Proving By Simplification

Bruno Buchberger, Mircea Marin
Research Institute for Symbolic Computation
A4232 Schloss Hagenberg, Austria
Bruno.Buchberger@RISC.uni−linz.ac.at
Mircea.Marin@RISC.uni−linz.ac.at

This notebook describes a simplification prover in natural deduction style. The prover is mainly a conditional term rewriting system used to generate abstract proof objects, i.e. structured data that contain enough information to describe all the individual steps in a proof. These proof objects can be regarded as standalone proofs, or can be further integrated into more complex proofs that use simplification for intermediary proofs. The prover is written in *Mathematica* 3.0, and provides a viewer that converts proof objects in nested *Mathematica* cells, which are natural language, easy−to−read versions of proofs.

1. Overview of the Prover

The main task of the simplification prover is to generate proofs for equalities modulo a given set of equalities. A simplification proof consists of a sequence of applications of the replacement and substitution rules on each of the terms of the equality to be proven, until identical terms are obtained. Two problems may occur:

◇ non−termination of the rewriting process, and
◇ failure in proving a valid equality.

We avoid the first problem by imposing a simplification order on the sequence of terms which are rewritten. In the second case the prover returns the simplified equality formula which may be passed to a different prover, for example an induction prover.

Notation

We give a list of the most important notationsused when proving by simplification:

◇ we denote equality by "≡";
◇ we denote lists by angle brackets "⟨ ⟩";
◇ we use overbars for denoting sequence symbols: $\overline{x}$;
◇ the quantified variables are written in a list subscripted to the quantifier, even if there is only one variable.
◇ equalities are labeled formulae written in one of the forms:

$$\langle \texttt{label}, \forall_{\langle \overline{V} \rangle} \texttt{lhs} \equiv \texttt{rhs} \rangle,$$
$$\langle \texttt{label}, \texttt{lhs} \equiv \texttt{rhs} \rangle,$$
$$\langle \texttt{label}, \forall_{\langle \overline{V} \rangle} \texttt{Condition} \Rightarrow \texttt{lhs} \equiv \texttt{rhs} \rangle,$$

where `label` is a string, `lhs, rhs` are terms with variables ranging over $\overline{V}$, and `Condition` is a first order logic formula containing type information for the variables occuring in the equality. The second form is a variant of the first form with an empty list of quantified variables.

For a detailed description of the language in the Theorema project, see (Buchberger, Kriftner 1997).

1.1. Quick Introduction by Examples

■ Example 1

Suppose we want to prove the formula

```
(C)  (a ⊕ c) ⊗ ( b ⊕ 0 ) ≡ (b ⊗ c) ⊕ ( b ⊗ a )
```

under the assumptions

```
(⊕.assoc) ∀⟨x,y,z⟩ (x⊕(y⊕z)) ≡ ((x⊕y)⊕z),
(o.left) ∀⟨x⟩ 0⊕x≡x,
(⊕.comm) ∀⟨x,y⟩ x⊕y ≡ y⊕x,
(⊗.distr.left) ∀⟨x,y,z⟩ (x⊗(y⊕z))≡((x⊗y) ⊕ (x⊗z)),
(⊗.distr.right) ∀⟨x,y,z⟩ ((x⊕y)⊗z)≡((x⊗z)⊕(y⊗z)).
```

When called, the simplification prover expects 3 arguments:

• the labeled equality to be proven

```
Theorem := ⟨"C", (a⊕c)⊗(b ⊕ 0) ≡ (b⊗c)⊕(b⊗a)⟩;
```

• a list containing the labeled equalities which are already known

```
Theory :=
⟨⟨"⊕.assoc", ∀⟨x,y,z⟩ (x⊕(y⊕z)) ≡ ((x⊕y)⊕z)⟩,
⟨"⊕.0.left", ∀⟨x⟩ 0⊕x ≡ x⟩,
⟨"⊗.0.left", ∀⟨x⟩ 0⊗x ≡ 0⟩,
⟨"⊗.distr.left", ∀⟨x,y,z⟩ (x⊗(y⊕z)) ≡ ((x⊗y) ⊕ (x⊗z))⟩,
⟨"⊗.distr.right", ∀⟨x,y,z⟩ ((x⊕y)⊗z) ≡ ((x⊗z)⊕(y⊗z))⟩,
⟨"⊕.comm", ∀⟨x,y⟩ (x⊕y) ≡ (y⊕x)⟩,
⟨"⊗.comm", ∀⟨x,y⟩ (x ⊗ y) ≡ (y⊗x)⟩⟩;
```

