Algorithms and Algorithmic Reasoning in Mathematica

Tutorial at the Workshop
"Introduction to Computer Algebra and Applications"
Douala, Cameroon, October 6 - 13, 2017

Bruno Buchberger

Research Institute for Symbolic Computation
Johannes Kepler University, Linz, Austria

Exercise I Look at Mathematica 'Help'

Check the Mathematica Help Facility (Menu 'Help → 'Wolfram Documentation') for one of your favorite topics and play with the examples given there.

For example, if your are intested in machine learning: Search for "machine learning" and go to the examples in the notebook that pops up:

```
trainingset = {1 → "A", 2 → "A", 3.5 → "B", 4 → "B"};
c = Classify[trainingset]
```

ClassifierFunction[Input type: Numerical
Classes: A, B]

Use the classifier function to classify a new unlabeled example:

```
c[2.6]
```

A

Obtain classification probabilities for this example:

```
c[2.6, "Probabilities"]
```

⟨| A → 0.999618, B → 0.000381686 |⟩

.....

Try to understand how to use the Mathematica functions in this section by experimenting with the examples and your own examples.

Exercise 2 Write a Recursive Mathematica Program for Merge Sort

The Task

Use the “pattern match” programming style of Mathematica for your programs:

For example:

```
left[{}] := {}
right[{}] := {}
left[{x_}] := {x}
right[{x_}] := {}
left[{x_, y___, z_}] := Prepend[left[{y}], x]
right[{x_, y___, z_}] := Append[right[{y}], z]
```

Tests:

```
left[{}]
right[{}]
```

```
{}
```

```
{}
```

```
left[{1}]
right[{1}]
```

```
{1}
```

```
{}
```

```
left[{1, 2}]
right[{1, 2}]
```

```
{1}
```

```
{2}
```

```
left[{1, 2, 3}]
right[{1, 2, 3}]
```

```
{1, 2}
```

```
{3}
```

```
left[{1, 2, 3, 4}]
right[{1, 2, 3, 4}]
```

```
{1, 2}
```

```
{3, 4}
```

```
left[{1, 2, 3, 4, 5}]
right[{1, 2, 3, 4, 5}]
```

```
{1, 2, 3}
```

```
{4, 5}
```

Exercise 3 (Try to) Give a Proof of the Correctness of the Program

Prerequisites

In the proof, for reducing writing effort, use abbreviated names ‘mS’, ‘| ...|’, ‘l’, ‘r’, ‘m’ for the long names ‘mergeSort’, ‘length’, ‘left’, ‘right’, ‘merge’.

You will need the predicates ‘isSortedVersion’, ‘isSorted’, ‘containSameElements’ abbreviated by ‘isSV’, ‘isS’, ‘~’:

Use ‘X’, ‘Y’, ‘Z’ etc. as variables for lists.

‘isSortedVersion’:

isSV[X, Y] :⟺ (isS[Y] ∧ X ~ Y).

‘isSorted’:

isS[X] :⟺ X is sorted (in ascending order) (try a definition by giving Mathematica program!)

‘containSameElements’:

X ~ Y :⟺ X and Y contain the same elements the same number of times (try definition by giving Mathematica program!)

Correctness Statement to be proved:

$\forall_X$ isSortedVersion[X, mergeSort[X]].

For the proof, use the following **induction principle:** For any property P,

i**n order to prove** $\forall_X$ P[X]

take X* **arbitrary but fix,**

assume $\forall_Y$ (| Y | < | X* | ⟹ P[X])

and **prove** P[X*].

In our case, the property P is

$P[X] :\Longleftrightarrow \text{isSortedVersion}[X, \text{mergeSort}[X]]$.

