It is well recognized that the notion of abstract data type is the key notion in modularization which is imperative in developing reliable and maintainable software.

An abstract data types defines data (to be encapsulated) with operations for manipulating those data. Algebraic specification based on equational logic (or its extension) is widely accepted in specifying abstract data types. In algebraic specification, an abstract data type is interpreted as a class of quotient algebras given by the equivalence class (over terms) defined by equational axioms.

On the other hand, in programming language research communities, a type system is usually formulated as a typed λ-calculus. Mitchell and Plotkin showed that abstract data types can be understood as existential type, which can be incorporated in typed λ-calculi. Scott's domain theory is widely used in interpreting types of programming languages (and of typed λ-calculi). This theory is based on an order-topology reflecting the notion of approximation/convergence of computation. This notion, however, has no relationship with quotient algebras for semantics of specifications of abstract data types. In summary, a specification of an abstract data types and a program for the same abstract data type lack any semantical correspondence.

In order to remedy this situation, it is much desirable to describe specifications and programs in a common semantical framework. That is, if we describe, with a wide-spectrum language, the specification and the program of an abstract data type, then we can give semantic foundations to the correctness (and its verification) between the program and the specification.

In this work, we propose a typed λ-calculus Funiq for wide-spectrum languages. Funiq is based on Cardelli & Wegner's Fun which is highly evaluated as a calculus for typed functional programming languages. Funiq enriches Fun with refined types which denote subsets. A refined type is defined with the well-known comprehension scheme (in set theory) with inequations as predicates. An inequation intuitively specifies the partial correctness of the behavior of equipped operations of abstract data types while an equation in algebraic specification roughly correspond to the total correctness. Then we formalize the type system of Funiq as a type theory called FUNIQ.

Our wide-spectrum calculus Funiq and its type theory FUNIQ are shown to have following properties which are always expected to any base calculi for practical computer languages.

(1) Proof-theoretical properties:
- the conservative extension property of the type theory FUNIQ with respect to the base type theory FUN (of Cardelli & Wegner's Fun);
- the faithfulness of the compile-time type-checking with respect to the partial correctness;
- the conservative extension property of the enriched (wide-spectrumized) type theory with respect to the base type theory (of a programming calculus) other than FUN, especially the base type theory with recursive types;

(2) Reduction-theoretic (operational semantic) properties:
- the subject-reduction property guaranteeing the type of an expression is preserved under reduction;
- the strong normalization property (providing the termination of evaluation) of expressions without the fixed-point
- the confluence property for the independence of the result of evaluation from the evaluation-order.

(3) Denotational semantic properties:
- the soundness of the type theory FUNIQ with respect to the cper (complete partial equivalence relation) semantics (over cpo);
- the type-safeness which guarantees that any syntactically typable expression does not run into run-time type-error;

These properties show that Funiq can be accepted as a base calculus for practical languages, hence Funiq and its type theory FUNIQ can nicely be extended to practical wide-spectrum languages.