• the simplification order used for generating the rewrite rule system of the simplifier

```
order := LPOrder;
```

and uses the value of the global variable `ConstantsOrder` for getting the precedences of the function symbols

```
ConstantsOrder := ⟨⟨CirclePlus, CircleTimes⟩⟩;
```

The prover may be called in the following way

```
Proof-by-simplification[Theorem, Theory, order]
```

and generates a notebook containing the proof shown below, but with additional cell brackets so that various parts of the proof can be clicked "open" or "closed" in order to make it easier for keeping an overview on the proof.

PROPOSITION :

(C) $(a \oplus c) \otimes (b \oplus 0) \equiv b \otimes c \oplus b \otimes a$

under the assumptions

(⊕.assoc) $\forall_{\langle x,y,z \rangle}\ x \oplus (y \oplus z) \equiv (x \oplus y) \oplus z$,

(⊕.0.left) $\forall_{\langle x \rangle}\ 0 \oplus x \equiv x$,

(⊗.0.left) $\forall_{\langle x \rangle}\ 0 \otimes x \equiv 0$,

(⊗.distr.left) $\forall_{\langle x,y,z\rangle}\ x \otimes (y \oplus z) \equiv x \otimes y \oplus x \otimes z$,

(⊗.distr.right) $\forall_{\langle x,y,z\rangle}\ (x \oplus y) \otimes z \equiv x \otimes z \oplus y \otimes z$,

(⊕.comm) $\forall_{\langle x,y\rangle}\ x \oplus y \equiv y \oplus x$,

(⊗.comm) $\forall_{\langle x,y\rangle}\ x \otimes y \equiv y \otimes x$.

PROOF:

A proof by simplification works.

Simplification of the lhs term:

$(a \oplus c) \otimes (b \oplus 0) =$ by (⊗.distr.left)

$(a \oplus c) \otimes b \oplus (a \oplus c) \otimes 0 =$ by (⊗.distr.right)

$(a \otimes b \oplus c \otimes b) \oplus (a \oplus c) \otimes 0 =$ by (⊕.assoc)

$a \otimes b \oplus (c \otimes b \oplus (a \oplus c) \otimes 0) =$ by (⊗.distr.right)

$a \otimes b \oplus (c \otimes b \oplus (a \otimes 0 \oplus c \otimes 0)) =$ by (⊕.comm)

$a \otimes b \oplus ((a \otimes 0 \oplus c \otimes 0) \oplus c \otimes b) =$ by (⊕.assoc)

$a \otimes b \oplus (a \otimes 0 \oplus (c \otimes 0 \oplus c \otimes b)) =$ by (⊗.comm)

$a \otimes b \oplus (0 \otimes a \oplus (c \otimes 0 \oplus c \otimes b)) =$ by (⊗.0.left)

$a \otimes b \oplus (0 \oplus (c \otimes 0 \oplus c \otimes b)) =$ by (⊕.0.left)

$a \otimes b \oplus (c \otimes 0 \oplus c \otimes b) =$ by (⊗.comm)

$a \otimes b \oplus (0 \otimes c \oplus c \otimes b) =$ by (⊗.0.left)

$a \otimes b \oplus (0 \oplus c \otimes b) =$ by (⊕.0.left)

$a \otimes b \oplus c \otimes b =$ by (⊗.comm)

$a \otimes b \oplus b \otimes c \rfloor$

Simplification of the rhs term:

$b \otimes c \oplus b \otimes a =$ by (⊕.comm)

$b \otimes a \oplus b \otimes c =$ by (⊗.comm)