In the proof, you may use the following **lemmas** (without proving them):

$|X| \leq 1 \Longrightarrow \text{isSorted}[X]$

$X \sim X, \quad X \sim Y \Longrightarrow Y \sim X, \quad X \sim Y \wedge Y \sim Z \Longrightarrow X \sim Z$

$|X| > 1 \Longrightarrow (\ |\text{left}[X]| < |X| \ \wedge \ |\text{right}[X]| < |X|\ )$

$(\ |X| > 1 \ \wedge \ Y \sim \text{left}[X] \ \wedge \ \text{isSorted}[Y] \ \wedge \ Z \sim \text{right}[X] \ \wedge \ \text{isSorted}[Z]\ )$

$\Longrightarrow (\ \text{merge}[Y, Z] \sim X \ \wedge \ \text{isSorted}[\ \text{merge}[Y, Z]\ ]\ )$

(Instead of proofs, give intuitive arguments why these lemmas are true. Use diagrams!)

Exercise 4 (Try to) Give an Automated Proof for the Correctness of the Program

For this, extend / modify the simple automated reasoner presented and discussed in the lecture.

This exercise, probably, cannot be done in the time available.

Exercise 5 Explore the Equational Theory of Natural Numbers Using the Induction Prover

Initialization

Go to the Mathematica menu point 'Evaluation' and click 'Evaluate Initialization Cells'. Then my induction prover in the appendix will be activated and can be used by calling 'Prove'.

Explore the Theory of 'plus'

Define:

```
$Theory = AxiomsℕplusPlus = {plus[x_, 0] → x, plus[x_, y_⁺] → plus[x, y]⁺}
```

```
{plus[x_, 0] → x, plus[x_, y_⁺] → plus[x, y]⁺}
```

Then you can try to prove interesting properties of 'plus' by calling the induction prover by 'Prove', for example:

```
Prove[forAll[{x}, plus[0, x] ≡ x]] // NestedCellsForm
```

```
NotebookObject[ Untitled–54 ]
```

If you want to see the internal representation of the reasoning tree then just call

```
Prove[forAll[{x}, plus[0, x] ≡ x]]
```

```
RT[Prove, forAll[{x}, plus[0, x] ≡ x], {plus[x_, 0] → x, plus[x_, y_⁺] → plus[x, y]⁺},
 Proved, {RT[ProveBySimplificationOrInduction, forAll[{x}, plus[0, x] ≡ x],
   {}, Proved, {RT[ProveBySimplification, plus[0, 0] ≡ 0, {}, Proved,
     {RT[SimplifyByRewriting, plus[0, 0] ≡ 0, {}, 0 ≡ 0]}], RT[ProveBySimplification,
     plus[0, x⁺] ≡ x⁺, {plus[0, 0] :↦ 0, plus[0, x] :↦ x}, Proved, {RT[SimplifyByRewriting,
       plus[0, x⁺] ≡ x⁺, {plus[0, 0] :↦ 0, plus[0, x] :↦ x}, x⁺ ≡ x⁺]}]}]}]
```

Before you actually call the prover, try to give the proofs "by hand", at least in some first examples.

Try to prove commutativity and associativity. Add the properties you proved to a list **Theoremsℕplus** and study how proofs change if you add more properties to $Theory. Is it always better to have more properties in $Theory? Study proofs that fail, i.e. yield 'Unproved' because simpler theorems should have been proved before. Study the failing proofs and guess from the point where they fail which lemmas should have been proved before. (This is a technique I automated for inventing theorems and algorithms automatically, see for example

B. Buchberger. Towards the Automated Synthesis of a Groebner Bases Algorithm.
RACSAM - Revista de la Real Academia de Ciencias (Review of the Spanish Royal Academy of Science), Serie A: Mathematicas 98(1), pp. 65-75. 2004.

In later proofs one may need a more general form of commutativity:

```
Prove[forAll[{x, y, z}, plus[plus[x, y], z] ≡ plus[plus[x, z], y]]] // NestedCellsForm
```

Adding commutativity may have a bad effect to later proofs. Why? How could one add a version of commutativity that avoids generating proof cycles?

Explore the Theory of 'times'

Add the following axioms

```
AxiomsℕtimesPlus = {times[x_, 0] :→ 0, times[x_, y_⁺] :→ plus[times[x, y], x]}
```

```
{times[x_, 0] :→ 0, times[x_, y_⁺] :→ plus[times[x, y], x]}
```

and start to prove properties like commutativity, associativity, distributivity in a way similar to what you did for the theory of 'plus'.

