Proof of the Distributive Law for Prioritisation and Pareto Composition

Bernhard Möller, Patrick Roocks

Institut für Informatik, Universität Augsburg, D-86135 Augsburg, Germany
moeller@informatik.uni-augsburg.de

Working Paper
April 10, 2012

In this document we show a sample theorem from [2] with the automated theorem prover Prover9 [1].

1 The Proof

The distributive law for Prioritisation and Pareto composition is stated in the following theorem:

Theorem 1. *For $a :: T_a$ we have that $(a\,\&)$ distributes over $\lessdot\!\!\otimes, \otimes\!\!\gtrdot$ and $\otimes$.*

1.1 Strategy

We show this using Prover 9. First, we show the auxiliary equation

$$a \,\&\, b + 1_{a \bowtie b} = a \,\&\, (b + 1_b) \ .$$

Afterwards we show the claim for $\otimes\!\!\gtrdot$, i.e.

$$(a \,\&\, b) \otimes\!\!\gtrdot (a \,\&\, c) = a \,\&\, (b \otimes\!\!\gtrdot c) \ .$$

We show this in the following three steps:

$$\begin{array}{ll} & (a \,\&\, b) \otimes\!\!\gtrdot (a \,\&\, c) \\ = & a \bowtie \top_b \bowtie a \bowtie \top_c \qquad\quad + a \bowtie \top_b \bowtie 1_a \bowtie c \,+ \\ & 1_a \bowtie (b + 1_b) \bowtie a \bowtie \top_c + 1_a \bowtie (b + 1_b) \bowtie 1_a \bowtie c \\ = & a \bowtie \top_b \bowtie \top_c + 1_a \bowtie (b + 1_b) \bowtie c \\ = & a \,\&\, (b \otimes\!\!\gtrdot c) \ . \end{array}$$

The axiomatisation is very similar to the definitions in [2] with some slight differences:

- We do not use explicit type assertions like $a :: T_a$ in our prover input, because the name of the type is not relevant; the only relevant point is, that every element has a type. This can be realized by the following axiom:
$$\forall\, x : \exists\, T : x :: T \ .$$
- The $\bowtie$-Operator is not overloaded, we distinguish between $x \bowtie y$ for elements of the algebra x, y and $T_1 \bowtie T_2$ for types T_1, T_2.

- For the $\bowtie$ Operator we do not assume general associativity, commutativity and distributivity over $+$. Instead of this we only assume some special cases of these properties which we need in the proofs.

The reason for this the brevity of the axiomatisation and the performance; we tried to model our algebra as general as possible in the prover, but especially formulas containing the $\bowtie$ operator would allow way to many deductions, hence the search tree of the prover becomes too big to get an answer in an acceptable time.

1.2 Operators

We use the following operators

Prover9 Input	mathematically
`a typed T_a`	$a :: T_a^2$
`a inf b`	$a \sqcap b$
`a join b`	$a \bowtie b$
`T_a tjoin T_b`	$T_a \bowtie T_b$
`a prior b`	$a \& b$
`a rpar b`	$a \otimes\!\!> b$
`a + b`	$a + b$

where the binding strength is descending. In Prover9 this is stated as follows:

```
op(401, infix, "typed").
op(402, infix, "inf").
op(410, infix, "join").
op(411, infix, "tjoin").
op(500, infix, "prior").
op(510, infix, "rpar").
op(600, infix, "+").
```

1.3 Common Input

In all of the proofs we will use the following prover input, which we will abbreviate with `[standard axioms]`:

```
% standard operators
% ------------------

% all elements are typed
exists T (x typed T).

% addition is associative, commutative and idempotent
(x + y) + z = x + (y + z).
x + y = y + x.
x + x = x.
```

```
% addition preserves type
x typed z & y typed z -> (x+y) typed z.

% subsumption order
x <= y <-> y = x + y.

% top and one elements
% --------------------

% top is greatest element
x typed z -> x <= top(z).

% typing of top
top(z) typed z.
top(z1 tjoin z2) = top(z1) join top(z2).

% typing of one
one(z) typed z.
one(z1 tjoin z2) = one(z1) join one(z2).

% abbreviated typing
x typed z -> top(x) = top(z).
x typed z -> one(x) = one(z).
```

Note that variables beginning with u–z are all-quantified in PROVER9 whereas other variables are handled as constants.

1.4 Auxiliary equation

We show:

$$a \,\&\, b + 1_{a \bowtie b} = a \,\&\, (b + 1_b)$$

This is proved by the following input:

{ Assumptions }

```
[standard axioms]

% joins
% -----

% distributivity of the join over addition
x join (y1 + y2) = x join y1 + x join y2.

% typing of join
x typed z1 & y typed z2 -> (x join y) typed (z1 tjoin z2).

```

```
% prioritisation
% --------------

% (without resulting type)
x prior y = x join top(y) + one(x) join y.
```

{ Goals }

```
% auxiliary equation for distributive law
u prior v + one(u join v) = u prior (v + one(v)).
```

1.5 Step 1

In the first step we show

$$(a \mathbin{\&} b) \mathbin{\otimes\!\!>} (a \mathbin{\&} c) \;=\; \begin{array}{ll} a \bowtie \top_b \bowtie a \bowtie \top_c & + \, a \bowtie \top_b \bowtie 1_a \bowtie c \, + \\ 1_a \bowtie (b + 1_b) \bowtie a \bowtie \top_c & + \, 1_a \bowtie (b + 1_b) \bowtie 1_a \bowtie c \end{array}$$

We use the definition of prioritisation/pareto and the distributivity of joins over addition.

