Proof Support for Hybrid Systems Verification, II

Ordered Resolution

Lawrence C Paulson, Computer Laboratory, University of Cambridge

A brief history of resolution

- Developed by J Alan Robinson in 1963
  - Based on earlier work by Davis and Putnam
  - Regarded by some (e.g. John McCarthy) as the key to Artificial Intelligence
- Overtaken by a revival of Davis-Putnam techniques during the 90s (today's SAT and SMT solvers)

The basic resolution rule

$$\frac{A \vee B \qquad \neg A' \vee C}{(B \vee C)\sigma} \qquad (A\sigma = A'\sigma)$$

Complementary atomic formulas are *unified.*

The intermediate formula disappears; the substitution, σ, affects the conclusion.

A resolution step

$$\boxed{R(f(X), X)} \vee P(X)$$
$$\boxed{\neg R(Y, 1)} \vee Q(Y)$$

identify complementary literals

$$\boxed{R(f(1), 1)} \vee P(1)$$
$$\boxed{\neg R(f(1), 1)} \vee Q(f(1))$$

unify and substitute

$$X \mapsto 1$$
$$Y \mapsto f(X) = f(1)$$

$$P(1) \vee Q(f(1))$$

The saturation algorithm (“Otter loop”)

At first, all clauses belong to the *passive* set. Then loop:

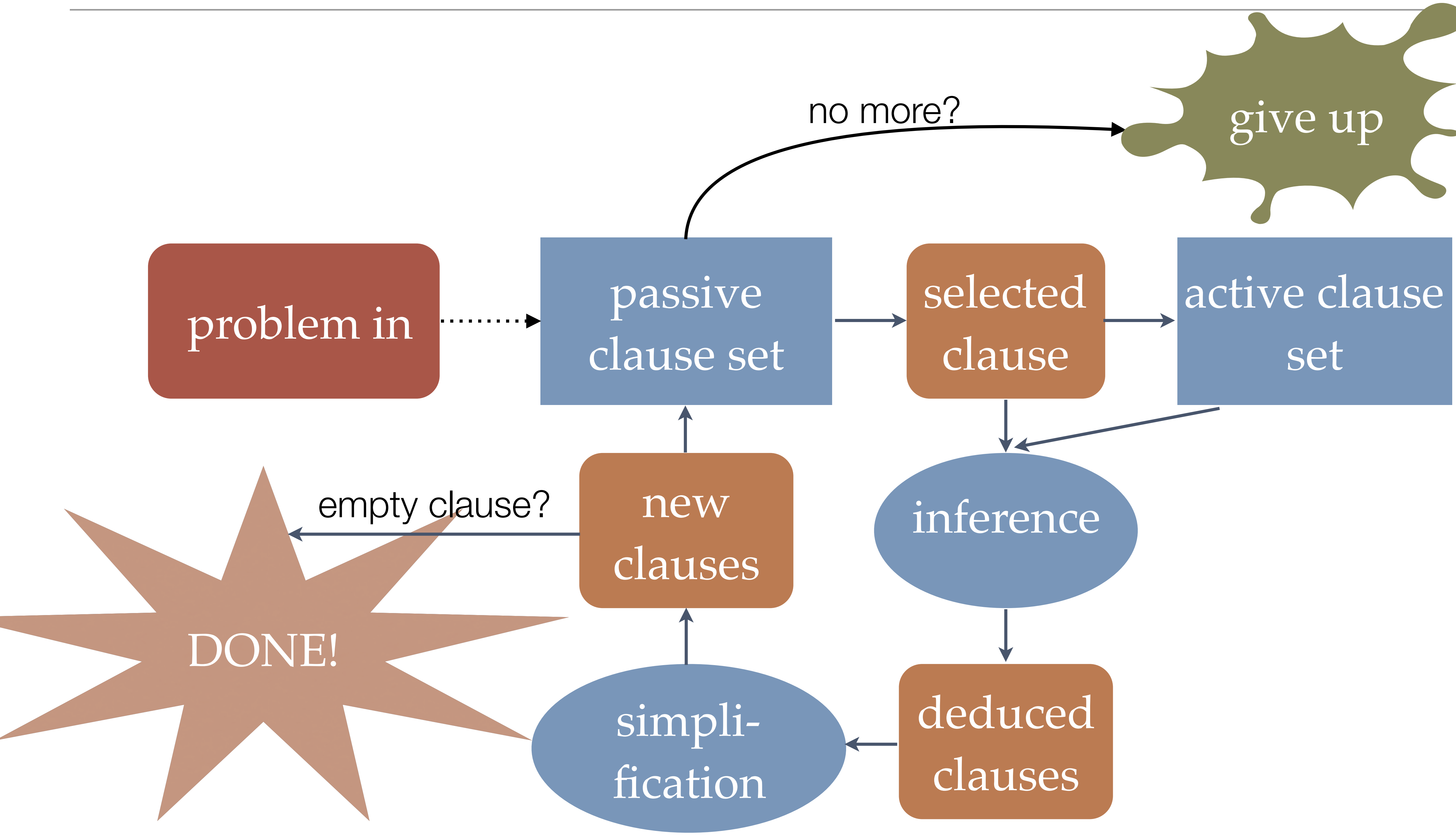
1. **Select** a clause for resolution. Move it to the *active* set.

