irrespective of how we order the axioms in the program).

Our original derivation of `iceMelts` is a valid derivation (irrespective of how we order the axioms in the program).

However, whether Prolog proof search is successful or not depends on the order in which the axioms in the program are given.

The Prolog proof search procedure is said to be *incomplete*: it does not always find a derivation, even if a derivation exists.

Notation

- Goal $g_1, \ldots, g_m$ is a *logical consequence* of $F_1, F_2, \ldots, F_n$

$$F_1, \ldots, F_n \models g_1 \wedge \cdots \wedge g_m \ .$$

- Goal $g_1, \ldots, g_m$ is *derivable* from $F_1, F_2, \ldots, F_n$:

$$F_1, \ldots, F_n \vdash g_1, \ldots, g_m \ .$$

  i.e., there is some derivation of $g_1, \ldots, g_m$ from axioms $F_1, F_2, \ldots, F_n$

- Goal $g_1, \ldots, g_m$ is *Prolog derivable* from $F_1, F_2, \ldots, F_n$:

$$F_1, \ldots, F_n \vdash_{\text{Prolog}} g_1, \ldots, g_m \ ,$$

  i.e., Prolog proof search is successful.

Fundamental properties

Correctness of Prolog proof search

$$F_1, \ldots, F_n \vdash_{\text{Prolog}} g_1, \ldots, g_m \quad \text{implies} \quad F_1, \ldots, F_n \vdash g_1, \ldots, g_m \ .$$

Soundness of inference system

$$F_1, \ldots, F_n \vdash g_1, \ldots, g_m \quad \text{implies} \quad F_1, \ldots, F_n \models g_1 \wedge \cdots \wedge g_m \ .$$

Completeness of inference system

$$F_1, \ldots, F_n \models g_1 \wedge \cdots \wedge g_m \quad \text{implies} \quad F_1, \ldots, F_n \vdash g_1, \ldots, g_m \ .$$

Incompleteness of Prolog proof search

$$F_1, \ldots, F_n \models g_1 \wedge \cdots \wedge g_m \text{ doesn't imply } F_1, \ldots, F_n \vdash_{\text{Prolog}} g_1, \ldots, g_m \ .$$

Fundamental properties

Correctness of Prolog proof search (straightforward)

$$F_1, \ldots, F_n \vdash_{\text{Prolog}} g_1, \ldots, g_m \quad \text{implies} \quad F_1, \ldots, F_n \vdash g_1, \ldots, g_m \ .$$

Soundness of inference system

$$F_1, \ldots, F_n \vdash g_1, \ldots, g_m \quad \text{implies} \quad F_1, \ldots, F_n \models g_1 \wedge \cdots \wedge g_m \ .$$

Completeness of inference system

$$F_1, \ldots, F_n \models g_1 \wedge \cdots \wedge g_m \quad \text{implies} \quad F_1, \ldots, F_n \vdash g_1, \ldots, g_m \ .$$

Incompleteness of Prolog proof search

$$F_1, \ldots, F_n \models g_1 \wedge \cdots \wedge g_m \text{ doesn't imply } F_1, \ldots, F_n \vdash_{\text{Prolog}} g_1, \ldots, g_m \ .$$

Fundamental properties

Correctness of Prolog proof search (straightforward)

$$F_1, \ldots, F_n \vdash_{\text{Prolog}} g_1, \ldots, g_m \quad \text{implies} \quad F_1, \ldots, F_n \vdash g_1, \ldots, g_m \ .$$

Soundness of inference system (not too hard; see appendix)

$$F_1, \ldots, F_n \vdash g_1, \ldots, g_m \quad \text{implies} \quad F_1, \ldots, F_n \models g_1 \wedge \cdots \wedge g_m \ .$$

Completeness of inference system

$$F_1, \ldots, F_n \models g_1 \wedge \cdots \wedge g_m \quad \text{implies} \quad F_1, \ldots, F_n \vdash g_1, \ldots, g_m \ .$$

Incompleteness of Prolog proof search

$$F_1, \ldots, F_n \models g_1 \wedge \cdots \wedge g_m \text{ doesn't imply } F_1, \ldots, F_n \vdash_{\text{Prolog}} g_1, \ldots, g_m \ .$$

