

Tutorial: Deduction of new rules from old ones.

Johan G. F. Belinfante
2001 November 2

```
<< goedel52.L08; << tests.m

:Package Title: GOEDEL52.L08      2001 October 30 at 4:50 p.m.

It is now:  2001 Nov 2 at 11:37

Loading Simplification Rules

TESTS.M                      Revised 2001 October 18

weightlimit = 30

Context switch to `Goedel`Private is needed for ReplaceTest

Just ignore the error message about Unterminated use of BeginPackage

Get::bebal : Unterminated uses of BeginPackage or Begin in << tests.m.
```

■ Using SubstTest for deductions.

Although the **GOEDEL** program has no mechanism for automated deduction, the **SubstTest** tool can be used to carry out hand-guided deductions of new rules from old ones. This notebook shows how some basic deductions can be done. The basic idea of this tool is to compare two different orders of evaluation for some expression. The **SubstTest** tool can be used to obtain rules for simplifying classes, as well statements, but in this tutorial only the case of statements will be treated. Moreover, no specific examples are considered here, only generic ones.

```
? SubstTest

Goedel`Private`SubstTest

SubstTest[f_, x_, r_] := f @@ ({x} /. r) == (f @@ {x} /. r)
```

The output of the test will be a valid equality of the form **a == b**, but not all such equalities can be made into new rules. In particular, one may need to reverse the equation or to think about whether a rule based on a given equality would lead to looping. The **GOEDEL** program itself currently has no tools to help make such decisions.

■ Rewriting an implication.

Implications are automatically rewritten in a clause format:

```
implies[p, q]

or[q, not [p]]
```

Suppose one has a rule of the following form:

```
or[q, not[p]] := True
```

Using **SubstTest**, we can derive related rules:

```
SubstTest[and, implies[x, y], x, {x -> p, y -> q}] // Reverse
and[p, q] == p

SubstTest[and, or[x, y], implies[x, y], {x -> p, y -> q}]
or[p, q] == q
```

In each case it would be possible to add new rules by replacing `==` with `:=`. Note that in one case, we used **Reverse** to turn the equality around.

■ Modus Ponens

Suppose one knows both of these facts:

```
p := True;
or[q, not[p]] := True
```

Without a new rule, *Mathematica* would not know that **q** can be reduced to **True**. One can use **SubstTest** to do a modus ponens type deduction of a new rule:

```
SubstTest[and, or[x, y], implies[x, y], {x -> p, y -> q}] // Reverse
q == True
```

■ Cut rule

Suppose one has rules that assert that **p1** implies **p2**, and that **p2** implies **p3**, and one wants to deduce the rule that **p1** implies **p3**.

```
or[p2, not[p1]] := True; or[p3, not[p2]] := True
```

In this case we must supplement **SubstTest** with an overall negation to get the desired equation:

```
Map[not, SubstTest[and, implies[x, y],
  implies[y, z], not[implies[x, z]], {x -> p1, y -> p2, z -> p3}]]
or[p3, not[p1]] == True
```

Mapping an equation with **not** produces a logically equivalent equation.