# section corelaws

/Reference manual/Extended toolkit

## 1. Introduction

This section contains toolkit-independent general-purpose tactics, and toolkit-independent laws, with (some of) their proofs.

section *corelaws*

## 2. Supplied Tactics for the CADiℤ Theorem-prover

The following tactics are supplied as part of the CADiℤ system. They are intended to serve two purposes: a) to be directly useful in proof work; b) to be used as models when users develop their own tactics.

The current version of this library should be regarded as provisional, since both the rules of inference and the tactic language are still in the course of development.

The proof system manipulates sequents. Each sequent used in a proof becomes a goal of the proof process, so in this document the words "goal" and "sequent"

will be used interchangably. Tactics can take several sorts of argument. In this document we proceed "top-down", considering first those tactics which operate on goals, then those which operate on predicates, then those which operate on expressions, and finally those for schema texts and declarations.

Although the tactics are here ordered on the basis of what sort of arguments they have, they also fall fairly neatly into two categories which cut across that basis, namely: a) non-recursive tactics which may fail; b) recursive tactics which are intended never to fail, but which may cause "Nothing changed" to be reported.

The former are: "bindselAbsorb", "unfoldRel", "unfoldFun", "pairInFunction" and "proveWithInduction".

The latter are "flattenGoal", "separateGoals", "flattenPred" and all the simplification tactics whose names begin with "puff".

The first tactics are those whose argument is a single goal.

The tactic "flattenGoal" is used to carry out all those eliminations on a single goal which cause as many as possible of the predicates in the goal to become separate antecedents or consequents, but without producing more than one subgoal.

The tactic takes a single argument, which must be a sequent, and which is bound to the joker "g". The tactic is to be called recursively, and the label "repeat" is declared as the recursion point. Then five alternative possibilities are considered, namely: a) we have finished; b) we can do something with a consequent; c) we can do something with an antecedent; d) we can do something with a declaration; e) the case which is always true.



The first of these cases is where the "axiom" command succeeds. This would otherwise be generated automatically, but in that case the "axiom" node is always displayed in the proof tree, even when "apply tactic" is being used, and there can be cases where the number of such nodes is embarrassingly high. Putting it here means that the "axiom" command is treated like any other command in the tactic.

In the next two of the five possibilities, the appropriate flattening action is carried out and the recursion is taken.

In the fourth case, each declaration is selected for normalization, and if this causes the "axiom" command to be offered, this is accepted and the whole tactic will succeed. This will happen if the result of normalisation is to produce as an antecedent a membership predicate which already occurs as a consequent. The normalization is attempted twice because the two predicates generated as antecedents may be different, one asserting a membership of the stated declarand set, the other asserting membership of the carrier set concerned. If "axiom" is not offered, however, this alternative in which normalization was done will also not succeed, so the result of the whole tactic will be obtained from alternatives without the normalization.

In the final case the tactic stops with a success. Since alternations are tried in the order they are written, the first complete tactic success will be one where either the "axiom" command has been offered, and the goal proved, or all the possible eliminations have been done, until finally a skip has been the only possibility left. The exclamation mark for a "curtail" is put in just to make sure that there is no backtracking, although this will never be attempted if everything goes according



to plan. The tactic as a whole always succeeds. If it achieves nothing, "Nothing changed" will be reported.

$$
\begin{array}{l}
\mathit{flattenGoalgoalg} \mid \mathit{recrepeat} \bullet \\
\quad !(\ \text{“}\mathit{axiom}\text{”}\ 0 \lor \\
\quad (\mathit{pat}_{\mathit{goal}}\mathit{predp} \mid\!\vdash ?\ p \bullet \\
\qquad !(\ \text{“}\mathit{predication}\text{”}\ p;\ \mathit{repeat} \lor \\
\qquad \mathit{matchp} :: \\
\qquad\quad \mid \neg \_\mathit{pred} \bullet \\
\qquad\quad \mid \_\mathit{pred} \lor \_\mathit{pred} \bullet \\
\qquad\quad \mid \_\mathit{pred} \Rightarrow \_\mathit{pred} \bullet \\
\qquad\quad \mid \forall \_\mathit{stxt} \bullet \_\mathit{pred} \bullet \ \text{“}\mathit{elimination}\text{”}\ p :: \\
\qquad\quad \mid \mathit{false} \bullet \ \text{“}\mathit{thin}\text{”}\ p :: \\
\qquad\quad .\ ;\ \mathit{repeat})) \lor \\
\quad (\mathit{pat}_{\mathit{goal}}\mathit{predp} \mid\mid p \vdash ?\ \bullet \\
\qquad !(\ \text{“}\mathit{predication}\text{”}\ p;\ \mathit{repeat} \lor \\
\qquad \mathit{matchp} :: \\
\qquad\quad \mid \neg \_\mathit{pred} \bullet \\
\qquad\quad \mid \_\mathit{pred} \land \_\mathit{pred} \bullet \\
\qquad\quad \mid \exists \_\mathit{stxt} \bullet \_\mathit{pred} \bullet \ \text{“}\mathit{elimination}\text{”}\ p :: \\
\qquad\quad \mid \mathit{true} \bullet \ \text{“}\mathit{thin}\text{”}\ p :: \\
\qquad\quad .\ ;\ \mathit{repeat})) \lor \\
\quad (\mathit{pat}_{\mathit{goal}}\mathit{declvar} \mid \mathit{var} \vdash ?\ \bullet \\
\qquad \text{“}\mathit{normalization}\text{”}\ \mathit{var};\ \text{“}\mathit{normalization}\text{”}\ \mathit{var};\ \text{“}\mathit{axiom}\text{”}\ 0) \lor \\
\quad \mathit{skip})
\end{array}
$$