Think about other equational theories over the naturals and try proving properties using my little induction prover.

If it fails, think about gradually extending the prover to handle more complicated formulae.

Appendix: The Executable Mathematica Program of My Induction Prover for Equalities over the Naturals

Reasoning Trees

Protect Tag 'RT' for Reasoning Trees

We protect the tag 'RT' for reasoning situations represented as tuples:

```
Protect[RT]
```

```
{RT}
```

'Result'

Definition

```
Clear[Result]
Result[RT[reason_, goal_, knowledge_, result_, reasontrees___]] := result
```

Examples

```
RT[ProveBySimplificationOrInduction,
  forAll[{x}, plus[0, x] ≡ x], {plus[x_, 0] → x, plus[x_, y_⁺] → plus[x, y]⁺},
  Proved, {RT[ProveBySimplification, plus[0, x] ≡ x,
    {plus[x_, 0] → x, plus[x_, y_⁺] → plus[x, y]⁺}, Proved, {RT[SimplifyByRewriting,
      plus[0, x] ≡ x, {plus[x_, 0] → x, plus[x_, y_⁺] → plus[x, y]⁺}, x ≡ x]}]}] // Result
```

```
Proved
```

NestedCellsForm

Definition

```
Clear[NestedCellsForm]

NestedCellsForm[reasonTree_] :=
 Notebook[{NF[reasonTree]},
   CellGrouping → Manual, Magnification → 2] // NotebookPut
```

```
Clear[NF]
NF[RT[
reason_,
goal_,
knowledge_,
   result_,
   {reasonTrees__}]] :=

 Module[{c, rt, l},
  c = {ExpressionsCell[reason],
    GoalCell[goal],
    (* Cell["using"], *)
    KnowledgeCell[knowledge],
    (* Cell["yields"], *)
    ResultCell[result]
     (* Cell["Namely:"]*) };
  rt = {reasonTrees}; l = Length[rt];
  Do[AppendTo[c, NF[rt[[i]]] ], {i, l}];
  CellGroupData[c]]

NF[RT[
reason_,
goal_,
knowledge_,
   result_]] :=
 CellGroupData[{
   ExpressionsCell[reason],
   GoalCell[goal],
   (* Cell["using"], *)
   KnowledgeCell[knowledge],
   (* Cell["yields"],*)
   ResultCell[result]
  }]
```

Examples

```
reasoningTree = RT[ProveBySimplificationOrInduction,
  forAll[{x}, plus[0, x] ≡ x], {plus[x_, 0] → x, plus[x_, y_⁺] → plus[x, y]⁺},
  Proved, {RT[ProveBySimplification, plus[0, x] ≡ x,
    {plus[x_, 0] → x, plus[x_, y_⁺] → plus[x, y]⁺}, Proved, {RT[SimplifyByRewriting,
      plus[0, x] ≡ x, {plus[x_, 0] → x, plus[x_, y_⁺] → plus[x, y]⁺}, x ≡ x]}]}]
```

```
RT[ProveBySimplificationOrInduction,
 forAll[{x}, plus[0, x] ≡ x], {plus[x_, 0] → x, plus[x_, y_⁺] → plus[x, y]⁺},
 Proved, {RT[ProveBySimplification, plus[0, x] ≡ x,
   {plus[x_, 0] → x, plus[x_, y_⁺] → plus[x, y]⁺}, Proved, {RT[SimplifyByRewriting,
     plus[0, x] ≡ x, {plus[x_, 0] → x, plus[x_, y_⁺] → plus[x, y]⁺}, x ≡ x]}]}]
```

```
reasoningTree // NestedCellsForm
```

```
NotebookObject[ Untitled-13 ]
```

ExpressionsCell

Definition

```
Clear[ExpressionsCell, GoalCell, KnowledgeCell, ResultCell]
ExpressionsCell[expression_] :=
    Cell[BoxData[ToBoxes[expression]]]
GoalCell[expression_] :=
    Cell[BoxData[ToBoxes[expression]], FontColor → Red]
KnowledgeCell[expression_] :=
    Cell[BoxData[ToBoxes[expression]], FontColor → Magenta]
ResultCell[expression_] :=
    Cell[BoxData[ToBoxes[expression]], FontColor → Blue]
```

Examples

```
GoalCell[0 ≡ 0]
```

```
Cell[BoxData[RowBox[{0, ≡, 0}]], FontColor → ■]
```

Global Variable '$Theory'

In '$Theory' we store a "theory" (rewrite rules) that are echoed in the reasoning tree only at the top level of the reasoning tree and are, however, added to the knowledge at all subsequent reasoning steps.

