Natural Proofs in Elementary Analysis by S-Decomposition

Tudor Jebelean
RISC-LINZ

Research Institute for Symbolic Computation
Johannes Kepler University, A-4040 Linz, Austria (Europe)
Tel: +43 (732) 2468-9946 Fax: +43 (732) 2468-9930
e-mail: `jebelean@risc.uni-linz.ac.at`

Abstract

The proof-situation is decomposed top-down, according to the structure of the definition of the main notion involved. The order of the processing steps alternates: first the assumptions and then the goal when the main quantifier is existential, and the other way around when it is universal. Finally the proof-situation is reduced to simpler lemmata, which can be solved independently either by matching against a database of auxiliary knowledge, or by other provers, possibly domain-specific (based on algebraic techniques). The method is implemented in the *Theorema* system (based on *Mathematica*) and produces very natural proofs in elementary analysis, which are particularly difficult for other methods because of the complex definitions with alternating quantifiers.

Introduction

Proving in natural style gained increased interest recently, on one hand because in many practical applications the [non-specialist] users need to see an understandable trace of the proof, on the other hand because such a trace makes interaction [for such users] possible and thus allows to approach non-trivial problems. The present work has been done in the frame of the *Theorema* project, whose aim is to provide a computer-aided environment for the working mathematician, including the possibility of proving in natural style [5].

The presented technique is motivated by the proving style of humans. Instead of doing inferences on a large set of formulae, a mathematician will try to decompose the goal into simpler components, finally establishing a set of simpler lemmata which are needed.

Consider the example from the appendix, lines (5), (6), and (7). A natural proof of (5) based on the assumption (6) and (7) may look like this:

- by the assumptions we can take a_1 and a_2 (such that the respective instances hold), then we prove the goal for $a = a_1 + a_2$;
- for proving the new goal we instantiate it for ϵ_0, then we instantiate the new assumptions for $\epsilon = \epsilon_0/2$; and we prove additionally $\epsilon_0 > 0 \Rightarrow \epsilon_0/2 > 0$;
- by the new assumptions we can take N_1, N_2, then we prove the goal for $N = max[N_1, N_2]$;
- etc. ...

The proof always proceeds from the assumptions towards the goal for the existential quantifiers, and the other way around for universal quantifiers. If, over the three formulae, we draw a line which follows the succession of the proof steps, this line will be S-shaped like a snake.

S-decomposition is a precise formalization of this method, which, although incomplete, generates very natural proofs for many interesting proof problems, notably for properties in elementary analysis which are based on definitions with alternating quantifiers.

S-decomposition separates the phases of the proof which should be treated by different type of methods: while the decomposition is based on logic rules only, the resulting lemmata can be attacked by domain-specific provers, possibly using techniques from Computer Algebra.

The importance of finding specific methods for the formulae with alternating quantifiers (notably for elementary analysis) has been noted by Buchberger [8], who also suggested the (off-line) recursive decomposition of them at the level of each quantifier in order to simplify the proving process. The same author introduced the PCS (Proving–Solving–Computing) method [5], which is also suitable for proofs in elementary analysis, because it combines logic and algebraic processing. This type of proofs have been also investigated in [11, 10], but based on different principles. The special use of (simpler) definitions in proofs has been formalized in [7].

1 S-decomposition

The method of top-down decomposition applies to proof-situations of the form:

$$A_1, \ldots, A_n \ \vdash \ A_0$$

where $A_1, \ldots, A_n, A_0$, have the same structure. Typically such a proof situation occurs when A_k are obtained by applying the same definition to different instances of a predicate, see formulae (2), (3), (4) in the appendix.