$a \otimes b \oplus b \otimes c \rfloor$

■ Example 2

Suppose we want to prove the formula

$$(\smile\_.\mathrm{IC})\quad \left(\langle a, \overline{as}\rangle \asymp \underline{\langle b, \overline{bs}\rangle}\right) \equiv a \smile \left(\langle \overline{as}\rangle \asymp \underline{\langle b, \overline{bs}\rangle}\right)$$

under the assumptions

$(\asymp\_.\ \mathrm{IH})\quad \left(\langle a, \overline{as}\rangle \asymp \underline{\langle \overline{bs}\rangle}\right) \equiv a \smile \left(\langle \overline{as}\rangle \asymp \underline{\langle \overline{bs}\rangle}\right)$
$(\smile\frown)\quad \forall_{\langle x,y,z\rangle}\ x \smile y \frown z \equiv (x \smile y) \frown z$,
$(\_\langle .\,,\ldots\rangle\ )\quad \forall_{\langle x,\bar{y}\rangle}\ \underline{\langle x, \bar{y}\rangle} \equiv \underline{\langle \bar{y}\rangle} \frown x$,
$(\frown\asymp)\quad \forall_{\langle x,y,z\rangle}\ (x \asymp y) \frown z \equiv (x \asymp y \frown z)$,
$(\smile\_)\quad \forall_{\langle x,y\rangle}\ x \smile \underline{y} \equiv \underline{y \frown x}$.

Note:

"$\smile$" denotes the appending operation of an element at the beginning of a list,
"$\asymp$" denotes the concatenation of two lists,
"$\frown$" denotes the prepending operation of an element at the of a list,
"_" denotes the reverting operation of lists.

$$\texttt{Theorem} := \left\langle \texttt{"}\smile\texttt{\_.IC"},\ \left(\langle a,\ \overline{as}\rangle \asymp \underline{\langle b,\ \overline{bs}\rangle}\right) \equiv a \smile \left(\langle \overline{as}\rangle \asymp \underline{\langle b,\ \overline{bs}\rangle}\right)\right\rangle$$

$$\begin{aligned}&\texttt{Theory} :=\\ &\Big\langle \Big\langle \texttt{"}\asymp\texttt{\_.IH"},\ \left(\langle a,\ \overline{as}\rangle \asymp \underline{\langle \overline{bs}\rangle}\right) \equiv a \smile \left(\langle \overline{as}\rangle \asymp \underline{\langle \overline{bs}\rangle}\right)\Big\rangle,\\ &\langle \texttt{"}\smile\frown\texttt{"},\ \forall_{\langle x,y,z\rangle}\ x \smile y \frown z \equiv (x \smile y) \frown z\rangle,\\ &\left\langle \texttt{"\_}\langle .,\ldots\rangle\texttt{"},\ \forall_{\langle x,\bar{y}\rangle}\ \underline{\langle x,\ \bar{y}\rangle} \equiv \underline{\langle \bar{y}\rangle} \frown x\right\rangle,\\ &\langle \texttt{"}\frown\asymp\texttt{"},\ \forall_{\langle x,y,z\rangle}\ ((x \asymp y) \frown z) \equiv (x \asymp (y \frown z))\rangle\Big\rangle;\end{aligned}$$

```
order := RPOrder;

ConstantsOrder := ⟨⟨List, Cup, Cap, CupCap, UnderBar⟩⟩;
```

The call

```
Proof-by-simplification[ Theorem, Theory, order]
```

generates the following notebook

PROPOSITION:

($\smile$_.IC) $(\langle a,\ \overline{as}\rangle \asymp \underline{\langle b,\ \overline{bs}\rangle}) \equiv a \smile (\langle \overline{as}\rangle \asymp \underline{\langle b,\ \overline{bs}\rangle})$

under the assumptions

($\asymp$_.IH) $(\langle a,\ \overline{as}\rangle \asymp \underline{\langle \overline{bs}\rangle}) \equiv a \smile (\langle \overline{as}\rangle \asymp \underline{\langle \overline{bs}\rangle})$,

($\smile\frown$) $\forall_{\langle x,y,z\rangle}\ x \smile y \frown z \equiv (x \smile y) \frown z$,

(_$\langle .,\ldots\rangle$) $\forall_{\langle x,\bar{y}\rangle}\ \underline{\langle x,\ \bar{y}\rangle} \equiv \underline{\langle \bar{y}\rangle} \frown x$,

($\frown\asymp$) $\forall_{\langle x,y,z\rangle}\ (x \asymp y) \frown z \equiv (x \asymp y \frown z)$.

PROOF:

A proof by simplification works.