2. **Perform all inferences** between the current clause and any active clause.

3. **Simplify** the new clauses. Add them to passive.

Repeat until contradiction found or *passive* = {}.

Resolution data flow



Completeness

- Most resolution calculi are *complete*: if a proof exists, it will be found in finite time.
- Immense theoretical importance (as a sanity check)
- Few practical implications: the runtime is **unlimited**.

Sacrificing completeness is an implementation decision!

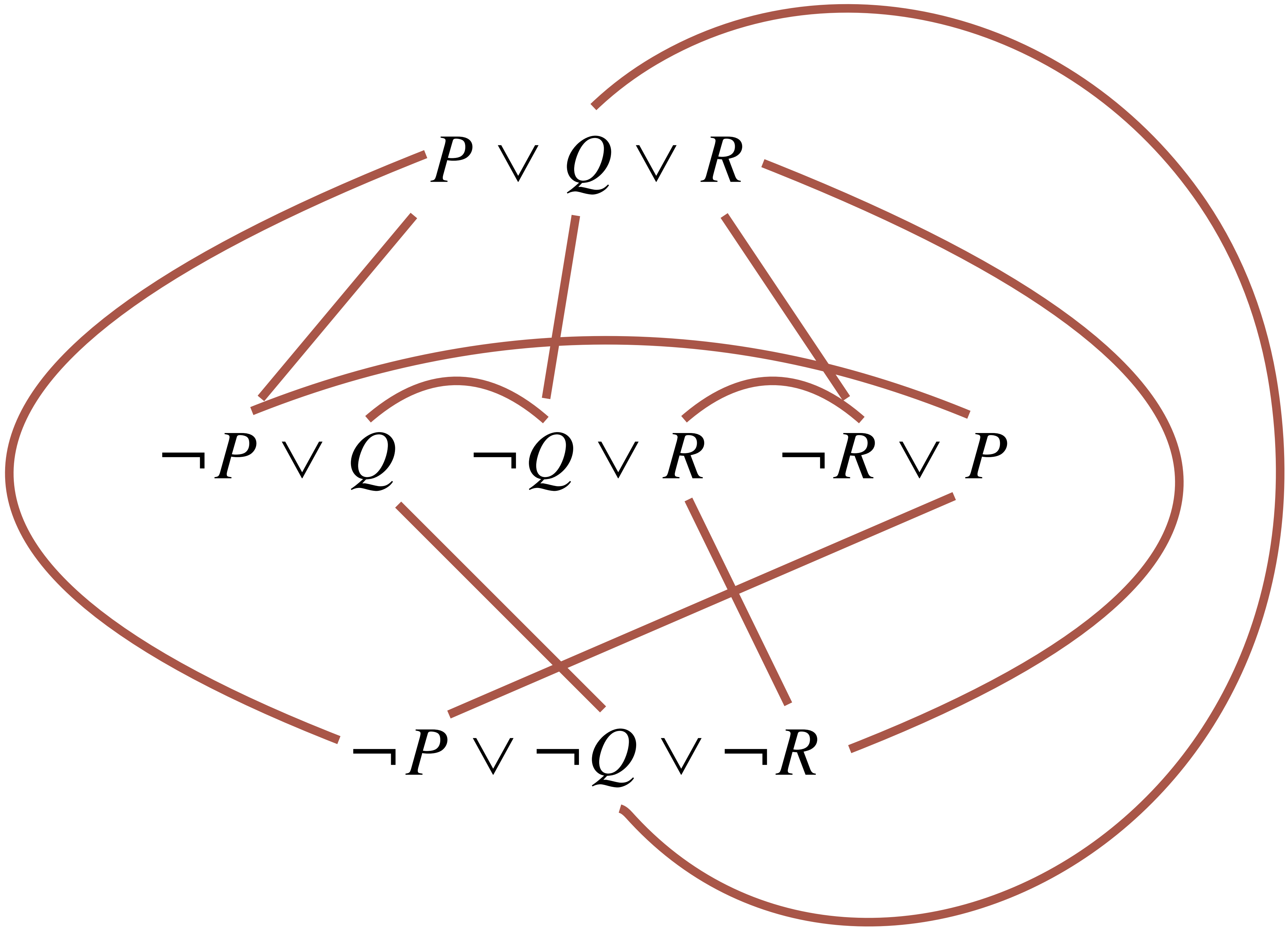
Selecting a clause

- A common heuristic is to choose the lightest *weight*
  - the *sum of the weights* of the constants in the clause
  - penalising clauses that are **too long**
  - … or that contain “difficult” constants such as $\mathbf{exp}$
- Must be **fair**: every clause is selected eventually (or else, reject completeness, even *discarding* very heavy clauses)

The problem of redundancy

- If there are n clauses, there are $O(n^2)$ pairs to check
- Within a clause, every literal will be complementary to literals in other clauses
- The number of possible combinations is exponential, but a proof requires only one!

An example with threefold symmetry



Ordering constraints on literals in a clause

- Within each clause, prioritise certain *literals*
  - typically the “most difficult” literals
  - no other literals in a clause can be resolved
- mainly based on **syntactic term orderings**

Reduction orderings on terms

- Relation $>$ is *well-founded* if no infinite chains $t_0 > t_1 > \ldots$
- It is *closed under substitution* if $t > u$ implies $C[t] > C[u]$
- *Reduction orderings* satisfy both properties.
  - Recursive path ordering
  - **Knuth-Bendix ordering (KBO)**
- Used to *constrain resolution* and to *orient equations*.

Knuth-Bendix ordering

- Choose an order $<$ on the function/predicate symbols, the *precedence*
- Choose *weights* for the symbols (these are separate from the weights used for clause selection)
- The weight of a term is the obvious recursive summation
- Define $\mathrm{occ}(x,t)$ = # of occurrences of x in t

Knuth-Bendix ordering: the details

$t >_{kbo} u$ if $\mathrm{occ}(x,t) \geq \mathrm{occ}(x,u)$ for all variables x in t, u **AND**