The method consists in reducing such a proof-situation to one or more simpler proof situations, recursively, according to the main logical operator of A_k. Since in typical proof-situations the structure of the formulae has one of the forms:

$$\underset{x}{\forall}\,(P[x] \Rightarrow Q[x]), \quad \underset{x}{\exists}\,(P[x] \wedge Q[x]),$$

the most useful decomposition rules are the following (for the case $n = 2$, since the generalization is obvious):

$$[\wedge] \quad P_1 \wedge Q_1, P_2 \wedge Q_2 \ \vdash \ P_0 \wedge Q_0 \quad \longmapsto \quad \begin{cases} P_1 \wedge P_2 \ \vdash \ P_0 \\ Q_1 \wedge Q_2 \ \vdash \ Q_0 \end{cases}$$

$$[\Rightarrow] \quad P_1 \Rightarrow Q_1, P_2 \Rightarrow Q_2 \ \vdash \ P_0 \Rightarrow Q_0 \quad \longmapsto \quad \begin{cases} P_0 \ \vdash \ P_1 \wedge P_2 \\ Q_1 \wedge Q_2 \ \vdash \ Q_0 \end{cases}$$

$$[\forall] \quad \underset{x}{\forall}\, P_1[x], \underset{x}{\forall}\, P_2[x] \ \vdash \ \underset{x}{\forall}\, P_0[x] \quad \longmapsto \quad P_1[x^*], P_2[x^*] \ \vdash \ P_0[x_0]$$

$$[\exists] \quad \underset{x}{\exists}\, P_1[x], \underset{x}{\exists}\, P_2[x] \ \vdash \ \underset{x}{\exists}\, P_0[x] \quad \longmapsto \quad P_1[x_1], P_2[x_2] \ \vdash \ P_0[x^*]$$

In the rules for $[\forall]$ and $[\exists]$, x_k denote *Skolem constants*, and x^* denote *meta-variables.* Skolem constants are arbitrary but fixed constants. Meta-variables denote unknown constant (ground) terms whose value will be determined later during the proof. (These terms will be in fact functions of the Skolem constants introduced by the respective rule.) Both kinds of symbols have to be *new* to the proof. This is insured in the current implementation by assigning increasing indices to the symbols. Due to the structure of S-decomposition, assigning new names to symbols, which are also intuitively relevant, is relatively easy.

A certain care has to be taken when using both Skolem constants and meta-variables, in order to avoid inconsistency. The literature is rich in such "care taking" schemes (see e. g. a survey [2]), we choose the use of a certain *precedence* relation $\succ$ (described in detail in [9]):

- when a new Skolem constant s is introduced in a formula F, then for all the Skolem constants and meta-variables f occurring in F we set $s \succ f$;
- when a substitution (solution-term) T is found for a metavariable m, we set $m \succ t$ for all Skolem constants and meta-variables t occurring in T, and we accept the solution only if $\succ$ has no cycles.

Intuitively, $s \succ x$ means "s should be created after x", and the absence of cycles in $\succ$ means that we can arrange the steps of the proof such that (after replacing the meta-variables by their solutions), every occurrence of a Skolem constant is after the proof step which creates it. Efficient proving methods based on meta-variables will usually need a rearrangement of the steps in order to fulfill the condition above (see [9]), however a[nother] nice feature of the S-decomposition is that it creates the Skolem constants (and the meta-variables) in the right order.

An alternative "care-taking" method is the use of Skolem functions instead of Skolem constants. In this case the solutions which would yield cyclic $\succ$ are automatically rejected by the unifier (because of self-dependence of the meta-variables). The disadvantages of this method include: manipulation of more complicated expressions, increase of workload for the unifier, change of the original signature of the problem. However, Skolemization explicitizes certain functions which have an interesting algorithmic and tutorial value (as noted e. g. in [5]). For instance, in appended example, the final solution $a_0^* = a_1 + a_2$ would be Skolemized to $lim[f_1 + f_2] = lim[f_1] + lim[f_2]$ (if one used the symbol lim instead of a). Obviously S-decomposition can be also applied in conjunction with explicit Skolem functions.

The rule $[\forall]$ can be also formulated in a more complete form:

$$P_1[x_1^*], P_2[x_2^*] \ \vdash \ P_0[x_0]$$

(a meta-variable is introduced for each of the premises), however in all the examples we tested one single meta-variable was sufficient. This may be important for the complexity of the work involved in proving the lemmata produced in the final steps of the proof, since some domain-specific provers (e. g. quantifier elimination, Groebner Bases) are very sensitive to the number of variables.