The tactic "separateGoals" takes a single argument, which must be a sequent, and which is bound to the joker "g". The label "repeat" is declared as the recursion point. First the tactic calls "flattenGoal", then it carries out those eliminations which produce more than one subgoal. Where that is successful, the recursive call is made on both of the subgoals. "separateGoals" can thus generate any number of subgoals. The tactic always succeeds. If it achieves nothing, "Nothing changed" will be reported.

$$
\begin{array}{l}
separateGoals goalg \mid recrepeat \bullet \\
\quad \text{“}apply\ tactic\text{”}\ 0\ \text{“}flattenGoal\text{”}; \\
\quad !((pat_{goal} predp \mid\mid p \vdash ?\ \bullet\ \text{“}elimination\text{”}\ p;\ (repeat \mid\mid repeat)) \vee \\
\quad (pat_{goal} predp \mid \vdash ?\ p \bullet\ \text{“}elimination\text{”}\ p;\ (repeat \mid\mid repeat)) \vee \\
\quad skip)
\end{array}
$$

Next we come to tactics each of whose argument is a single predicate.

The tactic bindselAbsorb is a dedicated auxiliary tactic used by some of those which follow. It assumes its predicate parameter is a conjunction of any number of equalities, where the left-hand side of each equality is in the form of a selection from a tuple, which may be absorbed. The purpose of the tactic is to carry out all these absorptions.



$$
\begin{array}{l}
bindselAbsorbpredp \mid \\
\quad matchp \\
\quad :: expre1 \mid e1 = \_expr \bullet \text{“absorption”}\ e1 \\
\quad :: predp1, p2 \mid p1 \wedge p2 \bullet \\
\qquad \text{“apply tactic”}\ p1\ \text{“bindselAbsorb”};\ \text{“apply tactic”}\ p2\ \text{“bindselAbsorb”} \\
\quad :: .
\end{array}
$$

The tactic "unfoldRel" is used to expand a predicate which expresses whether some explicitly stated term or tuple of terms is a member of a named comprehension. The named comprehension is frequently a relational operator.

The name is assumed to have been declared as equal to a comprehension, whose expression part is the tuple of the bound names of the comprehension. The name of the comprehension is expanded to give the comprehension explicitly, and the membership is also expanded. This yields an existential quantification, containing an equality between the actual arguments to the comprehension and their formal names in the comprehension. Expanding this gives us two separate equalities, and the absorption step selects the first and second component, respectively, of the pair formed by the two arguments. This then allows the one-pointing to remove the bound variables of the quantification, and the final absorptions discard the quantification itself and a true condition which gets generated. The alternative skips at the end allow the tactic to succeed in certain cases where the assumptions of use are not fully met.

$$
\begin{array}{l}
unfoldRelpredp \mid \\
\quad matchp :: exprr \mid \_expr \in r \bullet \\
\qquad \text{“expansion”}\ r :: . \ ; \ -- \ get\ the\ comprehension \\
\quad \text{“expansion”}\ p; \ -- \ expand\ the\ membership \\
\quad matchp :: predequ; \ expre1 \mid \qquad\qquad \exists \_stxt \bullet equas(e1 = (\_exprs)) \bullet \\
\qquad matche1 ::\mid (\_exprs) \bullet \\
\qquad\qquad \text{“expansion”}\ equ; \ \text{“apply tactic”}\ equ\ \text{“bindselAbsorb”} :: \\
\qquad \mid e1 \bullet \text{“expansion”}\ equ :: . ::\mid p \bullet skip :: . \ ; \\
\quad \text{“one-point”}\ p; \ -- \ the\ bound\ variables \\
\quad \text{“absorption”}\ p; \ -- \ the\ exists\ quantifier \\
\quad !(\text{“absorption”}\ p \lor skip); \ -- \ the\ final\ true\ conjunct\ if\ any \\
\quad matchp :: predside \mid side \land \_pred \bullet \\
\qquad \text{“apply tactic”}\ side\ \text{“puffPred”} :: . \ ; \\
\quad !(\text{“absorption”}\ p \lor skip) -- absorb\ the\ side - condition
\end{array}
$$

The largest supplied tactics are the general-purpose simplifying procedures whose names begin with "puff". They make frequent recursive calls on themselves and on each other. They are written without any assumption of the presence of toolkit definitions. Within that constraint, their design aim is to apply all simplifications which one would always want, except perhaps in very special circumstances, but to do nothing else.