Fundamental properties

Correctness of Prolog proof search (straightforward)

$$F_1, \ldots, F_n \vdash_{\text{Prolog}} g_1, \ldots, g_m \quad \text{implies} \quad F_1, \ldots, F_n \vdash g_1, \ldots, g_m \ .$$

Soundness of inference system (not too hard; see appendix)

$$F_1, \ldots, F_n \vdash g_1, \ldots, g_m \quad \text{implies} \quad F_1, \ldots, F_n \models g_1 \wedge \cdots \wedge g_m \ .$$

Completeness of inference system (a bit harder; see appendix)

$$F_1, \ldots, F_n \models g_1 \wedge \cdots \wedge g_m \quad \text{implies} \quad F_1, \ldots, F_n \vdash g_1, \ldots, g_m \ .$$

Incompleteness of Prolog proof search

$$F_1, \ldots, F_n \models g_1 \wedge \cdots \wedge g_m \text{ doesn't imply } F_1, \ldots, F_n \vdash_{\text{Prolog}} g_1, \ldots, g_m \ .$$

Fundamental properties

Correctness of Prolog proof search (straightforward)

$$F_1, \ldots, F_n \vdash_{\text{Prolog}} g_1, \ldots, g_m \quad \text{implies} \quad F_1, \ldots, F_n \vdash g_1, \ldots, g_m \ .$$

Soundness of inference system (not too hard; see appendix)

$$F_1, \ldots, F_n \vdash g_1, \ldots, g_m \quad \text{implies} \quad F_1, \ldots, F_n \models g_1 \wedge \cdots \wedge g_m \ .$$

Completeness of inference system (a bit harder; see appendix)

$$F_1, \ldots, F_n \models g_1 \wedge \cdots \wedge g_m \quad \text{implies} \quad F_1, \ldots, F_n \vdash g_1, \ldots, g_m \ .$$

Incompleteness of Prolog proof search (already shown)

$$F_1, \ldots, F_n \models g_1 \wedge \cdots \wedge g_m \text{ doesn't imply } F_1, \ldots, F_n \vdash_{\text{Prolog}} g_1, \ldots, g_m \ .$$

Discussion

We have a complete inference system for propositional definite clause logic, but Prolog's proof search procedure is incomplete.

Incompleteness can be remedied in different ways, e.g.:

Discussion

We have a complete inference system for propositional definite clause logic, but Prolog's proof search procedure is incomplete.

Incompleteness can be remedied in different ways, e.g.:

- Add *loop checking* to the Prolog search procedure to check when the search reconsiders an atom already encountered.

  This provides a *decision procedure* for propositional definite-clause logic: the search will say `yes` if a derivation exists, and `no` if no derivation exists.

Discussion

We have a complete inference system for propositional definite clause logic, but Prolog's proof search procedure is incomplete.

Incompleteness can be remedied in different ways, e.g.:

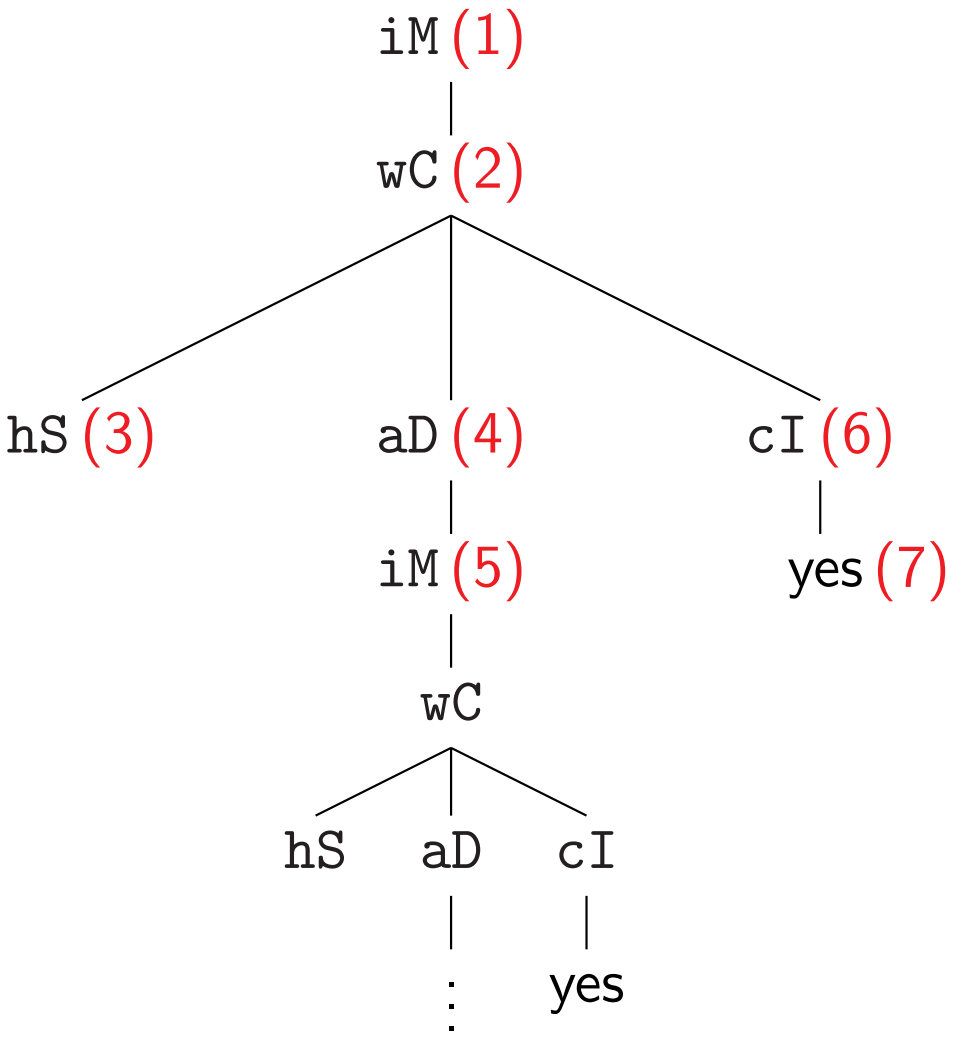
- Add *loop checking* to the Prolog search procedure to check when the search reconsiders an atom already encountered.

  This provides a *decision procedure* for propositional definite-clause logic: the search will say `yes` if a derivation exists, and `no` if no derivation exists.

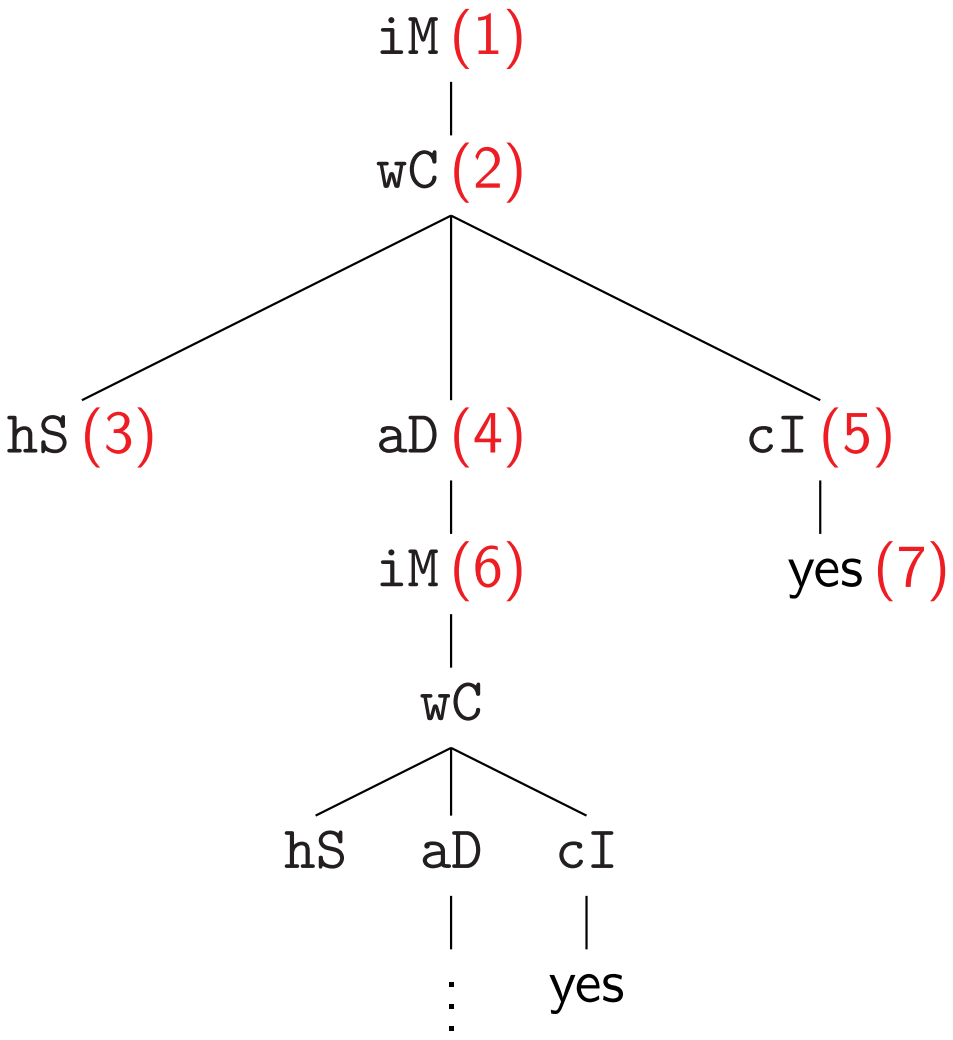
- Use *breadth-first* search instead of depth-first search.