Simplification of the lhs term:

$\langle a,\ \overline{as}\rangle \asymp \underline{\langle b,\ \overline{bs}\rangle} =$ by (_$\langle .,\ldots\rangle$)

$\langle a,\ \overline{as}\rangle \asymp \underline{\langle \overline{bs}\rangle} \frown b =$ by ($\frown\asymp$)

$(\langle a,\ \overline{as}\rangle \asymp \underline{\langle \overline{bs}\rangle}) \frown b =$ by ($\asymp$_.IH)

$(a \smile (\langle \overline{as}\rangle \asymp \underline{\langle \overline{bs}\rangle})) \frown b =$ by ($\smile\frown$)

$a \smile (\langle \overline{as}\rangle \asymp \underline{\langle \overline{bs}\rangle}) \frown b \rfloor$

Simplification of the rhs term:

$a \smile (\langle \overline{as}\rangle \asymp \underline{\langle b,\ \overline{bs}\rangle}) =$ by (_$\langle .,\ldots\rangle$)

$a \smile (\langle \overline{as}\rangle \asymp \underline{\langle \overline{bs}\rangle} \frown b) =$ by ($\frown\asymp$)

$a \smile (\langle \overline{as}\rangle \asymp \underline{\langle \overline{bs}\rangle}) \frown b \rfloor$

■ Remarks about the Proofs

The proofs are described in natural deduction stile. Each proof step consists of rewriting a term of the equality to be proven via one of the four equational axioms. In the first example, the first five axioms are always used to rewrite from left to right, while the commutativity axioms are used to arrange the terms in lexicographic order. In the second example, the axioms (_⟨.,...⟩) and (≍_.IH) rewrite from left to right, while the axioms (⌢≍), (⌣⌢) rewrite from right to left.

1.2. The Problem

The simplification prover is intended to solve problems of the following type:

Given:

F, a formula of the form `lhs ≡ rhs`, where `lhs` and `rhs` are *ground terms*, i.e. terms that do not contain variables,

L, a finite sequence of equalities of the form

$$(\mathtt{equality}_i)\ \forall_{\langle v_1, v_2, \ldots, v_n\rangle}\ \mathtt{lhs}_i \equiv \mathtt{rhs}_i$$

where lhs_i, $\mathtt{rhs}_i$ are terms with free variables from the set $\{v_1, v_2, \ldots, v_n\}$

Find:

P, a proof by simplification of **F** in the equational theory **L**, if **F** holds in our equational theory.

By proof by simplification we mean a finite sequence of applications of the substitution and replacement rules defined by the equalities in **L** on each of the terms of **F**, until we reach to an identical term

$$\mathrm{lhs} = (\text{ by equality}_{i_1}) = \mathrm{lhs}_1 = (\text{ by equality}_{i_2}) = \mathrm{lhs}_2 = \ldots = (\text{by equality}_{i_p}) = \mathrm{lhs}_{i_p} \equiv \text{term}$$
$$\mathrm{rhs} = (\text{ by equality}_{j_1}) = \mathrm{rhs}_1 = (\text{by equality}_{j_2}) = \mathrm{rhs}_2 = \ldots = (\text{by equality}_{j_q}) = \mathrm{rhs}_{i_q} \equiv \text{term}$$

1.3. The Method

We generate a finite set **R** of directed conditional equations (or conditional rewrite rules) by orienting the set of equations in **L.** Ideally, such a term rewriting system should have the following two properties:

⋄ Noetherianity, or termination. This means there are no infinite sequences $s_1 \mapsto_R s_2 \mapsto_R s_3 \mapsto_R \ldots$
⋄ confluence. A term rewriting system is called confluent if for every *s, r, t* with $r \mapsto_R s$ and $r \mapsto_R t$ there exists a term *u* such that $s \mapsto_R^* u$ and $t \mapsto_R^* u$.

Such a term rewriting system is called canonical and has the nice property that it defines a unique normal form $\Vert t \Vert$ for every term *t* given by:
$t \mapsto_R^* \Vert t \Vert$ and there does not exist a term s such that $\Vert t \Vert \mapsto_R s$.