"puffPred" takes a single predicate argument, which may appear anywhere and be of any form. The main section of the tactic has a matcher which fans out into some eleven different cases, corresponding to the sort of predicate supplied.

For each of these any applicable immediate simplifications are sought, usually after a recursive call to simplify the constituent elements. Finally, "resolution" and "linear decision" are tried, using the result of the previous simplifications. "puffPred" always succeeds. If it achieves nothing, the report "Nothing changed" is given.

$$
\begin{array}{l}
\mathit{puffPredpredp} \mid \mathit{recrepeat} \bullet \\
\quad \mathit{matchp} :: \\
\quad \mathit{exprq}, r \mid q = r \bullet \\
\qquad \text{“}\mathit{apply\ tactic}\text{”}\ q\ \text{“}\mathit{puffExpr}\text{”};\ \text{“}\mathit{apply\ tactic}\text{”}\ r\ \text{“}\mathit{puffExpr}\text{”}; \\
\qquad !(\ \text{“}\mathit{absorption}\text{”}\ p \lor \\
\qquad \mathit{matchp} :: \\
\qquad\quad \mid (\_\mathit{exprs}) = (\_\mathit{exprs}) \bullet \\
\qquad\quad \text{“}\mathit{expansion}\text{”}\ p;\ \mathit{repeat} :: \\
\qquad\quad \mid \langle\!|\ \_\mathit{decls}\ |\!\rangle = \langle\!|\ \_\mathit{decls}\ |\!\rangle \bullet \\
\qquad\quad \text{“}\mathit{expansion}\text{”}\ p;\ \mathit{repeat} :: \\
\qquad\quad \mid p \bullet \mathit{skip} :: .) :: \\
\quad \mathit{exprq}, r \mid q \in r \bullet \\
\qquad \text{“}\mathit{apply\ tactic}\text{”}\ q\ \text{“}\mathit{puffExpr}\text{”};\ \text{“}\mathit{apply\ tactic}\text{”}\ r\ \text{“}\mathit{puffExpr}\text{”}; \\
\qquad !(\ \text{“}\mathit{absorption}\text{”}\ p;\ \mathit{repeat} \lor \text{“}\mathit{expansion}\text{”}\ p;\ \mathit{repeat} \lor \mathit{skip}) :: \\
\quad \mathit{stxtdec};\ \mathit{predprred} \\
\qquad\quad \mid \exists\ \mathit{dec} \bullet \mathit{prred} \bullet \\
\qquad\quad \mid \forall\ \mathit{dec} \bullet \mathit{prred} \bullet \\
\qquad \text{“}\mathit{apply\ tactic}\text{”}\ \mathit{dec}\ \text{“}\mathit{puffStxt}\text{”}; \\
\qquad \text{“}\mathit{apply\ tactic}\text{”}\ \mathit{prred}\ \text{“}\mathit{puffPred}\text{”}; \\
\qquad !(\ \text{“}\mathit{absorption}\text{”}\ p;\ \mathit{repeat} \lor \text{“}\mathit{one\text{-}point}\text{”}\ p;\ \mathit{repeat} \lor \mathit{skip}) :: \\
\quad \mathit{stxtdec};\ \mathit{predprred} \mid \exists_1\ \mathit{dec} \bullet \mathit{prred} \bullet \\
\qquad \text{“}\mathit{apply\ tactic}\text{”}\ \mathit{dec}\ \text{“}\mathit{puffStxt}\text{”}; \\
\qquad \text{“}\mathit{apply\ tactic}\text{”}\ \mathit{prred}\ \text{“}\mathit{puffPred}\text{”}; \\
\qquad !(\ \text{“}\mathit{absorption}\text{”}\ p \lor \text{“}\mathit{one\text{-}point}\text{”}\ p \lor \text{“}\mathit{expansion}\text{”}\ p);\ \mathit{repeat} :: \\
\quad \mathit{predq} \mid \neg\ q \bullet \\
\qquad \text{“}\mathit{apply\ tactic}\text{”}\ q\ \text{“}\mathit{puffPred}\text{”};\ (\ \text{“}\mathit{absorption}\text{”}\ p \lor \mathit{skip}) :: \\
\quad \mathit{predq}, r \\
\qquad\quad \mid q \land r \bullet \\
\qquad\quad \mid q \lor r \bullet
\end{array}
$$

The tactic "flattenPred" is designed to bring an arbitrary predicate into what is expected to be a more tractable form by systematically absorbing where possiible, distributing quantifiers outwards and distributing negations inwards. It perform approximately the same sorts of simplifications as "flattenGoal". but all within a single predicate, hence its name.