Simplify by Rewriting

'SimplifyByRewriting'

Definition

```
Clear[SimplifyByRewriting]

SimplifyByRewriting[expression_, {equalities___}] :=

 RT[SimplifyByRewriting,
  expression,
  {equalities},
  (expression //. ($Theory ~Join~ {equalities}))]
```

Examples

```
$Theory = {}
```

```
{}
```

```
SimplifyByRewriting[f[a], {f[x_] :> g[x, x], g[a, b] :> h[a, a, b, b]}]
```

```
RT[SimplifyByRewriting, f[a], {f[x_] :⇢ g[x, x], g[a, b] :⇢ h[a, a, b, b]}, g[a, a]]
```

```
SimplifyByRewriting[f[a], {f[x_] :> g[x, x], g[a, a] :⇢ 25}]
```

```
RT[SimplifyByRewriting, f[a], {f[x_] :⇢ g[x, x], g[a, a] :⇢ 25}, 25]
```

```
SimplifyByRewriting[f[a], {f[x_] :> g[x, x], g[a, a] :⇢ h[a, a, u]}]
```

```
RT[SimplifyByRewriting, f[a], {f[x_] :⇢ g[x, x], g[a, a] :⇢ h[a, a, u]}, h[a, a, u]]
```

```
SimplifyByRewriting[f[a], {f[x_] :> g[x, u], g[a_, b_] :⇢ h[a, a, b, b]}]
```

```
RT[SimplifyByRewriting, f[a], {f[x_] :⇢ g[x, u], g[a_, b_] :⇢ h[a, a, b, b]}, h[a, a, u, u]]
```

```
$Theory = {plus[x_, 0] → x, plus[x_, y_⁺] → plus[x, y]⁺}
```

```
{plus[x_, 0] → x, plus[x_, y_⁺] → plus[x, y]⁺}
```

```
SimplifyByRewriting[plus[0, 0], {plus[x_, 0] → x, plus[x_, y_⁺] → plus[x, y]⁺}]
```

```
RT[SimplifyByRewriting, plus[0, 0], {plus[x_, 0] → x, plus[x_, y_⁺] → plus[x, y]⁺}, 0]
```

```
SimplifyByRewriting[plus[0⁺⁺, 0⁺⁺⁺⁺], {plus[x_, 0] → x, plus[x_, y_⁺] → plus[x, y]⁺}]
```

```
RT[SimplifyByRewriting, plus[(0⁺)⁺, (((0⁺)⁺)⁺)⁺],
 {plus[x_, 0] → x, plus[x_, y_⁺] → plus[x, y]⁺}, (((((0⁺)⁺)⁺)⁺)⁺)⁺]
```

```
SimplifyByRewriting[plus[0⁺⁺, 0⁺⁺⁺⁺⁺⁺⁺⁺], {dec[0] :⇢ 0, dec[x_⁺] :> dec[x] + 1}]
```

```
RT[SimplifyByRewriting, plus[(0⁺)⁺, (((((((0⁺)⁺)⁺)⁺)⁺)⁺)⁺)⁺],
 {dec[0] :⇢ 0, dec[x_⁺] :⇢ dec[x] + 1}, (((((((((0⁺)⁺)⁺)⁺)⁺)⁺)⁺)⁺)⁺)⁺]
```

```
SimplifyByRewriting[dec[plus[0⁺⁺, 0⁺⁺⁺⁺⁺⁺⁺⁺]], {dec[0] :⇢ 0, dec[x_⁺] :> dec[x] + 1}]
```

```
RT[SimplifyByRewriting, dec[plus[(0⁺)⁺, (((((((0⁺)⁺)⁺)⁺)⁺)⁺)⁺)⁺]],
 {dec[0] :⇢ 0, dec[x_⁺] :⇢ dec[x] + 1}, 10]
```

```
SimplifyByRewriting[dec[times[0⁺⁺⁺⁺⁺, 0⁺⁺⁺⁺⁺⁺⁺⁺]], {dec[0] :⇢ 0, dec[x_⁺] :> dec[x] + 1}]
```

```
RT[SimplifyByRewriting, dec[times[((((0⁺)⁺)⁺)⁺)⁺, (((((((0⁺)⁺)⁺)⁺)⁺)⁺)⁺)⁺]],
 {dec[0] :⇢ 0, dec[x_⁺] :⇢ dec[x] + 1}, 40]
```