Note that the rules do not yield an equivalent [conjunction of] proof-situation[s]: if all the new proof-situations are tautologies, then the original formula is also a tautology, but not the other way around. This means that if proving the new formulae fails, the original formula may still be a tautology. Thus the method is not complete, however it is still very useful in many practical situations, as it will be seen from the examples. For instance, the method will work for proving the continuity of various arithmetic combinations of functions, but will not work for the continuity of function composition.

For the sake of completeness we will now formulate similar rules for the other logical connectives, however we do not know how useful they are, since we did not test them on practical examples.

The only reasonable way to formulate a rule for $\sim$ appears to be:

$$[\sim] \qquad \sim P_1, \sim P_2 \ \vdash \ \sim P_0 \quad \longmapsto \quad P_0 \ \vdash \ P_1 \vee P_2$$

However, this decomposition does not produce a proof-situation of the type $A_1, \ldots, A_n \ \vdash \ A_0$, thus it cannot be used in the recursive decomposition. Therefore, one should move $\sim$ inside the formulae (using de Morgan rules) and apply the rule above only when P_k are literals.

If we transform the disjunctions into implications, then apply the rule $[\Rightarrow]$ and contraposition, then we obtain:

$$[\vee] \quad P_1 \vee Q_1, P_2 \vee Q_2 \ \vdash \ P_0 \vee Q_0 \quad \longmapsto \quad \begin{cases} P_1 \vee P_2 \ \vdash \ P_0 \\ Q_1 \wedge Q_2 \ \vdash \ Q_0 \end{cases}$$

If we transform the equivalences into conjunctions of implications and then apply the rules for $[\wedge]$ and $[\Rightarrow]$, then we obtain:

$$[\Leftrightarrow] \quad P_1 \Leftrightarrow Q_1, P_2 \Leftrightarrow Q_2 \ \vdash \ P_0 \Leftrightarrow Q_0 \quad \longmapsto \quad \begin{cases} P_1 \wedge P_2 \Leftrightarrow P_0 \\ Q_1 \wedge Q_2 \Leftrightarrow Q_0 \end{cases}$$

This again does not yield a form on which the transformation rules can be applied recursively, but can be applied when P_k, Q_k are literals. However, by further logical manipulations, one can obtain the following rule:

$$[\Leftrightarrow] \quad P_1 \Leftrightarrow Q_1, P_2 \Leftrightarrow Q_2 \ \vdash \ P_0 \Leftrightarrow Q_0 \quad \longmapsto \quad \begin{cases} P_1 \wedge P_2 \ \vdash \ P_0 \\ \sim P_1 \wedge \sim P_2 \ \vdash \sim P_0 \\ Q_1 \wedge Q_2 \ \vdash \ Q_0 \\ \sim Q_1 \wedge \sim Q_2 \ \vdash \ Q_0 \end{cases}$$

2 Proof Organization

Initial step. At the beginning the prover identifies the *main assumptions* and the *main definition* by inspecting the goal and the definitions (characterized by $:\Leftrightarrow$). In the appendix, the main assumptions are (3) and (4), while the main definition is (conv:). Then the main definition is applied to the goal and to the main assumptions, yielding a proof-situation of type $A_1, \ldots, A_n \ \vdash \ A_0$. In the appendix, these are formulae (6), (7), and (5).

S-Decomposition. Now the proof proceeds by top-down decomposition according to the rules described in the previous section. For conciseness, in the current implementation the rules [$\forall$] with [$\Rightarrow$], as well as [$\exists$] with [$\wedge$] are applied together when this is possible - see the proof in the appendix. During the decomposition process, only the main assumptions and the main goal are used, thus the complexity of the proof does not depend on the number of auxiliary assumptions.

Typically, at each decomposition step some "side" lemma is produced by regrouping proof situations of the type $A_1, \ldots, A_n a \ \vdash \ A_0$ (where A_k are literals) into goals of type: $A_1 \wedge \ldots \wedge A_n \Rightarrow A_0$. The last lemma is produced when the main assumptions and the main goal are reduced to (similar) literals, usually this is the most complicated one. Thus, after the decomposition phase the leafs of the proof-tree consist in such lemmata. (In the appendix, these are (18), (29), and (30).)