"flattenPred" always succeeds. If it achieves nothing, the report "Nothing changed"

is given.

$$
\begin{array}{l}
\mathit{flattenPredpredp} \mid \mathit{recrepeat} \bullet \\
\quad \mathit{matchp} :: \\
\quad \mathit{expre} \mid e \bullet \\
\quad (\text{“}\mathit{absorption}\text{”}\ p;\ \mathit{repeat} \lor \text{“}\mathit{predication}\text{”}\ p;\ \mathit{repeat} \lor \mathit{skip}) :: \\
\quad \mathit{predprred} \\
\quad \mid \exists\_\mathit{stxt} \bullet \mathit{prred} \bullet \\
\quad \mid \forall\_\mathit{stxt} \bullet \mathit{prred} \bullet \\
\quad \mid \exists_1\_\mathit{stxt} \bullet \mathit{prred} \bullet \\
\quad \text{“}\mathit{apply\ tactic}\text{”}\ \mathit{prred}\ \ \text{“}\mathit{flattenPred}\text{”}; \\
\quad !(\text{“}\mathit{absorption}\text{”}\ p;\ \mathit{repeat} \lor \\
\quad \text{“}\mathit{predication}\text{”}\ p;\ \mathit{repeat} \lor \\
\quad \text{“}\mathit{normalization}\text{”}\ p;\ \mathit{repeat} \lor \mathit{skip}) :: \\
\quad \mathit{predq} \mid \neg\ q \bullet \\
\quad \text{“}\mathit{apply\ tactic}\text{”}\ q\ \ \text{“}\mathit{flattenPred}\text{”}; \\
\quad !(\text{“}\mathit{absorption}\text{”}\ p;\ \mathit{repeat} \lor \text{“}\mathit{predication}\text{”}\ p;\ \mathit{repeat} \lor \\
\quad \text{“}\mathit{de\ Morgan}\text{”}\ q;\ \text{“}\mathit{absorption}\text{”}\ p;\ \mathit{repeat} \lor \mathit{skip}) :: \\
\quad \mathit{predq}, r \mid q \land r \bullet \\
\quad \text{“}\mathit{apply\ tactic}\text{”}\ q\ \ \text{“}\mathit{flattenPred}\text{”};\ \text{“}\mathit{apply\ tactic}\text{”}\ r\ \ \text{“}\mathit{flattenPred}\text{”}; \\
\quad !(\text{“}\mathit{absorption}\text{”}\ p;\ \mathit{repeat} \lor \text{“}\mathit{predication}\text{”}\ p;\ \mathit{repeat} \lor \\
\quad \mathit{matchq} ::\mid \exists\_\mathit{stxt} \bullet \_\mathit{pred} \bullet \\
\quad \text{“}\mathit{distribution}\text{”}\ q;\ \mathit{repeat} :: . \lor \\
\quad \mathit{matchr} ::\mid \exists\_\mathit{stxt} \bullet \_\mathit{pred} \bullet \\
\quad \text{“}\mathit{distribution}\text{”}\ r;\ \mathit{repeat} :: . \lor \mathit{skip}) :: \\
\quad \mathit{predq}, r \mid q \lor r \bullet \\
\quad \text{“}\mathit{apply\ tactic}\text{”}\ q\ \ \text{“}\mathit{flattenPred}\text{”};\ \text{“}\mathit{apply\ tactic}\text{”}\ r\ \ \text{“}\mathit{flattenPred}\text{”}; \\
\quad !(\text{“}\mathit{absorption}\text{”}\ p;\ \mathit{repeat} \lor \text{“}\mathit{predication}\text{”}\ p;\ \mathit{repeat} \lor \\
\quad \mathit{matchq} ::\mid \forall\_\mathit{stxt} \bullet \_\mathit{pred} \bullet
\end{array}
$$



Next we have two tactics each of which takes two predicate arguments.

The tactic "pairInFunction" is designed to show that an inspected predicate, p, of the form $(a, fa) \in f$ is true, using a crossed antecedent, ante, of the form $forallb : s \bullet \exists_1 p : f \bullet p.1 = b$ and where we assume that the condition $a \in s$ is axiomatic or becomes so by absorption.