Examples in Nested Cells Form

```
$Theory = {}
```

```
{}
```

```
SimplifyByRewriting[f[a], {f[x_] :> g[x, x], g[a, b] :> h[a, a, b, b]}] // NestedCellsForm
```

```
NotebookObject[ Untitled–14 ]
```

```
SimplifyByRewriting[f[a], {f[x_] :> g[x, x], g[a, a] :⇢ 25}] // NestedCellsForm
```

```
NotebookObject[ Untitled–15 ]
```

```
SimplifyByRewriting[f[a], {f[x_] :> g[x, x], g[a, a] :> h[a, a, u]}] // NestedCellsForm
```

```
NotebookObject[ Untitled-73 ]
```

```
SimplifyByRewriting[f[a], {f[x_] :> g[x, u], g[a_, b_] :> h[a, a, b, b]}] //
 NestedCellsForm
```

```
NotebookObject[ Untitled-16 ]
```

```
$Theory = {plus[x_, 0] → x, plus[x_, y_⁺] → plus[x, y]⁺}
```

```
{plus[x_, 0] → x, plus[x_, y_⁺] → plus[x, y]⁺}
```

```
SimplifyByRewriting[plus[0, 0], {}] // NestedCellsForm
```

```
NotebookObject[ Untitled-17 ]
```

```
SimplifyByRewriting[plus[0⁺⁺, 0⁺⁺⁺⁺], {}] // NestedCellsForm
```

```
NotebookObject[ Untitled-76 ]
```

```
SimplifyByRewriting[plus[0⁺⁺, 0⁺⁺⁺⁺⁺⁺⁺⁺],
  {dec[0] :> 0, dec[x_⁺] :> dec[x] + 1}] // NestedCellsForm
```

```
NotebookObject[ Untitled-18 ]
```

```
SimplifyByRewriting[dec[plus[0⁺⁺, 0⁺⁺⁺⁺⁺⁺⁺⁺]],
  {dec[0] :> 0, dec[x_⁺] :> dec[x] + 1}] // NestedCellsForm
```

```
NotebookObject[ Untitled-19 ]
```

Transform Formulae (e.g. Equalities) into Rewrite Rules

'RuleFromEquality'

Definition

```
Clear[RuleFromEquality]

RuleFromEquality[forAll[{variables___}, leftExpression_ ≡ rightExpression_]] :=
 IntroduceVariables[{variables}, leftExpression] :↦ rightExpression
(* RuleFromEquality[forAll[{variables___},not[expression_]]]:=
 IntroduceVariables[{variables},expression]:↦ false
    RuleFromEquality[forAll[{variables___},if[premise_,conclusion_]]]:=
  IntroduceVariables[{variables},conclusion]:↦premise
     RuleFromEquality[
      forAll[{variables___},if[and[premise1_,premise2_],conclusion_]]]:=
   IntroduceVariables[{variables},conclusion]:↦and[premise1,premise2]

      RuleFromEquality[forAll[{variables___},expression_]]:=
    IntroduceVariables[{variables},expression]:↦true *)

RuleFromEquality[leftExpression_ ≡ rightExpression_] :=
 leftExpression :↦ rightExpression

(* RuleFromEquality[not[expression_]]:=expression:↦false
    RuleFromEquality[if[premise_,conclusion_]]:=conclusion:↦premise
     RuleFromEquality[if[and[premise1_,premise2_],conclusion_]]:=
   conclusion:↦and[premise1,premise2]
     RuleFromEquality[expression_]:=expression:↦true *)
```