Final Solutions. After the decomposition phase, each leave of the proof-tree consists in a lemma (to prove). A simple preprocessing step is the application of some equality-based rewriting, which expands subterms using definitions by equality (in the appendix this happens at step (35)).

Then the subproof may proceed in various ways – two are described in the next section – but we do not see this as the task of the main prover: the subproof should be a separate task, which is probably done in a different way (e. g. domain specific). This corresponds to the typical proofs in (human written) mathematical texts, in which the lemmata are only *used*, but proved elsewhere.

For the main prover it is only important that after each such "sub-proof" a solution to the meta-variables involved in the lemma is obtained. The main prover will send these substitutions to all the other unsolved nodes and will update the lemmata accordingly. (In the appendix, the solution obtained at step (29) is used at step (32).) This insures the consistency of the proof (each meta-variable has to have the same value for the whole proof), but also introduces a certain amount of nondeterminism (and thus backtracking) in the method: it is important in which order the lemmata are attacked, because wrong solutions will make proving of other lemmata impossible. For instance, in the appendix, lemma (18) has the simple (but wrong) solution $\epsilon_k^* = \epsilon_0$. In the current implementation we avoid backtracking for such proofs by using a "simplicity" criterion: the subproof of a "simple" lemma is delayed until (some of) the meta-variables are solved by other branches. In the appendix, one can see that the solution $\epsilon_k^* = \epsilon_0/2$ is used, which, although it appears later in the listing of the proof, is produced earlier in the proof process. However, a backtracking (and book-keeping of solutions) mechanism is still necessary, this is implemented in the system and is described in detail in [9].

3 Proving the Lemmata

We implemented two methods for solving the resulting subproofs: the *database search* and the *domain specific proofs*.

Database search. This is the method used in the appendix and consists in matching the resulting lemma against (universal) formulae listed as auxiliary knowledge. For simplicity of this presentation the method is implemented in the main prover itself and the auxiliary formulae are input to the initial proof-situation (however the prover uses them only in the final phase). We think that this proving style is already useful for the working mathematician: a user can try the proof first without any auxiliary knowledge, and then by inspecting the unsolved lemmata he can complete the knowledge until the proof works. (This is exactly what we do when we prove by hand!). In the appendix, we left intentionally the lemma (36) without correspondence in the auxiliary assumptions, thus it cannot be proven. One can see that the proof of the other branches is not affected, and also that the shape of the unsolved lemma suggests immediately the necessary auxiliary assumption.

Alternatively, one can also imagine that the prover calls an external engine which inspects a database of formulae (like e. g. Mizar [12], Omega and OM-Doc [1], or even the TPTP library).

In fact, the auxiliary formulae are of the type which is obtained by "theory exploration" (see [4]), which is the natural approach when constructing mathematical theories: we introduce new symbols (predicates, functions) one by one, and at each new symbol we investigate the theorems (lemmata) which relate it to the older symbols.

Domain specific proofs. A more sophisticated method consists in attacking the lemma with another prover (possibly using specific techniques for the type at hand). For this, the lemma has to be first transformed in a "standard" form, which contains all the information from the (so far) proof. Namely, we transform the Skolem constants and the meta-variables into quantified variables: the order and the type of the quantifiers encode the relationship between the symbols which has been detected during the proof. The order of the quantifiers is the order in which the corresponding symbols have been introduced during the proof (which is encoded as the relation $\succ$). The constants of the initial proof-situation which are not specific to the domain (for instance f_1, f_2, but not $+$) become universally quantified as the *first* ones. Skolem constants become universal variables, while meta-variables become existential variables. In the example from the appendix, the following transformations will be applied:

- (36) becomes
$$\underset{\epsilon_0}{\forall} \left(\epsilon_0 > 0 \Rightarrow \frac{\epsilon_0}{2} > 0\right)$$
- (29) becomes
$$\underset{N_1,N_2}{\forall} \; \underset{N_0^*}{\exists} \; \underset{n_0}{\forall} \; \underset{n_1^*}{\exists} \left(n_0 \geq N_0^* \Rightarrow n_1^* \geq N_1 \wedge n_1^* \geq N_2\right)$$
- (35) becomes
$$\underset{f_1,f_2}{\forall} \; \underset{a_1,a_2}{\forall} \; \underset{a_0^*}{\exists} \; \underset{\epsilon_0}{\forall} \; \underset{\epsilon_1^*}{\exists} \; \underset{n_0}{\forall} \left(|f_1[n_0] - a_1| < \epsilon_1^* \wedge |f_2[n_0] - a_2| < \epsilon_1^* \Rightarrow |(f_1[n_0] + f_2[n_0]) - a_0^*| < \epsilon_0\right)$$

Formula simplification. The lemmata above can be directly passed to a specialized prover over the reals, for instance quantifier elimination by the CAD method [6], or Groebner Bases [3], however some simple transformations can be applied in advance in order to reduce the size of the problem (and thus to increase the probability of success).

We illustrate the required transformations by an example – formula derived from (35), mostly in order to emphasize the intimate relationship between proving and algebraic manipulations in this case.

Step 1. Replacement of common universal sub-terms. We substitute: $\{x_1 \to f_1[n_0],\ x_2 \to f_2[n_0]\}$ and we *merge right* the universal quantifiers:

$$\underset{a_1,a_2}{\forall}\ \underset{a_0^*}{\exists}\ \underset{\epsilon_0}{\forall}\ \underset{\epsilon_1^*}{\exists}\ \underset{x_1,x_2}{\forall} \left(|x_1 - a_1| < \epsilon_1^* \wedge |x_2 - a_2| < \epsilon_1^* \Rightarrow |(x_1 + x_2) - a_0^*| < \epsilon_0\right)$$

Step 2. Emphasize "known" subterms. We substitute: $\{u_1 \leftarrow x_1 - a_1,\ u_2 \leftarrow x_2 - a_2,\}$ and obtain:

$$\underset{a_1,a_2}{\forall}\ \underset{a_0^*}{\exists}\ \underset{\epsilon_0}{\forall}\ \underset{\epsilon_1^*}{\exists}\ \underset{u_1,u_2}{\forall} \left(|u_1| < \epsilon_1^* \wedge |u_2| < \epsilon_1^* \Rightarrow |((u_1 + a_1) + (u_2 + a_2)) - a_0^*| < \epsilon_0\right)$$

Step 3. Algebraic simplifications. This phase is domain specific (to reals) and is performed by calls to the underlying computer algebra system. The goal is to simplify the complex formula in the conclusion in order to be able to use the premises:

$$|((u_1 + a_1) + (u_2 + a_2)) - a_0^*| = |a_1 + a_2 - a_0^* + u_1 + u_2|$$

Then we eliminate the metavariable by equating to zero the subexpression containing a_0^* and the subterms containing only the symbols on which the expression of a_0^* should depend (from the order of the quantifiers we know that these are a_1 and a_2):

$$a_1 + a_2 - a_0^* = 0$$

gives the solution

$$a_0^* \longrightarrow a_1 + a_2$$

and the expression can be further simplified:

$$|u_1 + u_2|$$

Step 4. Use special properties. These are built-in to the domain-specific prover and are encoded as rewrite rules related to absolute value, ordering, addition, multiplication, etc.

$$|u_1 + u_2| \leq |u_1| + |u_2| < \epsilon_1^* + \epsilon_1^* = 2 * \epsilon_1^*$$

(The last one is using the premises.)
Step 5. Make equation and solve meta-variable.

$$2 * \epsilon_1^* = \epsilon_0, \quad \epsilon_1^* = \epsilon_0/2$$

The reader may check that using the same principles (and the additional assumption $\epsilon_1^* \leq 1$ in order to linearize the equation from step 5), one can obtain for the case of multiplication:

$$\epsilon_1^* = \epsilon_0/(1 + |a_1| + |a_2|)$$

4 Experiments and Conclusion

We implemented this method in the *Theorema* system, which is based on *Mathematica* – thus the prover has easy access to various routines for algebraic manipulations.