$pairInFunctionpredante, p \mid$
$matchp :: expra, f, fa \mid (a, fa) \in f \bullet$
$\text{“}expansion\text{”}\ fa;\ \text{“}mu\ tac\text{”}\ fa :: . ;$
$(($
$pat_{cons}predex \mid 1 \bullet$
$matchex :: declsx1;\ expra1, fa1, f1 \mid$
$\exists_1 x1 \mid (a1, fa1) \in f1 \bullet true \bullet$
$\text{“}quantification\ tac\text{”}\ \text{“}\{1\}\text{”}\ a1\ ante;$
$matchante :: predante1, ante2 \mid ante1 \wedge ante2 \bullet$
$\text{“}elimination\text{”}\ ante;\ \text{“}one\text{-}point\text{”}\ ante2;\ \text{“}absorption\text{”}\ ante2;$
$matchante2 ::$
$predante21, ante22 \mid ante21 \Rightarrow ante22 \bullet$
$\text{“}apply\ tactic\text{”}\ ante21\ \text{“}puffPred\text{”} ::$
$\mid ante2 \bullet skip :: . ;$
$!(\text{“}absorption\text{”}\ ante2 \vee skip);$
$\text{“}expansion\text{”}\ ante2;$
$matchante2 ::$
$declppr;\ predant22, ant221;\ exprpr, val;$
$predant222;\ stxtant2221;\ expra2, b2, c2, d2 \mid$
$\exists ppr \bullet$
$(ant22as(ant221aspr.1 = val)$
$\wedge\ ant222as(\forall ant2221 \bullet a2 = b2 \Rightarrow c2 = d2)) \bullet$
$\text{“}elimination\text{”}\ ante2;$
$\text{“}elimination\text{”}\ ant22;$
$\text{“}Leibniz\text{”}\ a1\ b2\ ant221;$
$\text{“}expansion\text{”}\ ex;$
$\text{“}quantification\ tac\text{”}\ \text{“}\{1\}\text{”}\ (pr{+}{+}\text{“}.2\text{”})\ ex;$
$matchex :: predex1, ex2 \mid ex1 \vee ex2 \bullet$
$\text{“}elimination\text{”}\ ex;\ \text{“}one\text{-}point\text{”}\ ex2;\ \text{“}absorption\text{”}\ ex2;$

The tactic "proveWithInduction" is designed to do the right call of quantification on a (crossed) induction principle, ind, given an (inspected) target predicate, targ, which takes the form of a universal quantification which the induction principle can prove; then to prove and eliminate the predicate corresponding to the fact that this is what we have done.

$$
\begin{array}{l}
proveWithInductionpredind, targ \mid \\
matchtarg :: \\
declss;\ predp1, p2 \mid \forall\, s \mid p1 \bullet p2 \bullet \\
\text{“}quantification\ tac\text{”}\ \text{“}\{1\}\text{”} \\
(\text{“}\{\text{”}{+}{+}s{+}{+}\text{“}{-}{-}(\text{”}{+}{+}p1{+}{+}\text{“}) \Rightarrow (\text{”}{+}{+}p2{+}{+}\text{“})\}\text{”})ind; \\
matchind :: predi1, i2 \mid i1 \wedge i2 \bullet \text{“}elimination\text{”}\ ind; \\
\text{“}one\text{-}point\text{”}\ i2;\ \text{“}absorption\text{”}\ i2; \\
matchi2 :: predj1, j2 \mid j1 \Rightarrow j2 \bullet \\
\text{“}elimination\text{”}\ i2; \\
(skip \parallel skip) \\
:: . :: . :: .
\end{array}
$$

Next we come to tactics each of whose argument is a single expression.

The tactic "unfoldFun" takes a single argument, which is an application expression. The tactic is designed to expand the application of the function to its argument or arguments. The function is assumed to have been declared as equal to a suitable set, typically a lambda-expression.

The initial expansion of the function application creates a mu-expression, which the tactic then attempts to simplify. It separates out the bound variables which are equated to the arguments, so that "one-point" can then carry the substitution through.

In the simple cases, the condition part of the mu-expression will be true. If the function is not recognised as being total, however, this will not be so. In that case, the final "absorption" will not succeed, and the skip is taken instead. The user must then reduce that predicate by other means, in order to absorb it away.