Examples

```
RuleFromEquality /@ {forAll[{x}, f[x] ≡ g[x, x]], g[a, b] ≡ h[a, a, b, b]}
```

```
{f[x_] :↦ g[x, x], g[a, b] :↦ h[a, a, b, b]}
```

```
RuleFromEquality /@ {forAll[{x}, f[x] ≡ g[x, x]], forAll[{x}, g[a, b] ≡ h[a, a, b, b]]}
```

```
{f[x_] :↦ g[x, x], g[a, b] :↦ h[a, a, b, b]}
```

```
RuleFromEquality /@ {forAll[{x}, f[x] ≡ g[x, x]], forAll[{a, b}, g[a, b] ≡ h[a, a, b, b]]}
```

```
{f[x_] :↦ g[x, x], g[a_, b_] :↦ h[a, a, b, b]}
```

```
RuleFromEquality /@ {forAll[{x}, f[x] ≡ g[x, x]], forAll[{a}, g[a, b] ≡ h[a, a, b, b]]}
```

```
{f[x_] :↦ g[x, x], g[a_, b] :↦ h[a, a, b, b]}
```

```
RuleFromEquality /@ {forAll[{x}, f[x] ≡ g[x, x]], forAll[{g, b}, g[a, b] ≡ h[a, a, b, b]]}
```

```
{f[x_] :↦ g[x, x], g_[a, b_] :↦ h[a, a, b, b]}
```

```
RuleFromEquality[plus[0, 0] ≡ 0]
```

```
plus[0, 0] :↦ 0
```

'IntroduceVariables'

Definition

```
Clear[IntroduceVariables]
IntroduceVariables[{}, expression_] := expression
IntroduceVariables[{v1_, v___}, expression_] :=
 IntroduceVariables[{v}, expression /. v1 :> Pattern[v1, 1 Blank[]]]
```

RuleDelayed: Pattern v1 : 1 _ appears on the right-hand side of rule IntroduceVariables[{v1_, v___}, expression_] :↦ IntroduceVariables[{v}, expression /. v1 :↦ v1 : 1 _].

Examples

```
IntroduceVariables[{y}, plus[x, 0]]
```

```
plus[x, 0]
```

```
IntroduceVariables[{x}, plus[x, 0]]
```

```
plus[x_, 0]
```

```
IntroduceVariables[{x}, plus[x, y⁺]]
```

```
plus[x_, y⁺]
```

```
IntroduceVariables[{x, y, z}, plus[x, y⁺]]
```

```
plus[x_, y_⁺]
```

```
IntroduceVariables[{x, y}, plus[x, a, y⁺]]
```

```
plus[x_, a, y_⁺]
```

```
IntroduceVariables[{x, a, y}, plus[x, a, y⁺]]
```

```
plus[x_, a_, y_⁺]
```

```
IntroduceVariables[{x, plus, y}, plus[x, a, y⁺]]
```

```
plus_[x_, a, y_⁺]
```

```
IntroduceVariables[{plus, y, x}, plus[x, a, y⁺]]
```

```
plus_[x_, a, y_⁺]
```

Prove By Simplification

Calling the Prover by ‘Prove’

Definition

```
Clear[Prove]

Prove[expression_] :=

 Module[{reasonTree, result},

  reasonTree = ProveBySimplification[expression, {}];
  result = Result[reasonTree];
  RT[Prove,
   expression,
   $Theory,
   result,
   {reasonTree}]]
```

‘ProveBySimplification’ for Equalities

Background

In the goal, here, equalities, have the form ‘leftExpression ≡ rightExpression’.

```
a ≡ b // FullForm
```

```
Congruent[a, b]
```

In the knowledge, here, equalities, are represented as Mathematica rewrite rules.