  This gives a complete proof procedure (a derivation will be found whenever one exists) without loop checking, but not a decision procedure. Loop checking still needs to be added in order to detect when no derivation is available.

Loop checking



Breadth-first search



Discussion (continued)

There are sound pragmatic reasons, however, that Prolog does not implement such modified search algorithms.

- Loop checking makes proof search less efficient, and it is anyway not a useful modification once we move to the predicate (first-order) logic used by Prolog (where non-terminating behaviour can occur without the same goal ever repeating itself).
- Breadth-first search is in general inefficient, since it always explores all branches, irrespective of their usefulness.

  The benefit of program-order depth-first search is that the programer can order the clauses in the program to maximize the efficiency of Prolog's proof search algorithm.

Conclusions

Prolog proof search is an example of good engineering design based on an interplay between theoretical and practical considerations.

- It has a strong theoretical foundation, in being based on a complete inference system for definite-clause logic.
- However, its incomplete proof search procedure is a good implementation choice, allowing efficient proof search in the context of predicate logic, and permitting the programmer to tailor programs with regard to efficiency issues.

Main points today

Prolog search tree

inference system

Prolog search strategy as program-order depth-first search

incompleteness of Prolog proof search strategy

correctness of Prolog proof search strategy

soundness of inference system

completeness of inference system

Proofs of soundness and completeness are given as non-examinable appendices to this lecture.

Appendix: Proof of soundness

We need to prove:

$$F_1, \ldots, F_n \vdash g_1, \ldots, g_m \quad \text{implies} \quad F_1, \ldots, F_n \models g_1 \wedge \cdots \wedge g_m \ .$$

Suppose that $F_1, \ldots, F_n \vdash g_1, \ldots, g_m$.

Let $\mathcal{I}$ be an interpretation for which

$$\mathcal{I} \models F_1 \text{ and } \ldots \text{ and } \mathcal{I} \models F_n$$

We need to show that $\mathcal{I} \models g_1 \wedge \cdots \wedge g_m$.

To this end, let Π be a derivation tree for $g_1, \ldots, g_m$.

We show on next 2 slides that, for every established goal $g'_1, \ldots, g'_{m'}$. that occurs in Π, it holds that $\mathcal{I} \models g'_1 \wedge \cdots \wedge g'_{m'}$.

Thus, in particular, $\mathcal{I} \models g_1 \wedge \cdots \wedge g_m$, as required.

Proof of soundness (continued)

We show: for every established goal $g'_1, \ldots, g'_{m'}$. that occurs in Π, it holds that $\mathcal{I} \models g'_1 \wedge \cdots \wedge g'_{m'}$.