$$
\begin{array}{l}
unfoldFunexpre \mid \\
\quad \text{“expansion”}\ e;\ --\ generate\ the\ mu-expression \\
\quad matche :: predprred;\ exprop \mid \\
\quad\quad \mu\_decl \mid prredas(\_expr \in op) \bullet \_expr \bullet \\
\quad \text{“expansion”}\ op;\ --\ get\ the\ function \\
\quad \text{“expansion”}\ prred;\ --\ expand\ the\ membership\ of\ lambda \\
\quad matchprred :: predequ \mid \exists \_stxt \bullet equ \bullet \\
\quad \text{“expansion”}\ equ;\ --\ the\ two\ pairs \\
\quad matchequ :: predp;\ expra, c \mid \\
\quad\quad (pasa = \_expr) \wedge c = \_expr \bullet \\
\quad \text{“absorption”}\ a\ c;\ --\ tuple\ selection,\ separating\ args\ from\ results \\
\quad matchp :: expre1 \mid (e1 = (\_exprs)) \bullet \\
\quad\quad matche1 ::\mid (\_exprs) \bullet \\
\quad\quad\quad \text{“expansion”}\ p;\ \text{“apply tactic”}\ p\ \text{“bindselAbsorb”} :: \\
\quad\quad \mid e1 \bullet \text{“expansion”}\ p :: . ::\mid p \bullet skip :: . :: . :: . ; \\
\quad \text{“one-point”}\ prred;\ --\ within\ exists \\
\quad \text{“absorption”}\ prred :: . ;\ --\ drop\ exists\ quantifier \\
\quad \text{“one-point”}\ e;\ --\ the\ mu-expression \\
\quad matche :: predside \mid \mu \mid side \bullet \_expr \bullet \\
\quad \text{“apply tactic”}\ side\ \text{“puffPred”} :: . ; \\
\quad !(\text{“absorption”}\ e \vee skip) --\ the\ mu\ if\ possible
\end{array}
$$

”puffExpr” is the expression counterpart of ”puffPred”. It takes a single expression argument, which may appear anywhere and be of any form. The main section of the tactic has a matcher which fans out into over twenty different cases,



corresponding to the sort of expression supplied. For each of these any applicable immediate simplifications are sought, usually after a recursive call to simplify the constituent elements. "puffExpr" always succeeds. If it achieves nothing, the report "Nothing changed" is given.

$$
\begin{array}{l}
puffExprexprt \mid recrepeat \bullet matcht :: \\
\quad \mid \theta\_expr \bullet \text{“}expansion\text{”}\ t :: \\
\quad exprp \mid p.1 \bullet \\
\qquad\qquad \mid p.2 \bullet \\
\qquad \text{“}apply\ tactic\text{”}\ p\ \ \text{“}puffExpr\text{”};\ !(\text{“}absorption\text{”}\ t \lor skip) :: \\
\quad exprses \mid (es) \bullet \text{“}apply\ tactic\text{”}\ es\ \ \text{“}puffExprs\text{”} :: \\
\quad declsds \mid \langle\!|\ ds\ |\!\rangle \bullet \text{“}apply\ tactic\text{”}\ ds\ \ \text{“}puffConstDecls\text{”} :: \\
\quad expre \mid \mathbb{P}\, e \bullet \text{“}apply\ tactic\text{”}\ e\ \ \text{“}puffExpr\text{”} :: \\
\quad exprp, q \mid p \times q \bullet \\
\qquad \text{“}apply\ tactic\text{”}\ p\ \ \text{“}puffExpr\text{”};\ \ \text{“}apply\ tactic\text{”}\ q\ \ \text{“}puffExpr\text{”} :: \\
\quad predp;\ exprq, r \mid \text{if}\ p\ \text{then}\ q\ \text{else}\ r \bullet \\
\qquad \text{“}apply\ tactic\text{”}\ p\ \ \text{“}puffPred\text{”}; \\
\qquad (\text{“}absorption\text{”}\ t;\ repeat \lor \\
\qquad \text{“}apply\ tactic\text{”}\ q\ \ \text{“}puffExpr\text{”};\ \ \text{“}apply\ tactic\text{”}\ r\ \ \text{“}puffExpr\text{”}; \\
\qquad (\text{“}absorption\text{”}\ t \lor skip)) :: \\
\quad stxts;\ exprct \mid \{s \bullet ct\} \bullet \\
\qquad\qquad \mid \lambda\, s \bullet ct \bullet \\
\qquad \text{“}apply\ tactic\text{”}\ s\ \ \text{“}puffStxt\text{”}; \\
\qquad \text{“}apply\ tactic\text{”}\ ct\ \ \text{“}puffExpr\text{”}; \\
\qquad !(\text{“}absorption\text{”}\ t \lor \text{“}one\text{-}point\text{”}\ t;\ repeat \lor skip) :: \\
\quad stxts;\ exprct \mid \mu\, s \bullet ct \bullet \\
\qquad\qquad \mid \text{let}\ s \bullet ct \bullet \\
\qquad \text{“}apply\ tactic\text{”}\ s\ \ \text{“}puffStxt\text{”}; \\
\qquad \text{“}apply\ tactic\text{”}\ ct\ \ \text{“}puffExpr\text{”}; \\
\qquad !(\text{“}absorption\text{”}\ t;\ repeat \lor \text{“}one\text{-}point\text{”}\ t;\ repeat \lor skip) :: \\
\quad exprfun, args \mid fun\ args \bullet \\
\qquad !(\text{“}evaluation\text{”}\ t \lor \\
\qquad \text{“}apply\ tactic\text{”}\ fun\ \ \text{“}puffExpr\text{”}; \\
\qquad !(\text{“}expansion\text{”}\ t;
\end{array}
$$