The appendix presents an automatically produced proof for the convergence of real functions over natural numbers. (For keeping the proof short we did not include the info on types, but this is easy to handle in *Theorema*, both at explicit and implicit level). The text is transformed into TeX from the *Mathematica* notebook of the proof – and unfortunately some of the details are lost, notably the colors of different types of formulae (goal, assumptions) and the right-hand side brackets marking the cells, which in the *Theorema* proofs show the structure of the proof-tree.

Similar proofs have been produced for this kind of problems based on definitions with alternating quantifiers: continuity, uniform continuity, etc. and for various operations of functions: sum, product, inverse.

Currently we are implementing the simplification of formulae presented in the last section, as well as an interface to quantifier elimination. Further research include testing with a larger class of formulae and a deeper theoretical investigation possibly leading to precise criteria for the applicability of this method.

References

[1] C. Benzmueller, L. Cheikhrouhou, D. Fehrer, A. Fiedler, X. Huang, M. Kerber, M. Kohlhase, K. Konrad, E. Mellis, A. Meier, W. Schaarschmidt, J. Siekman, and V. Sorge. ΩMEGA: Towards a Mathematical Assistant. In William McCune, editor, *Proceedings of the 14th Conference on Automated Deduction*, number 1249 in LNAI, pages 252–255, Townsville, Australia, 1997. Springer-Verlag. See also http://www.ags.uni-sb.de/projects/deduktion/.

[2] W. Bibel and P. Schmitt, editors. *Automated Deduction—a Bases for Applications*, volume 1–3 of *Applied Logic Series*. Kluver Academic Publishers, 1998.

[3] B. Buchberger. Gröbner-Bases: An Algorithmic Method in Polynomial Ideal Theory. In N.K. Bose, editor, *Multidimensional Systems Theory*, pages 184–232. Reidel Publishing Company, Dodrecht, 1985.

[4] B. Buchberger. Theory exploration versus theorem proving. In A. Armando and T. Jebelean, editors, *CALCULEMUS '99 Workshop*, volume 23 of *Electronic Notes in Theoretical Computer Science*, Trento, Italy, 1999. Elsevier. Extended version published as RISC report 99-46, 1999.

[5] B. Buchberger, C. Dupré, T. Jebelean, F. Kriftner, K. Nakagawa, D. Văsaru, and W. Windsteiger. The theorema project: A progress report. In M. Kerber and M. Kohlhase, editors, *Calculemus 2000: Integration of Symbolic Computation and Mechanized Reasoning*. A. K. Peters, Natick, Massatchussets, 2000.

[6] G. E. Collins. Quantifier Elimination for Real Closed Fields by Cylindrical Algebraic Decomposition. In *Second GI Conference on Authomata Theory and Formal Languages*, volume 33 of *LNCS*, pages 134–183. Springer-Verlag, Berlin, 1975.

[7] A. Degtyarev and A. Voronkov. Stratified Resolution. In D. McAllester, editor, *Proc.of 17th International Conference on Automated Deduction (CADE'17)*, pages 365–384. Springer LNCS 1831, 2000.

[8] B. Buchberger et al. A Survey of the Theorema Project. In W. Kuechlin, editor, *ISSAC'97 (International Symposium on Symbolic and Algebraic Computation, Maui, Hawaii, July 21-23, 1997)*. ACM Press, 1997.

[9] B. Konev and T. Jebelean. Using meta-variables for natural deduction in theorema. In M. Kerber and M. Kohlhase, editors, *Calculemus 2000: Integration of Symbolic Computation and Mechanized Reasoning*. A. K. Peters, Natick, Massatchussets, 2000.

[10] E. Melis. Progress in proof planning: Planning limit theorems automatically. Technical report, Univ. Saarlandes, FB Informatik, Saarbrücken, Germany, 1997.

[11] E. Melis. The limit domain. In R. Simmons et al., editor, *Procs. AIPS'98*, 1998.

