1. Record Nr. UNISA996465908003316

Titolo	Types in Compilation [[electronic resource]] : Third International Workshop, TIC 2000, Montreal, Canada, September 21, 2000. Revised Selected Papers / / edited by Robert Harper
Pubbl/distr/stampa	Berlin, Heidelberg : , : Springer Berlin Heidelberg : , : Imprint : Springer, , 2001
ISBN	3-540-45332-6
Edizione	[1st ed. 2001.]
Descrizione fisica	1 online resource (X, 214 p.)
Collana	Lecture Notes in Computer Science, , 0302-9743 ; ; 2071
Disciplina	005.4/53
Soggetti	Programming languages (Electronic computers) Computer logic Mathematical logic Programming Languages, Compilers, Interpreters Logics and Meanings of Programs Mathematical Logic and Formal Languages
Lingua di pubblicazione	Inglese
Formato	Materiale a stampa
Livello bibliografico	Monografia
Note generali	Bibliographic Level Mode of Issuance: Monograph
Nota di bibliografia	Includes bibliographical references at the end of each chapters and index.
Nota di contenuto	Types in Compilation -- Sound and Complete Elimination of Singleton Kinds -- Program Representation Size in an Intermediate Language with Intersection and Union Types -- An Abstract Model of Java Dynamic Linking and Loading -- Sharing in Typed Module Assembly Language -- Scalable Certification for Typed Assembly Language -- Safe and Flexible Dynamic Linking of Native Code -- Alias Types for Recursive Data Structures.
Sommario/riassunto	The importance of typed languages for building robust software systems is, by now, an undisputed fact. Years of research have led to languages with richly expressive, yet easy to use, type systems for high-level programming languages. Types provide not only a conceptual framework for language designers, but also a ord positive bene ts to the programmer, principally the ability to express and enforce levels of abstraction within a program. Early compilers for typed languages followed closely the methods used for their untyped

counterparts. The role of types was limited to the earliest s- ges of compilation, and they were thereafter ignored during the remainder of the translation process. More recently, however, implementors have come to - cognize the importance of types during compilation and even for object code. Several advantages of types in compilation have been noted to date: { They support self-checking by the compiler. By tracking types during c- pilation it is possible for an internal type checker to detect translation errors at an early stage, greatly facilitating compiler development. { They support certi cation of object code. By extending types to the ge- rated object code, it becomes possible for a code user to ensure the basic integrity of that code by checking its type consistency before execution. { They support optimized data representations and calling conventions, even in the presence of modularity. By passing types at compile-, link-, and even run-time, it is possible to avoid compromises of data representation imposed by untyped compilation techniques.