{ Assumptions }

```
[standard axioms]

% joins
% -----

% distributivity
x join (y1 + y2) = x join y1 + x join y2.
(x1 + x2) join y = x1 join y + x2 join y.

% typing of join
x typed z1 & y typed z2 -> (x join y) typed (z1 tjoin z2).

% prioritisation and pareto
% -------------------------

% prioritisation and resulting type
x prior y = x join top(z2) + one(z1) join y.
x typed z1 & y typed z2 -> (x prior y) typed (z1 tjoin z2).

% Right-Semipareto (without resulting type)
x typed z1 & y typed z2 -> x rpar y = (x + one(z1)) join y.

% auxiliary equation
% ------------------
a prior b + one(A tjoin B) = a prior (b + one(B)).
```

```

% To show
% =======

% Left side of equation
left = (a prior b) rpar (a prior c).

% intermediate step 1
step1 = ((a       join top(a))        join  (a       join top(c))   +
          (a       join top(b))        join  (one(a) join c))       +
         ((one(a) join (b + one(b)))  join  (a       join top(c))   +
          (one(a) join (b + one(b)))  join  (one(a) join c)).
```

{ Goals }

```
left = step1.
```

1.6 Step 2

In the second step we show

$$\begin{array}{l} a \bowtie \top_b \bowtie a \bowtie \top_c \qquad + a \bowtie \top_b \bowtie 1_a \bowtie c + \\ 1_a \bowtie (b+1_b) \bowtie a \bowtie \top_c + 1_a \bowtie (b+1_b) \bowtie 1_a \bowtie c \end{array} = a \bowtie \top_b \bowtie \top_c + 1_a \bowtie (b+1_b) \bowtie c$$

We use that joins on the same type are infimas, the isotony on joins, and a special case of commutativity and associativity. Unfortunately, assuming general axioms for that does not work.

{ Assumptions }

```
[standard axioms]

% joins
% -----

% special case of commutivity/associativity
(x1 join x2) join (x3 join x4) = (x1 join x3) join (x2 join x4).

% join on equivalent types is the infimum
x1 typed z & x2 typed z -> x1 join x2 = x1 inf x2.

% isotony on joins (follows from subsumption order and distributivity)
x1 <= x2 -> x1 join y <= x2 join y.
y1 <= y2 -> x join y1 <= x join y2.
x1 <= x2 & y1 <= y2 -> x1 join y1 <= x2 join y2.

% To show
% =======
```

```

% intermediate step 1
step1 = ((a      join top(b))       join  (a      join top(c))  +  % S1.1
         (a      join top(b))       join  (one(a) join c))      +  % S1.2
        ((one(a) join (b + one(b))) join  (a      join top(c))  +  % S1.3
         (one(a) join (b + one(b))) join  (one(a) join c)).        % S1.4

% intermediate step 2
step2 = a      join (top(b)       join top(c)) +  % S2.1
        one(a) join ((b + one(b)) join c).        % S2.2

% This lemmas can be proved one by one (NOT all at once)

% Lemma 1 (S1.1 + S2.2 = S2.1)
((a join top(b)) join (a join top(c)) +
 (a join top(b)) join (one(a) join c)) = a join (top(b) join top(c)).

% Lemma 2 (S1.3 <= S2.1)
(one(a) join (b + one(b))) join (a join top(c)) <=
     a join (top(b) join top(c)).

% Lemma 3 (S1.4 = S2.2)
((a join top(b)) join (a join top(c)) +
 (a join top(b)) join (one(a) join c)) = a join (top(b) join top(c)).
```

{ Goals }

```
step1 = step2.
```

1.7 Step 3

In the final step we show

$$a \bowtie \top_b \bowtie \top_c + 1_a \bowtie (b + 1_b) \bowtie c = a \,\&\, (b \mathbin{\otimes\!\!>} c)$$

We use the definition of prioritisation/pareto and type-propagation of joins is used.

{ Assumptions }

```
[standard axioms]

% joins
% -----

% type of join
x typed z1 & y typed z2 -> (x join y) typed (z1 tjoin z2).

```

```
% prioritisation and pareto
% -------------------------

% prioritisation (without resulting type)
x prior y = x join top(y) + one(x) join y.

% Right-Semipareto (with resulting type)
x rpar y = (x + one(z1)) join y.
x typed z1 & y typed z2 -> (x rpar y) typed (z1 tjoin z2).

% To show
% =======

% intermediate step 2
step2 = a join (top(b) join top(c)) + one(a) join ((b + one(b)) join c).

% Right side of equation
right = a prior (b rpar c).
```

{ Goals }

```
step2 = right.
```

In summary, `left = right` has been shown and with this the theorem is proved.

References

1. W. McCune: `Prover9` and `Mace4`. `http://www.cs.unm.edu/~mccune/mace4/`
2. B. Möller, P. Roocks, M. Endres: An Algebraic Calculus of Database Preferences. In: J. Gibbons, P., Nogueira (eds.): Mathematics of Program Construction. LNCS 7342. Springer 2012, 241–262