[12] A. Trybulec and H. Blair. Computer Assisted Reasoning with MIZAR. In Aravind Joshi, editor, *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, pages 26–28, Los Angeles, CA, 1985. Morgan Kaufmann. See also http://mizar.uw.bialystok.pl/.

Appendix: Automatically generated proof

.

Prove:

(conv of sum) $(conv[f_1] \wedge conv[f_2] \Rightarrow conv[f_1 \oplus f_2])$,

under the assumptions:

(conv:) $\underset{f}{\forall} \left(conv[f] :\Leftrightarrow \underset{a}{\exists} \underset{\epsilon}{\forall} \left(\epsilon > 0 \Rightarrow \underset{N}{\exists} \underset{n}{\forall} (n \geq N \Rightarrow |f[n] - a| < \epsilon) \right) \right)$,

(sum of functions:) $\underset{f,g,x}{\forall} ((f \oplus g)[x] := f[x] + g[x])$,

(distance of sum) $\underset{x,y,a,b,\epsilon}{\forall} \left(|x - a| < \frac{\epsilon}{2} \wedge |y - b| < \frac{\epsilon}{2} \Rightarrow |(x + y) - (a + b)| < \epsilon \right)$,

(max:) $\underset{k,i,j}{\forall} (k \geq \max[i, j] \Rightarrow k \geq i \wedge k \geq j)$,

For proving (conv of sum), we assume:

(3) $conv[f_1]$,

(4) $conv[f_2]$,

and we prove:

(2) $conv[f_1 \oplus f_2]$.

Using the definition (conv:), we expand the goal (2) and the assumptions (3) and (4) into:

(5) $\underset{a}{\exists} \underset{\epsilon}{\forall} \left(\epsilon > 0 \Rightarrow \underset{N}{\exists} \underset{n}{\forall} (n \geq N \Rightarrow |(f_1 \oplus f_2)[n] - a| < \epsilon) \right)$

(6) $\underset{a}{\exists} \underset{\epsilon}{\forall} \left(\epsilon > 0 \Rightarrow \underset{N}{\exists} \underset{n}{\forall} (n \geq N \Rightarrow |f_1[n] - a| < \epsilon) \right)$

(7) $\underset{a}{\exists} \underset{\epsilon}{\forall} \left(\epsilon > 0 \Rightarrow \underset{N}{\exists} \underset{n}{\forall} (n \geq N \Rightarrow |f_2[n] - a| < \epsilon) \right)$

By the assumptions (6) and (7), we can take appropriate constants (a_1, a_2) such that:

(9) $\underset{\epsilon}{\forall} \left(\epsilon > 0 \Rightarrow \underset{N}{\exists} \underset{n}{\forall} (n \geq N \Rightarrow |f_1[n] - a_1| < \epsilon) \right)$

(10) $\underset{\epsilon}{\forall} \left(\epsilon > 0 \Rightarrow \underset{N}{\exists} \underset{n}{\forall} (n \geq N \Rightarrow |f_2[n] - a_2| < \epsilon) \right)$

For proving (5), we have to find an appropriate value a_0^* such that:

(8) $\underset{\epsilon}{\forall} \left(\epsilon > 0 \Rightarrow \underset{N}{\exists} \underset{n}{\forall} \left(n \geq N \Rightarrow |(f_1 \oplus f_2)[n] - a_0^*| < \epsilon \right) \right)$.

For proving (8), we replace the universal variables by arbitrary constants (ϵ_0), we assume:

(11) $\epsilon_0 > 0$,

and we prove:

(12) $\underset{u}{\exists} \underset{u}{\forall} \left(n \geq N \Rightarrow |(f_1 \oplus f_2)[n] - a_0^*| < \epsilon_0 \right)$.

For using the assumptions (9) and (10) we have to find an appropriate value ϵ_1^*, and we assume:

(13) $\epsilon_1^* > 0 \Rightarrow \underset{N}{\exists}\,\underset{n}{\forall}\left(n \geq N \Rightarrow |f_1[n] - a_1| < \epsilon_1^*\right)$

(14) $\epsilon_1^* > 0 \Rightarrow \underset{N}{\exists}\,\underset{n}{\forall}\left(n \geq N \Rightarrow |f_2[n] - a_2| < \epsilon_1^*\right)$

We prove first:

(15) $\epsilon_1^* > 0$.

