

Contents

1	Introduction	1
1.1	Java concurrency	2
1.2	Historical overview	7
1.3	Contributions	8
1.4	Isabelle/HOL	13
1.4.1	Notation	14
1.4.2	Locales	17
1.4.3	Induction and coinduction	19
2	Sequential JinjaThreads	23
2.1	Source code	23
2.1.1	Abstract syntax	24
2.1.2	Type system	29
2.1.3	Native methods	34
2.1.4	Well-formedness	35
2.1.5	Dynamic semantics	37
2.1.6	Type safety	42
2.2	The JinjaThreads virtual machine	43
2.2.1	The bytecode language	44
2.2.2	Semantics	46
2.2.3	Well-typings	50
2.2.4	Type safety	54
2.3	Comparison with Jinja, Bali, and μ Java	55
3	Interleaving semantics	59
3.1	Framework for interleaving semantics	60
3.1.1	The multithreaded state	61
3.1.2	Thread actions	66
3.1.3	Interleaving semantics	75
3.1.4	Infrastructure for well-formedness constraints	78
3.2	Multithreading in JinjaThreads	82
3.2.1	Native methods for synchronisation	82

3.2.2	Source code	91
3.2.3	Bytecode	96
3.3	Deadlock and type safety	100
3.3.1	Deadlock as a state property	101
3.3.2	Deadlock for threads	105
3.3.3	Progress up to deadlock	109
3.3.4	Type safety for source code	112
3.3.5	Type safety for bytecode	122
3.4	Related work	127
3.4.1	Formalisations of Java and Java bytecode	127
3.4.2	Type safety proofs and deadlocks	129
3.4.3	Large-scale programming language formalisations	130
4	Memory models	131
4.1	The heap as a module	132
4.1.1	Abstract operations and their properties	133
4.1.2	Adaptations to semantics and proofs	136
4.1.3	Design considerations	140
4.2	Sequential consistency	141
4.3	Java memory model	142
4.3.1	Informal explanation	143
4.3.2	Formal definition	148
4.3.3	The data race freedom guarantee	164
4.3.4	Consistency	185
4.3.5	Type safety	188
4.3.6	Discussion	192
4.4	Related work	201
4.4.1	Memory models and data race freedom	201
4.4.2	Abstract heap modules	203
4.4.3	Modular formalisations	204
5	Compiler	205
5.1	Semantic preservation via bisimulation	208
5.1.1	Semantic preservation	208
5.1.2	Simulation properties	210
5.1.3	Lifting simulations in the interleaving framework	218
5.1.4	Semantic preservation for the Java memory model	225
5.2	Explicit call stacks for source code	226
5.2.1	State and semantics	227

5.2.2	Semantic equivalence	232
5.3	Register allocation	236
5.3.1	Intermediate language J_1	236
5.3.2	Compilation stage 1	242
5.3.3	Preservation of well-formedness	243
5.3.4	Semantic preservation	244
5.4	Code generation	250
5.4.1	Compilation stage 2	250
5.4.2	Preservation of well-formedness	251
5.4.3	Semantic preservation	252
5.5	Complete compiler	255
5.6	Discussion	257
5.7	Related work	259
6	JinjaThreads as a Java interpreter	261
6.1	Isabelle code extraction facilities	262
6.1.1	The code generator	263
6.1.2	The predicate compiler	265
6.1.3	Data structures	266
6.1.4	Locales and code extraction	267
6.2	Static semantics	268
6.2.1	Generic well-formedness	268
6.2.2	The bytecode verifier	269
6.2.3	Type inference for source code	271
6.3	Interpreter and virtual machine	272
6.3.1	The single-threaded semantics	272
6.3.2	Schedulers	274
6.3.3	Tabulation	276
6.3.4	Efficiency of the interpreter	277
6.4	Guidelines for executable formalisations	280
6.5	The translator Java2Jinja	282
6.5.1	The translation	284
6.5.2	Validation	287
6.6	Related Work	289
7	Discussion and Future Work	291
7.1	Efforts and rewards of a machine-checked formalisation	291
7.2	Experience: Working with Isabelle/HOL	294
7.3	From Java ^{light} to JinjaThreads	299

7.4	Comparison between Java and JinjaThreads	301
7.5	Future work	305
8	Conclusion	307
A	Producer-consumer example	311
B	Formal definitions	315
B.1	Declarations and lookup functions	315
B.2	Binary operators	317
B.3	Heap module implementations	319
B.3.1	Sequential consistency	319
B.3.2	The Java memory model	321
B.4	Native methods	322
B.4.1	Signatures	322
B.4.2	Semantics of method <i>clone</i>	323
B.4.3	Semantics of native methods	324
B.4.4	Observability	327
B.5	Generic well-formedness	327
B.6	Source code	328
B.6.1	Syntax	328
B.6.2	Typing rules for expressions	328
B.6.3	Definite Assignment	330
B.6.4	Well-formedness	332
B.6.5	Small-step semantics	332
B.6.6	Observability	338
B.7	Bytecode	339
B.7.1	Syntax	339
B.7.2	Applicability and effect	339
B.7.3	The virtual machine	343
B.7.4	Observability	348
B.8	The Java memory model	349
B.9	The compiler	352
B.9.1	Program compilation	352
B.9.2	Compilation stage 1	352
B.9.3	Compilation stage 2	353
B.9.4	Preprocessor	357
	List of Figures	361

List of Tables	365
Bibliography	367
Index	389