$$
\begin{array}{l}
puffExprsexprsts \mid matchts :: expre;\ exprses \mid e, es \bullet \\
\quad \text{“apply tactic” } e \text{ “puffExpr”}; \\
\quad \text{“apply tactic” } es \text{ “puffExprs”} ::\mid ts \bullet skip :: .
\end{array}
$$

$$
\begin{array}{l}
puffStxtstxts \mid \\
\quad matchs \\
\quad :: declsds \mid ds \mid \_pred \bullet \\
\quad \text{“apply tactic” } ds\ s \text{ “puffDecls”}; \\
\quad matchs :: predbarpart2 \mid \_decls \mid barpart2 \bullet \\
\quad \text{“apply tactic” } barpart2 \text{ “puffPred”} \\
\quad :: . \\
\quad :: .
\end{array}
$$

```
puffDeclsdeclsds; stxts |
    matchds
    :: decld; declsds2 | d; ds2 •
        matchd
        :: expre | _name : e •
            "apply tactic" e "puffExpr"
            ; matche
            ::| {_stxt • _expr} • "normalization" d
            ::| {_exprs} • "normalization" d
            ::| e • skip
            :: .
        :: expre | _name == e • "apply tactic" e "puffExpr"
        :: expre | e •
            "apply tactic" e "puffExpr";
            !("distribution" d ∨ skip)
        :: . ;
        matchds
            :: decldd; declsdds2 | dd; dds2 •
            "apply tactic" dds2 s "puffDecls"
            :: .
    ::| • skip
    :: .
```

$$
\begin{array}{l}
puffConstDeclsdeclsds \mid \\
\quad matchds \\
\quad :: expre;\ declsds2 \mid \_name == e;\ ds2 \bullet \\
\qquad\quad \text{“apply tactic”}\ e\ \text{“puffExpr”};\ \text{“apply tactic”}\ ds2\ \text{“puffConstDecls”} \\
\quad ::\mid \bullet\ skip \\
\quad :: .
\end{array}
$$

# 3. Commutation Laws

## 3.1. Description

We begin with equality, then go on with the rules of commutation of predicates which we can represent using empty schemas as predicates.

## 3.2. Laws

com1 ==

$$[X] \vdash ?\ \forall t, u : X \bullet t = u \Leftrightarrow u = t$$

Proved by a single call of ”puffPred” on the consequent. com2 ==

$$\vdash ?\ \forall p, q : \mathbb{P}[] \bullet p \wedge q \Leftrightarrow q \wedge p$$

com3 ==

$$\vdash? \ \forall p, q : \mathbb{P}[] \bullet p \vee q \Leftrightarrow q \vee p$$

com4 ==

$$\vdash? \ \forall p, q : \mathbb{P}[] \bullet (p \Leftrightarrow q) \Leftrightarrow (q \Leftrightarrow p)$$

The above three laws can be proved by a single call of "commutation" on one of the arguments to the "iff", followed by a single call of "puffPred" on the whole consequent.

# 4. Absorption Laws

## 4.1. Description

We begin with true and false, then go on with the rules of absorption of predicates which we can represent using empty schemas as predicates.

All the following laws can be proved by a single call of "puffPred" on the consequent.

## 4.2. Laws

abs1 ==

$$\vdash? \ \neg \mathit{false} \Leftrightarrow \mathit{true}$$



abs2 ==

$$\vdash? \neg \mathit{true} \Leftrightarrow \mathit{false}$$

abs3 ==

$$\vdash? \forall p : \mathbb{P}[] \bullet \neg \neg p \Leftrightarrow p$$

absAnd1 ==

$$\vdash? \forall p : \mathbb{P}[] \bullet p \wedge \mathit{false} \Leftrightarrow \mathit{false}$$

absAnd2 ==

$$\vdash? \forall p : \mathbb{P}[] \bullet \mathit{false} \wedge p \Leftrightarrow \mathit{false}$$

absAnd3 ==

$$\vdash? \forall p : \mathbb{P}[] \bullet p \wedge \neg p \Leftrightarrow \mathit{false}$$

absAnd4 ==

$$\vdash? \forall p : \mathbb{P}[] \bullet \neg p \wedge p \Leftrightarrow \mathit{false}$$

absAnd5 ==

$$\vdash? \forall p : \mathbb{P}[] \bullet p \wedge \mathit{true} \Leftrightarrow p$$

absAnd6 ==

$$\vdash? \forall p : \mathbb{P}[] \bullet \mathit{true} \wedge p \Leftrightarrow p$$



