

Contents

1	Introduction	1
1.1	Motivation	2
1.1.1	Code Generation	2
1.1.2	System Abstractions	3
1.2	Our Approach: Certifying Translations	5
1.3	Contributions	7
1.3.1	General Methodology	7
1.3.2	Certifying Code Generation	7
1.3.3	Certifying System Abstractions	8
1.3.4	Evaluation	9
1.3.5	Restrictions	9
1.4	Overview of the Thesis	9
2	Related Work	11
2.1	Work on Behavioral Equivalence	11
2.2	Work on Compiler Correctness	11
2.2.1	Work Related to the Certifying Translations Approach	12
2.2.2	Work Related to Certified Compilers	14
2.3	Work on Correct System Abstractions	15
3	Prerequisites	17
3.1	Basic Mathematical Notions	17
3.2	Transition Systems and Simulation	17
3.3	A Brief Introduction to Isabelle/HOL	24
3.4	A Brief Introduction to Coq	27
4	A Framework for Certifying Translations	31
4.1	Our Semantic Framework	32
4.1.1	Framework Specification	32
4.1.2	Derived Correctness Criteria	41
4.1.3	Applying the <i>Generic Correctness Criteria</i> to Verification Scenarios	43
4.2	Embedding System Descriptions into the Framework - Shallow and Deep Embedding	43
4.3	Working with the <i>Generic Correctness Criteria</i> Family	45
4.3.1	Partial Orders of Correctness Criteria, State Correspondence and Simulation Relations	45
4.3.2	Handling Inputs and Non-Determinism	48

4.3.3	Combining Correctness Criteria	48
4.4	Our Correctness Criteria and their Relation to Other Formalisms	50
4.4.1	Final Value Semantics	51
4.4.2	Explicit Trace Representations in Transition Systems	51
4.4.3	Coalgebraic Trace Representations	52
4.4.4	Explicit vs. Implicit Trace Representation	54
4.5	Discussion: Formalizing the Framework in a Theorem Prover	55
5	Certifying Code Generation	57
5.1	Syntax and Semantics of an Intermediate Language	58
5.2	Syntax and Semantics of MIPS code	64
5.3	The Code Generation Algorithm	67
5.4	Correct Code Generation	68
5.4.1	Adapting the <i>Generic Correctness Criterion</i>	69
5.4.2	Proof Sketch on Proving Code Generation Runs Correct	72
5.4.3	Analyzing Proving Effort	74
5.4.4	Setting up the Proof: Establishing the Simulation Relation	74
5.4.5	Proving the Prerequisites	76
5.4.6	Proving the Steps Correct	78
5.4.7	Putting it All Together	86
5.5	Generating and Handling of the Certificates	86
5.5.1	Program Representation Generation	87
5.5.2	Certificate Generation Algorithm	88
5.5.3	Preparing for the Theorem Prover	91
5.6	Using Checker Predicates to Optimize the Proof Process	92
5.6.1	The Nature of Checker Predicates	92
5.6.2	A Fast Injectivity Proof	93
5.7	A Checker Predicate for Semantical Equivalence	94
5.7.1	Our Checker Intermediate Language Semantics	95
5.7.2	Our Checker MIPS Semantics	99
5.7.3	Verifying Steps using the Checker Semantics	102
5.7.4	A Checker Predicate Implementation for Fast Step Correctness Checking	105
5.7.5	Checker Predicate Correctness	107
6	Verifying System Abstractions	113
6.1	SAS Transition Systems	115
6.2	Correctness of Abstractions	117
6.2.1	Formalizing Properties	117
6.2.2	Property Preservation via Simulation	119
6.3	Proving Abstractions Correct	124
7	Evaluation	127
7.1	Our Certifying Compilers	127

7.2	Verification of System Abstractions	128
7.3	Certificate Checking with Isabelle and Coq	129
7.3.1	Look-up Operations	129
7.3.2	Other Performance Effecting Formalization Choices	132
7.3.3	Certificate Checking for Code Generation	132
7.3.4	Speedup using Checkers and Explicit Proof Terms	134
7.3.5	Proof Checking for System Abstractions	135
7.4	Framework Evaluation	136
7.5	Some Concluding Remarks	136
8	Conclusion	137
	Bibliography	141
A	Coq Formalization of Intermediate and MIPS language	151
A.1	Intermediate Language	151
A.2	MIPS language	155
B	Correctness Formalization in Coq	161
C	Generation of Code Generation Certificates	163
C.1	Variable-Assignment Rules	163
C.2	Array-Assignment Rule 1	170
C.3	Branch-Statement Rules	170
C.4	Procedure Call and Return Rules	171
C.5	Print-Statement Rules	173