For proving (15), by (11), it suffices to prove

(18) $\epsilon_0 > 0 \Rightarrow \epsilon_1^* > 0$.

Using the substitution from step (35), the goal (18) becomes:

(36) $\epsilon_0 > 0 \Rightarrow \frac{\epsilon_0}{2} > 0$.

Pending proof of (36).

Now we assume (15) and we continue with the proof of (12).

From (15), by the assumptions (13) and (14) we obtain:

(16) $\underset{N}{\exists}\,\underset{n}{\forall}\left(n \geq N \Rightarrow |f_1[n] - a_1| < \epsilon_1^*\right)$

(17) $\underset{N}{\exists}\,\underset{n}{\forall}\left(n \geq N \Rightarrow |f_2[n] - a_2| < \epsilon_1^*\right)$

By the assumptions (16) and (17), we can take appropriate constants (N_1, N_2) such that:

(20) $\underset{n}{\forall}\left(n \geq N_1 \Rightarrow |f_1[n] - a_1| < \epsilon_1^*\right)$

(21) $\underset{n}{\forall}\left(n \geq N_2 \Rightarrow |f_2[n] - a_2| < \epsilon_1^*\right)$

For proving (12), we have to find an appropriate value N_0^* such that:

(19) $\underset{u}{\forall}\left(n \geq N_0^* \Rightarrow |(f_1 \oplus f_2)[n] - a_0^*| < \epsilon_0\right)$.

For proving (19), we replace the universal variables by arbitrary constants (n_0), we assume:

(22) $n_0 \geq N_0^*$,

and we prove:

(23) $|(f_1 \oplus f_2)[n_0] - a_0^*| < \epsilon_0$.

For using the assumptions (20) and (21) we have to find an appropriate value n_1^*, and we assume:

(24) $n_1^* \geq N_1 \Rightarrow |f_1\left[n_1^*\right] - a_1| < \epsilon_1^*$

(25) $n_1^* \geq N_2 \Rightarrow |f_2\left[n_1^*\right] - a_2| < \epsilon_1^*$

We prove first:

(26) $n_1^* \geq N_1 \wedge n_1^* \geq N_2$.

For proving (26), by (22), it suffices to prove

(29) $n_0 \geq N_0^* \Rightarrow n_1^* \geq N_1 \wedge n_1^* \geq N_2$.

Formula (29) is proved because it becomes an instance of (max:) by taking: $\{n_1^* \to n_0, N_0^* \to \max[N_1, N_2]\}$.

Now we assume (26) and we continue with the proof of (23).

From (26), by the assumptions (24) and (25) we obtain:

(27) $|f_1[n_1^*] - a_1| < \epsilon_1^*$

(28) $|f_2[n_1^*] - a_2| < \epsilon_1^*$

For proving (23), by (27, 28), it suffices to prove

(30) $|f_1[n_1^*] - a_1| < \epsilon_1^* \wedge |f_2[n_1^*] - a_2| < \epsilon_1^* \Rightarrow |(f_1 \oplus f_2)[n_0] - a_0^*| < \epsilon_0$.

Using the substitution from step (29), the goal (30) becomes:

(32) $|f_1[n_0] - a_1| < \epsilon_1^* \wedge |f_2[n_0] - a_2| < \epsilon_2^* \Rightarrow |(f_1 \oplus f_2)[n_0] - a_0^*| < \epsilon_0$.

Using (sum of functions:), the goal (32) is transformed into:

(35) $|f_1[n_0] - a_1| < \epsilon_1^* \wedge |f_2[n_0] - a_2| < \epsilon_1^* \Rightarrow |(f_1[n_0] + f_2[n_0]) - a_0^*| < \epsilon_0$.

Formula (35) is proved because it becomes an instance of (distance of sum) by taking: $\{a_0^* \to a_1 + a_2, \epsilon_1^* \to \frac{\epsilon_0}{2}\}$.