Definition

```
Clear[ProveBySimplification]

ProveBySimplification[leftExpression_ ≡ rightExpression_, {equalities___}] :=

 Module[{reasonTree, result, r, lhs, rhs},

  reasonTree = SimplifyByRewriting[leftExpression ≡ rightExpression, {equalities}];
  r = Result[reasonTree]; lhs = r[[1]]; rhs = r[[2]];
  result = If[lhs === rhs, Proved, Unproved];
  (* Result[reasonTree] has the form lhs ≡ rhs,
  where lhs and rhs are simplified w.r.t. the equalities.*)
  RT[ProveBySimplification,
   leftExpression ≡ rightExpression,
   {equalities},
   result,
   {reasonTree}]]
```

Examples with $Theory=Axiomsℕplusforplus'

```
$Theory = Axiomsℕplusforplus = {plus[x_, 0] → x, plus[x_, y_⁺] → plus[x, y]⁺}
```

```
{plus[x_, 0] → x, plus[x_, y_⁺] → plus[x, y]⁺}
```

```
SimplifyByRewriting[plus[0, 0], {}] // NestedCellsForm
```

```
NotebookObject[ Untitled-20 ]
```

```
SimplifyByRewriting[plus[0⁺⁺⁺⁺, 0⁺⁺⁺], {}] // NestedCellsForm
```

```
NotebookObject[ Untitled-10 ]
```

```
SimplifyByRewriting[plus[0⁺⁺⁺⁺, 0⁺] ≡ 0⁺⁺⁺⁺, {}] // NestedCellsForm
```

```
NotebookObject[ Untitled-21 ]
```

```
ProveBySimplification[plus[0⁺⁺⁺⁺, 0⁺] ≡ 0⁺⁺⁺⁺, {}] // NestedCellsForm
```

```
NotebookObject[ Untitled-22 ]
```

```
Prove[plus[0⁺⁺⁺⁺, 0⁺] ≡ 0⁺⁺⁺⁺⁺] // NestedCellsForm
```

```
NotebookObject[ Untitled-23 ]
```

```
Prove[plus[0⁺⁺⁺⁺, 0⁺] ≡ plus[0⁺, 0⁺⁺⁺⁺]] // NestedCellsForm
```

```
NotebookObject[ Untitled-24 ]
```

```
Prove[plus[0⁺⁺⁺⁺⁺, 0⁺⁺⁺] ≡ plus[0⁺, plus[0⁺⁺⁺, 0⁺⁺⁺⁺]]] // NestedCellsForm
```

NotebookObject[Untitled–25]

```
Prove[plus[n⁺, 0⁺] ≡ n⁺⁺] // NestedCellsForm
```

NotebookObject[Untitled–26]

Why does this proof show that, in fact, $\forall_n$ plus[n⁺, 0⁺] ≡ n⁺⁺.

The next equality cannot be proved at this stage. We would have to prove left induction for plus first!

```
Prove[plus[m⁺, n⁺] ≡ plus[m, n]⁺⁺] // NestedCellsForm
```

NotebookObject[Untitled–93]

ProveBySimplificationOrInduction

Calling the Prover by 'Prove'

Definition

```
Clear[Prove]

Prove[expression_] :=

 Module[{reasonTree, result},

  reasonTree = ProveBySimplificationOrInduction[expression, {}];
  result = Result[reasonTree];
  RT[Prove,
   expression,
   $Theory,
   result,
   {reasonTree}]]
```

'ProveBySimplificationOrInduction' (for Equalities)

Background

In the goal, here, equalities, have the form 'forAll[{n1, n2,...}, leftExpression ≡ rightExpression'.

Attention: Induction variables n1, n2, ... must be different from each other and must be fresh!

In the knowledge, here, equalities, are represented as Mathematica rewrite rules.