We work from the initial empty goal in Π downwards, propagating the property that $\mathcal{I} \models g'_1 \wedge \cdots \wedge g'_{m'}$ as we go.

(Formally, this is a proof by induction on the structure/height of the derivation.)

There are two cases:

1. If the goal $g'_1, \ldots, g'_{m'}$ is empty then $m' = 0$.

   It is immediate that $\mathcal{I} \models g'_1 \wedge \cdots \wedge g'_{m'}$ since an empty conjunction is **true** by convention.

Proof of soundness (continued)

2. Otherwise the derivation of $g'_1, \ldots, g'_{m'}$ within Π must look like

$$\dfrac{\begin{array}{c}\vdots\\ g'_1, \ldots, g'_{l-1}, h_1, \ldots, h_k, g'_{l+1}, \ldots, g'_{m'}\end{array} \qquad h_1 \wedge \cdots \wedge h_k \rightarrow g'_l}{g'_1, \ldots, g'_{m'}}$$

where $h_1 \wedge \cdots \wedge h_k \rightarrow g'_l$ is an axiom.

By the property we are propagating through the tree,

$$\mathcal{I} \models g'_1 \wedge \cdots \wedge g'_{l-1} \wedge h_1 \wedge \cdots \wedge h_k \wedge g'_{l+1} \wedge \cdots \wedge g'_{m'} \ .$$

Because $h_1 \wedge \cdots \wedge h_k \rightarrow g'_l$ is an axiom,

$$\mathcal{I} \models h_1 \wedge \cdots \wedge h_k \rightarrow g'_l \ .$$

Thus indeed $\mathcal{I} \models g'_1 \wedge \cdots \wedge g'_{m'}$. $\square$

Illustration of the propagation process

$$\dfrac{\dfrac{\dfrac{\epsilon}{\mathtt{cI}}}{\mathtt{cI}} \qquad \mathtt{cI} \longrightarrow \mathtt{wC}}{\mathtt{wC}} \qquad \mathtt{wC} \longrightarrow \mathtt{iM}}{\mathtt{iM}}$$

Illustration of the propagation process

$$\dfrac{\dfrac{\dfrac{(1)\,\mathcal{I} \models \epsilon \quad (1)\,\mathcal{I} \models \mathtt{cI}}{\mathtt{cI}} \qquad (1)\,\mathcal{I} \models \mathtt{cI} \rightarrow \mathtt{wC}}{\mathtt{wC}} \qquad (1)\,\mathcal{I} \models \mathtt{wC} \rightarrow \mathtt{iM}}{\mathtt{iM}}$$

Illustration of the propagation process

$$\dfrac{\dfrac{\dfrac{(1)\,\mathcal{I} \models \epsilon \quad (1)\,\mathcal{I} \models \mathtt{cI}}{(2)\,\mathcal{I} \models \mathtt{cI}} \qquad (1)\,\mathcal{I} \models \mathtt{cI} \rightarrow \mathtt{wC}}{\mathtt{wC}} \qquad (1)\,\mathcal{I} \models \mathtt{wC} \rightarrow \mathtt{iM}}{\mathtt{iM}}$$

Illustration of the propagation process

$$\dfrac{\dfrac{\dfrac{(1)\,\mathcal{I} \models \epsilon \quad (1)\,\mathcal{I} \models \mathtt{cI}}{(2)\,\mathcal{I} \models \mathtt{cI}} \qquad (1)\,\mathcal{I} \models \mathtt{cI} \rightarrow \mathtt{wC}}{(3)\,\mathcal{I} \models \mathtt{wC}} \qquad (1)\,\mathcal{I} \models \mathtt{wC} \rightarrow \mathtt{iM}}{\mathtt{iM}}$$

Illustration of the propagation process

$$\dfrac{\dfrac{\dfrac{(1)\,\mathcal{I} \models \epsilon \quad (1)\,\mathcal{I} \models \mathtt{cI}}{(2)\,\mathcal{I} \models \mathtt{cI}} \qquad (1)\,\mathcal{I} \models \mathtt{cI} \rightarrow \mathtt{wC}}{(3)\,\mathcal{I} \models \mathtt{wC}} \qquad (1)\,\mathcal{I} \models \mathtt{wC} \rightarrow \mathtt{iM}}{(4)\,\mathcal{I} \models \mathtt{iM}}$$

Appendix: Proof of completeness

We need to prove:

$$F_1, \ldots, F_n \models q_1 \wedge \cdots \wedge q_m \quad \text{implies} \quad F_1, \ldots, F_n \vdash q_1, \ldots, q_m \ .$$

Suppose that $F_1, \ldots, F_n \models q_1 \wedge \cdots \wedge q_m$.