We guarantee the termination property of **R** by orienting the equations with respect to a simplification ordering " ≻" on the ground terms. For each non−definitional equality

$$\langle \text{label}, \forall_{\langle V\rangle} \mathrm{lhs}_i \equiv \mathrm{rhs}_i \rangle \in \mathbf{L}$$

we generate the following rewrite rules

a) $\{\mathtt{lhs}_i \to \mathtt{rhs}_i\}$, if $\mathrm{Vars}(\mathrm{rhs}_i) \subseteq \mathrm{Vars}(\mathrm{lhs}_i)$ and we can decide that $\forall\, \sigma\ (\sigma$ = ground instance of $\mathrm{Vars}(\mathrm{lhs}_i) \Rightarrow \sigma(\mathrm{lhs}_i) \succ \sigma(\mathrm{rhs}_i))$
b) $\{\mathtt{rhs}_i \to \mathtt{lhs}_i\}$, if $\mathrm{Vars}(\mathrm{lhs}_i) \subseteq \mathrm{Vars}(\mathrm{rhs}_i)$ and we can decide that $\forall\, \sigma\ (\sigma$ = ground instance of $\mathrm{Vars}(\mathrm{rhs}_i) \Rightarrow \sigma(\mathrm{rhs}_i) \succ \sigma(\mathrm{lhs}_i))$
c) $\{\mathtt{lhs}_i \to_c \mathtt{rhs}_i\}$, if $\mathrm{Vars}(\mathrm{rhs}_i) \subset \mathrm{Vars}(\mathrm{lhs}_i)$ and $\exists\, \sigma\ (\sigma$ = ground instance of $\mathrm{Vars}(\mathrm{lhs}_i) \wedge \sigma(\mathrm{lhs}_i) \nsucc \sigma($

$\mathrm{rhs}_i))$
d) $\{\mathtt{rhs}_i \to_c \mathtt{lhs}_i\}$, if $\mathrm{Vars}(\mathrm{lhs}_i) \subset \mathrm{Vars}(\mathrm{rhs}_i)$ and $\exists\, \sigma$ (σ = ground instance of $\mathrm{Vars}(\mathrm{lhs}_i) \wedge \sigma(\mathrm{rhs}_i) \not\succ \sigma(\mathrm{lhs}_i))$
e) $\{\mathtt{lhs}_i \to_c \mathtt{rhs}_i, \mathtt{rhs}_i \to_c \mathtt{lhs}_i\}$, if $\mathrm{Vars}(\mathrm{rhs}_i) = \mathrm{Vars}(\mathrm{lhs}_i)$ and the formula $\forall \sigma$ (σ = ground instance of $\mathrm{Vars}(\mathrm{lhs}_i) \Rightarrow \sigma(\mathrm{lhs}_i) \succ \sigma(\mathrm{rhs}_i)) \vee \forall \sigma$ (σ = ground instance of $\mathrm{Vars}(\mathrm{rhs}_i) \Rightarrow \sigma(\mathrm{rhs}_i) \succ \sigma(\mathrm{lhs}_i))$ can not be proven to hold.

"$\to$" denotes ordinary rewrite rules and "$\to_c$" denotes rewrite rules which are applied only after checking that a term is replaced with a smaller term w.r.t. "$\succ$".
Definitional equalities are always oriented from the newly defined term to its definition.

- **Examples**

In the first introductory example we considered the lexicographic path ordering and the precedence relation $\oplus \prec \otimes$ for function symbols. The corresponding rewrite rule system R contains the following rewrite rules:

$$
\begin{array}{ll}
(\oplus.\mathtt{assoc}) & \forall_{\langle x,y,z\rangle}\ (x \oplus y) \oplus z \to x \oplus (y \oplus z), \\
(\oplus.0.\mathtt{left}) & \forall_{\langle x\rangle}\ 0 \oplus x \to x, \\
(\otimes.0.\mathtt{left}) & \forall_{\langle x\rangle}\ 0 \otimes x \to 0, \\
(\otimes.\mathtt{distr.left}) & \forall_{\langle x,y,z\rangle}\ x \otimes (y \oplus z) \to (x \otimes y) \oplus (x \otimes z), \\
(\otimes.\mathtt{distr.right}) & \forall_{\langle x,y,z\rangle}\ (x \oplus y) \otimes z \to (x \otimes z) \oplus (y \otimes z), \\
(\oplus.\mathtt{comm}) & \forall_{\langle x,y\rangle}\ x \oplus y \to_c y \oplus x, \\
(\otimes.\mathtt{comm}) & \forall_{\langle x,y\rangle}\ x \otimes y \to_c y \otimes x.
\end{array}
$$