1. $\mathrm{weight}(t) > \mathrm{weight}(u)$ **OR**
2. $\mathrm{weight}(t) = \mathrm{weight}(u)$ **AND**
   a. $f > g$, comparing the head function symbols by precedence **OR**
   b. $f = g$, the head function symbols match and KBO holds on the arguments lexicographically

What was that again?

compare weights!

if equal, compare head functions by precedence

if identical, compare the arguments recursively

But most important: compare the variable occurrences!

This will cause problems later…

The **ordered** resolution rule

$$\frac{A \vee B \qquad \neg A' \vee C}{(B \vee C)\sigma} \quad (A\sigma = A'\sigma)$$

where the complementary literals are both **maximal** in their clauses (after unification)

Can impose further constraints, such as *negative selection*

Our example with clause weights and ordering restrictions

$P \lor Q \lor R$
6

$\neg P \lor Q$ $\neg Q \lor R$ $\neg R \lor P$
3   5   4

lightest clause

$\neg P \lor \neg Q \lor \neg R$
6

And the **one** possible resolution proof

$\neg P \vee Q$

$\neg R \vee P$

$\neg Q \vee R$

$P \vee \neg Q$

$P \vee Q \vee R$

$P \vee Q$

P

$\neg P \vee \neg Q \vee \neg R$

$\neg P \vee \neg Q$

$\neg P$

$\square$

Precedences are $P < Q < R$

Weights are P=1, Q=2, R=3

(For both KBO and clause selection)

Benefits of the heuristics and constraints

- *much smaller* search space
- **focused** on the *simplest* (most promising) clauses
- within them, only the *most difficult* literals can be resolved
- for MetiTarski:
  - focuses on clauses containing the **fewest** functions
  - works to eliminate the **remaining** functions

The ordered *paramodulation* rule

$$\frac{t = u \vee B \quad A[t'] \vee C}{(A[u] \vee B \vee C)\sigma} \quad (t\sigma = t'\sigma)$$

a generalisation of rewriting: the equality is applied after *unification*

ordering constraint: the replacement term is no larger, $u\sigma \not\succ t\sigma$

a **complete** treatment of equality

Summary

Resolution calculi enjoy strong theoretical properties, such as *completeness*

A well-established algorithm saturates the set of clauses in search for contradiction

Heuristics exist to focus on the most promising clauses

Ordering constraints focus on the "most difficult" literals in a clause.