We need to show that $F_1, \ldots, F_n \vdash q_1, \ldots, q_m$.

To this end, define the following interpretation $\mathcal{I}$:

$$\mathcal{I}(q) \quad = \quad \begin{cases} \textbf{true} & \text{if } F_1, \ldots, F_n \vdash q \\ \textbf{false} & \text{otherwise} \end{cases}$$

We show on next 2 slides that

$$\mathcal{I} \models F_1 \text{ and } \ldots \text{ and } \mathcal{I} \models F_n$$

Proof of completness (continued)

$$\mathcal{I}(q) \quad = \quad \begin{cases} \textbf{true} & \text{if } F_1, \ldots, F_n \vdash q \\ \textbf{false} & \text{otherwise} \end{cases}$$

We show that $\mathcal{I} \models F_i$ for every F_i in $F_1, \ldots, F_n$.

There are 2 cases to consider.

1. If F_i is an atom q then trivially $F_1, \ldots, F_n \vdash q$. So $\mathcal{I}(q) = \textbf{true}$. I.e., $\mathcal{I} \models q$.

Proof of completness (continued)

$$\mathcal{I}(q) \quad = \quad \begin{cases} \textbf{true} & \text{if } F_1, \ldots, F_n \vdash q \\ \textbf{false} & \text{otherwise} \end{cases}$$

2. In the case F_i is an implication $p_1 \wedge \cdots \wedge p_k \rightarrow q$, we need to show $\mathcal{I} \models p_1 \wedge \cdots \wedge p_k \rightarrow q$.

   Suppose $\mathcal{I} \models p_1$ and ... and $\mathcal{I} \models p_k$. That is, $F_1, \ldots, F_n \vdash p_1$ and ... and $F_1, \ldots, F_n \vdash p_k$.

   Combine the derivations of $p_1, \ldots, p_k$ to get one of q:

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ p_1\end{array} \quad \ldots \quad \begin{array}{c}\vdots\\ p_k\end{array}}{p_1, \ldots, p_k} \qquad p_1 \wedge \cdots \wedge p_k \rightarrow q}{q}$$

   So $F_1, \ldots, F_n \vdash q$. Thus $\mathcal{I} \models q$.

Proof of completeness (completed)

$$\mathcal{I}(q) \quad = \quad \begin{cases} \textbf{true} & \text{if } F_1, \ldots, F_n \vdash q \\ \textbf{false} & \text{otherwise} \end{cases}$$

We have assumed that $F_1, \ldots, F_n \models q_1 \wedge \cdots \wedge q_m$,
and we have shown that $\mathcal{I} \models F_1$ and ... and $\mathcal{I} \models F_n$.

Therefore $\mathcal{I} \models q_1 \wedge \cdots \wedge q_m$. Whence, by definition of $\mathcal{I}$:

$$F_1, \ldots, F_n \vdash q_1 \text{ and } \ldots \text{ and } F_1, \ldots, F_n \vdash q_m$$

We now *combine the derivations*:

$$\frac{\begin{array}{ccc} \vdots & & \vdots \\ q_1 & \ldots & q_m \end{array}}{q_1, \ldots, q_m}$$

Showing that $F_1, \ldots, F_n \vdash q_1, \ldots, q_m$ as required. $\square$

Combining derivations

In two places in the proof we needed to "combine" m derivations of several individual goals q_1 and ... and q_m to obtain a combined derivation the single goal $q_1, \ldots, q_m$. In the proof we drew this combined derivation in the schematic form:

$$\frac{\begin{array}{ccc} \vdots & & \vdots \\ q_1 & \ldots & q_m \end{array}}{q_1, \ldots, q_m}$$

It is left as an exercise to explicitly define the method of combining derivations being used here.