For the recursive path ordering considered in the second example, we used the precedence relation 'List' $\prec$ '$\smile$' $\prec$ '$\frown$' $\prec$ '$\asymp$' $\prec$ '_'. The corresponding rewrite rule system R consists of

$$
\begin{array}{l}
(\asymp\_.\mathtt{IH})\ \ \langle a, \overline{as}\rangle \asymp \underline{\langle \overline{bs}\rangle} \to a \smile \left(\langle \overline{as}\rangle \asymp \underline{\langle \overline{bs}\rangle}\right), \\
(\smile\frown)\ \ \forall_{\langle x,y,z\rangle} (x \smile y) \frown z \to x \smile (y \frown z), \\
(\_\langle \cdot, \ldots\rangle)\ \ \forall_{\langle x,\bar{y}\rangle}\ \underline{\langle x, \bar{y}\rangle} \to \underline{\langle \bar{y}\rangle} \frown x, \\
(\_\langle \cdot, \ldots\rangle)\ \ \forall_{\langle x\rangle}\ \underline{x} \to \underline{\langle\rangle} \frown x, \\
(\frown\asymp)\ \ \forall_{\langle x,y,z\rangle}\ x \asymp (y \frown z) \to (x \asymp y) \frown z.
\end{array}
$$

Here, two rewrite rules were generated for the equality

$$(\_\langle \cdot, \ldots\rangle)\ \ \forall_{\langle x,\bar{y}\rangle}\ \underline{\langle x, \bar{y}\rangle} \to \underline{\langle \bar{y}\rangle} \frown x.$$

This behavior is motivated in next section.

1.3.1. Sequence Variables and Rewrite Rules

Sequence variables are placeholders for zero or more ground terms. In order to simplify the decision procedures for cases a) and b) when generating rewrite rules, we first replace each equality

$$\left\langle \mathtt{label}, \forall_{\langle \bar{v}, v, \bar{w}\rangle}\ \mathtt{lhs}_i \equiv \mathtt{rhs}_i \right\rangle \tag{1}$$

with two equations equivalent to (1):

$$
\begin{array}{l}
\left\langle \mathtt{label}, \forall_{\langle \bar{v}, \bar{w}\rangle}\ \mathtt{lhs}_i[\bar{v} \leftarrow] \equiv \mathtt{rhs}_i[\bar{v} \leftarrow] \right\rangle, \\
\left\langle \mathtt{label}, \forall_{\langle \bar{v}, v, \bar{w}\rangle}\ \mathtt{lhs}_i \equiv \mathtt{rhs}_i \right\rangle,
\end{array} \tag{2}
$$

where for the second case we interpret sequence variables as placeholders for sequences of at least one ground term.

1.3.2. The Simplification Algorithm

The algorithm for proving an equality `lhs ≡ rhs` performs the following steps:

1) Compute the derivations `lhs` $= \mathbf{lhs_0} \mapsto_R \ldots \mapsto_R \mathbf{lhs_m} = \Downarrow$ `lhs` $\Downarrow$ and `rhs` $= \mathbf{rhs_0} \mapsto_R \ldots \mapsto_R \mathbf{rhs_n} = \Downarrow$ `rhs` $\Downarrow$

2) Compare " lhs " and " rhs " for syntactical equality. If they are identical, return "proved", otherwise return the equality " lhs " ≡" rhs ".

1.5. Term Orderings

All the simplification orderings used in our prover are so called syntactic orderings, i.e. orderings which are induced by a given precedence ordering $\precsim_{op}$ on function symbols. If c_1, c_2 are constant symbols, then $c_1 \prec_{op} c_2$ iff $c_1 \prec_{lex-order} c_2$. If c is a constant symbol and **f** is an operator, we assume $\mathbf{c} \prec_{op}$ **f**.
The user of the simplification prover may influence the behavior of the simplification prover in two ways:

1) by specifying the precedence ordering on the function symbols of the equational theory, and
2) by specifying the term ordering used to orient the equations.