Definition

```
Clear[ProveBySimplificationOrInduction]
```

```
ProveBySimplificationOrInduction[
  forAll[{n1_, n__}, expression_], {equalities___}] :=
 Module[{reasonTree0, reasonTree1, reasonTree2, result},

  reasonTree0 = ProveBySimplification[expression, {equalities}];
  result = Result[reasonTree0];
  If[result === Proved,
   Return[
    RT[ProveBySimplificationOrInduction,
     forAll[{n1, n}, expression], {equalities}, Proved, {reasonTree0}]]];

  reasonTree1 = ProveBySimplificationOrInduction[
    forAll[{n}, expression /. (n1 → 0)], {equalities}];
  result = Result[reasonTree1];
  If[result === Unproved,
   Return[
    RT[ProveBySimplificationOrInduction,
     forAll[{n1, n}, expression], {equalities}, Unproved,
     {reasonTree1}]]];

  reasonTree2 =
   ProveBySimplificationOrInduction[forAll[{n}, expression /. (n1 → n1^+)],
    {equalities,
     RuleFromEquality[forAll[{n}, expression /. (n1 → 0)]],
     RuleFromEquality[forAll[{n}, expression]]}];
  result = Result[reasonTree2];
  RT[ProveBySimplificationOrInduction,
   forAll[{n1, n}, expression], {equalities}, result,
   {reasonTree1, reasonTree2}]]

ProveBySimplificationOrInduction[forAll[{n1_}, expression_], {equalities___}] :=

 Module[{reasonTree0, reasonTree1, reasonTree2, result},

  reasonTree0 = ProveBySimplification[expression, {equalities}];
  result = Result[reasonTree0];
  If[result === Proved,
   Return[
    RT[ProveBySimplificationOrInduction,
     forAll[{n1}, expression], {equalities}, Proved, {reasonTree0}]]];

  reasonTree1 = ProveBySimplification[expression /. (n1 → 0), {equalities}];
  result = Result[reasonTree1];
  If[result === Unproved,
   Return[
    RT[ProveBySimplificationOrInduction,
     forAll[{n1}, expression], {equalities}, Unproved,
     {reasonTree1}]]];

  reasonTree2 = ProveBySimplification[expression /. (n1 → n1^+),
    {equalities,
     RuleFromEquality[expression /. (n1 → 0)], RuleFromEquality[expression]}];
  result = Result[reasonTree2];
  RT[ProveBySimplificationOrInduction,
```

```
    forAll[{n1}, expression], {equalities}, result,
    {reasonTree1, reasonTree2}]]

ProveBySimplificationOrInduction[expression_, {equalities___}] :=
 ProveBySimplification[expression, {equalities}]
```

Examples

```
$Theory = Axioms`plus = {plus[x_, 0] → x, plus[x_, y_⁺] → plus[x, y]⁺}
```

```
{plus[x_, 0] → x, plus[x_, y_⁺] → plus[x, y]⁺}
```

```
Prove[forAll[{y}, plus[0, y] ≡ y]]
```

```
RT[Prove, forAll[{y}, plus[0, y] ≡ y], {plus[x_, 0] → x, plus[x_, y_⁺] → plus[x, y]⁺},
 Proved, {RT[ProveBySimplificationOrInduction, forAll[{y}, plus[0, y] ≡ y],
   {}, Proved, {RT[ProveBySimplification, plus[0, 0] ≡ 0, {}, Proved,
     {RT[SimplifyByRewriting, plus[0, 0] ≡ 0, {}, 0 ≡ 0]}], RT[ProveBySimplification,
     plus[0, y⁺] ≡ y⁺, {plus[0, 0] :↦ 0, plus[0, y] :↦ y}, Proved, {RT[SimplifyByRewriting,
       plus[0, y⁺] ≡ y⁺, {plus[0, 0] :↦ 0, plus[0, y] :↦ y}, y⁺ ≡ y⁺]}]}]}]]
```

```
Prove[forAll[{n}, plus[0, n] ≡ n]] // NestedCellsForm
```

```
NotebookObject[ Untitled–35 ]
```

Proved.

This seems dangerous but it works. It is even “normal” in every day math!

```
Prove[forAll[{x}, plus[0, x] ≡ x]] // NestedCellsForm
```

```
Part: Part 2 of Result[Unproved] does not exist.
```

```
Part: Part 2 of Result[Unproved] does not exist.
```

```
NotebookObject[ Untitled–32 ]
```

```
Prove[forAll[{x, y}, plus[x⁺, y] ≡ plus[x, y]⁺]] // NestedCellsForm
```

```
NotebookObject[ Untitled–29 ]
```

Proved.