absAnd7 ==

$$\vdash?\ \forall\, p : \mathbb{P}[] \bullet p \wedge p \Leftrightarrow p$$

absOr1 ==

$$\vdash?\ \forall\, p : \mathbb{P}[] \bullet p \vee \mathit{true} \Leftrightarrow \mathit{true}$$

absOr2 ==

$$\vdash?\ \forall\, p : \mathbb{P}[] \bullet \mathit{true} \vee p \Leftrightarrow \mathit{true}$$

absOr3 ==

$$\vdash?\ \forall\, p : \mathbb{P}[] \bullet p \vee \neg\, p \Leftrightarrow \mathit{true}$$

absOr4 ==

$$\vdash?\ \forall\, p : \mathbb{P}[] \bullet \neg\, p \vee p \Leftrightarrow \mathit{true}$$

absOr5 ==

$$\vdash?\ \forall\, p : \mathbb{P}[] \bullet p \vee \mathit{false} \Leftrightarrow p$$

absOr6 ==

$$\vdash?\ \forall\, p : \mathbb{P}[] \bullet \mathit{false} \vee p \Leftrightarrow p$$

absOr7 ==

$$\vdash?\ \forall\, p : \mathbb{P}[] \bullet p \vee p \Leftrightarrow p$$

absImplies1 ==

$$\vdash? \ \forall p : \mathbb{P}[] \bullet \mathit{false} \Rightarrow p \Leftrightarrow \mathit{true}$$

absImplies2 ==

$$\vdash? \ \forall p : \mathbb{P}[] \bullet p \Rightarrow \mathit{true} \Leftrightarrow \mathit{true}$$

absImplies3 ==

$$\vdash? \ \forall p : \mathbb{P}[] \bullet p \Rightarrow p \Leftrightarrow \mathit{true}$$

absImplies4 ==

$$\vdash? \ \forall p : \mathbb{P}[] \bullet \mathit{true} \Rightarrow p \Leftrightarrow p$$

absImplies5 ==

$$\vdash? \ \forall p : \mathbb{P}[] \bullet p \Rightarrow \mathit{false} \Leftrightarrow \neg\, p$$

absImplies6 ==

$$\vdash? \ \forall p : \mathbb{P}[] \bullet p \Rightarrow \neg\, p \Leftrightarrow \neg\, p$$

absIff1 ==

$$\vdash? \ \forall p : \mathbb{P}[] \bullet (p \Leftrightarrow p) \Leftrightarrow \mathit{true}$$

absIff2 ==

$$\vdash? \ \forall p : \mathbb{P}[] \bullet (p \Leftrightarrow \neg\, p) \Leftrightarrow \mathit{false}$$

absIff3 ==

$$\vdash? \ \forall p : \mathbb{P}[] \bullet (\neg\, p \Leftrightarrow p) \Leftrightarrow \mathit{false}$$

absIff4 ==

$$\vdash? \ \forall p : \mathbb{P}[] \bullet (\mathit{true} \Leftrightarrow p) \Leftrightarrow p$$

absIff5 ==

$$\vdash? \ \forall p : \mathbb{P}[] \bullet (p \Leftrightarrow \mathit{true}) \Leftrightarrow p$$

absIff6 ==

$$\vdash? \ \forall p : \mathbb{P}[] \bullet (\mathit{false} \Leftrightarrow p) \Leftrightarrow \neg\, p$$

absIff7 ==

$$\vdash? \ \forall p : \mathbb{P}[] \bullet (p \Leftrightarrow \mathit{false}) \Leftrightarrow \neg\, p$$

## 5. Theorems about the Natural Numbers

We could here state and prove various theorems about $\mathbb{N}$, both because of their usefulness, and to assist in proving the consistency of the definitions given later in numdefs.

The theorems would mainly be to the effect that $\mathbb{N}$ is an Abelian monoid under $\_ + \_$.

zeroPlusBEqB ==

$$\vdash? \ \forall b : \mathbb{N} \bullet 0 + b = b$$

Proved by induction on b succAPlusB ==

$$\vdash? \ \forall a, b : \mathbb{N} \bullet (a + 1) + b = (a + b) + 1$$

Proved similarly using induction on b PlusCommutes ==

$$\vdash? \ \forall a, b : \mathbb{N} \bullet a + b = b + a$$

Proved by induction on a using the previous two results as lemmas PlusClosed ==

$$\vdash? \ \forall a, b : \mathbb{N} \bullet a + b \in \mathbb{N}$$

PlusAssociates ==

$$\vdash? \ \forall a, b, c : \mathbb{N} \bullet (a + b) + c = a + (b + c)$$

PlusConstInjective ==

$$\vdash? \ \forall a, b, c : \mathbb{N} \mid a + c = b + c \bullet a = b$$

*IT 22-Jan-2002*