We considered by now the following categories of simplification orderings

◇ Knuth & Bendix lexicographic orderings: depthOrder, widthOrder, depthFirstOrder, widthFirstOrder, depthLastOrder, widthLastOrder. The quasi–simplification ordering $term_1 \precsim term_2$ considered on the set of terms is

- $term_1 \leq_{depth} term_2$, for depthOrder[$term_1$, $term_2$],
- ($term_1 <_{depth} term_2$) $\vee$(($term_1 =_{depth} term_2$) $\wedge$ firstSubterm[$term_1$] $\precsim_{depth}$ firstSubterm[$term_2$], for firstDepthOrder[$term_1$, $term_2$],
- ($term_1 <_{depth} term_2$) $\vee$(($term_1 =_{depth} term_2$) $\wedge$ lastSubterm[$term_1$] $\precsim_{depth}$ lastSubterm[$term_2$], for lastDepthOrder[$term_1$, $term_2$],
- $term_1 \precsim_{width} term_2$, for widthOrder[$term_1$, $term_2$],
- ($term_1 <_{width} term_2$) $\vee$(($term_1 =_{width} term_2$) $\wedge$ firstSubterm[$term_1$] $\precsim_{width}$ firstSubterm[$term_2$], for firstWidthOrder[$term_1$, $term_2$],
- ($term_1 <_{width} term_2$) $\vee$(($term_1 =_{width} term_2$) $\wedge$ lastSubterm[$term_1$] $\precsim_{width}$ lastSubterm[$term_2$], for lastWidthOrder[$term_1$, $term_2$].

◇ recursive path orderings: RPOrder.
◇ lexicographic path orderings: LPOrder.

For a quick introduction to simplification orderings, see (Derschowitz 1987).

2. *The Main Program for Simplification*

"ProofBySimplification" is the main program for generating proofs by simplification. The syntax call is

```
ProofBySimplification[formula,labeled-formulae,order,flag]
```

and generates a proof object which is an abstract representation of the proof, `formula` is a labeled equality of the form 〈`label, lhs ≡ rhs` 〉 is the labeled equality to be proven, `labeled-formulae` is the list of equalities which are known, `order` is an identifier for the simplification order used to generate the rewrite rule system, and `flag` is a boolean value used to make distinction between a call to the simplifier for generating a stand–alone proof, or for generating a proof object which is later integrated into a more general proof. The only difference to the generated proof object is whether it contains or not the list of equations used during the proof.
A simplification proof object has the stucture

```
〈tag,,
formula,
[labeled-formulae],
 lhsSimplificationObject,
 rhsSimplificationObject,
 proofResult〉,
```

where `tag` is a tag for simplification proofs, `formula` and `labeled-formulae` are the first two arguments of the call, `lhsSimplificationObject` and `rhsSimplificationObject` are sequence structures containing all the reductions steps performed when rewriting `lhsTerm` and `rhsTerm` to normal forms by using the rewrite rule system generated from `labeledEqualityList`. `proofResult` is either *"proved"* if the proof succeeds, or the simplified formula " `lhs` " ≡ " `rhs` " if not.

The simplification prover can be called directly or via Proof-by-simplification. The second approach is used when we want to see the result of the proof by using a viewer, or when we want other representations of the proof object. One can choose among 3 representations: "nested cells", "nested cells notebook" and "proof object". The first two representations are useful when generating *Mathematica* notebooks. The syntax call for Proof-by-simplification is

```
Proof-by-simplification[formula,labeled-formulae,
                          simplification-order,
                         ["nested cells"
                         |"nested cells notebook"
                         |"proof object"] ]
```

If the fourth argument is missing, it is by default considered to be `"nested cells notebook"`.

The main components of the simplification prover are the rewrite rule system generator and the term simplifier.

■ 2.1. The Rewrite Rule System Generator

The syntax call to the rewrite rule system generator is

```
RewriteRules[formulae,simplification-order]
```

where `formulae` is a list of labeled formulae and `order` is the simplification ordering used for orienting the equalities contained in `formulae`.

■ 2.2. The Term Simplifier

The term simplifier has the syntax call

```
Simplified[term, rewrite-rules,simplification-order]
```

and is used to generate the sequence of reductions of `term` to its normal form.

References

(Buchberger, Kriftner 1997) B. Buchberger, F. Kriftner *Theorema: The Language*. Technical Report Theorema.

(Buchberger, Marin, Tomuta 1997) B. Buchberger, M. Marin, E. Tomuta *A Simplification Prover for Noninductive Functors*. Technical Report Theorema–97–01–14. Research Institute for Symbolic Computation, A–4232 Schloss Hagenberg, Austria.

(Derschowitz 1987) N. Derschowitz, Termination of Rewriting, *J.Symbolic Computation* 1987, Vol.3, pp.69–116.

(Wolfram 1996) S. Wolfram *The Mathematica Book*. Wolfram Media.