A Dual-Engine for Early Analysis of Critical Systems

Aboubakr Achraf El Ghazi,
Ulrich Geilmann, Mattias Ulbrich, Mana Taghdiri

Karlsruhe Institute of Technology, Germany

October 6, 2011

Motivation

- Alloy is a widely-used modeling language
  - Stand-alone framework for checking high-level system designs
    - Network protocols
    - File system policies
    - Schedulers
  - Intermediate language for checking programs
    - Functional properties of Java programs
    - Test case generation
    - Specification extraction
  - Engine for generating counterexamples
    - For theorem provers
- Taught in about 30 universities
- Used in industry (AT&T, Telcordia, etc.)

Alloy for critical systems

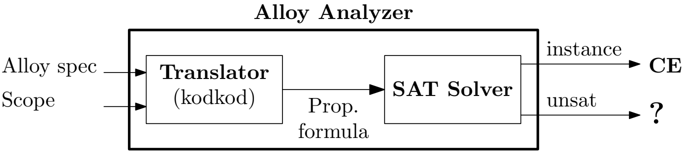
- The intentional naming system for dynamic networked environments [Khurshid et. al, 2000]
- The NASA's Direct-To system for helping air traffic controllers [Ghassemi et. al, 2001]
- A role-based access control schema for sensitive resources [Zao et. al, 2003]
- A pull-based asynchronous rekeying framework in secure multi-cast [Taghdiri et. al, 2003]
- The Mondex electronic purse system [Ramananandro et. al, 2007]
- The New York City subway signaling system [Sarma et. al, 2008]
- The flash file system, responsible for NASA's mars rover breakdown [Kang et. al, 2008]
- The security domain model analysis for illicit information flows [Shaffer et. al, 2008]
- A constraint analysis on Java Bytecodes for security vulnerabilities [Reynolds et. al, 2010]

The Alloy language

- Declarative modeling language
  - Support for various abstraction levels
  - Various development/interest levels
- Simple, uniform semantics
  - Everything is a relation
  - Semantics equivalent to first-order relational logic
- Expressive, familiar syntax
  - Set and relational operators, first-order quantifiers, transitive closure, linear integer arithmetic
  - Concise formulation of rich properties
  - Syntax similar to an OO language
- Analyzable
  - The Alloy Analyzer (AA) is fully automatic
  - Looks for an instance violating a given assertion

The Alloy Analyzer (AA)

- Performs bounded analysis
  - Requires a user-provided scope
  - Reduction to a satisfiability problem (SAT)
  - Enables automation



- Shortcomings
  - Can never prove an assertion correct, even for the simplest models
  - Limited support for numerical expressions
  - These are critical for critical infrastructures

▷ Need for an automatic, proof-capable engine!

Verification – *Approach*

Undecidable

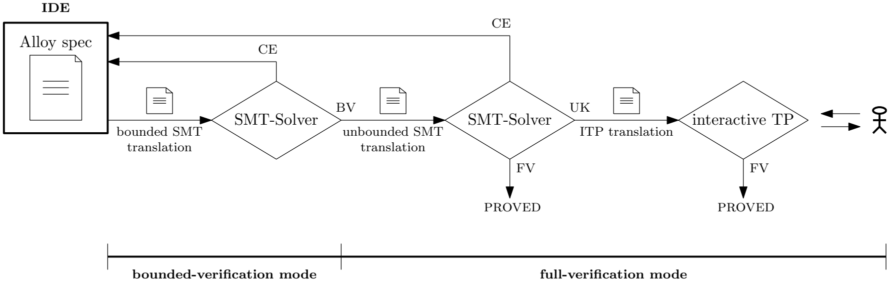
- SMT engine [GT11]
  - No type finitization $\Rightarrow$ Can prove valid assertions
  - Fully automatic
  - Full support of linear integer arithmetic
- Interactive theorem proving – *only as a last resort*
  - Semi-decidable – *FOL version*
  - Needs correctness confidence
- Bounded verification
  - Economical, and suitable for counterexamples
  - If unsat, provides confidence in correctness

$\Rightarrow$ *The larger the bounds, the higher the confidence*

Verification – *Approach*

A Dual-Engine

- Counterexample/Confidence:
  - SMT-based bounded analysis (UFBV): Decidable and fully automatic
  - Improvements in performance and scalability
- Verification:
  - SMT-based engine: not complete, but still fully automatic in many cases
  - ITP-based framework: in general interactive, but complete



Example – a simple file system

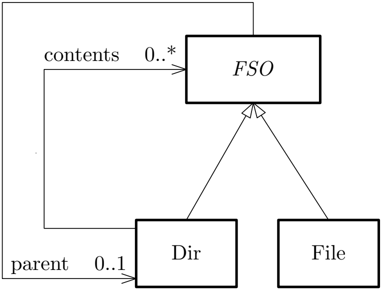
Alloy Model:

```
abstract sig FSO {
 parent: lone Dir
}
sig Dir extends FSO {
 contents: set FSO
}
sig File extends FSO {}
fact {
 contents = ~parent
 all d:Dir| not(d in d.^contents)
 all d:Dir| #(d.contents) <= 5
}
assert oneLocation {
 all o:FSO, lone d:FSO| o in d.contents
}
check oneLocation for 8
```

Example – declarations

Alloy Model:

```
abstract sig FSO {
 parent: lone Dir
}
sig Dir extends FSO {
 contents: set FSO
}
sig File extends FSO {}
fact {
 contents = ~parent
 all d:Dir| not(d in d.^contents)
 all d:Dir| #(d.contents) <= 5
}
assert oneLocation {
 all o:FSO, lone d:FSO|
  o in d.contents
}
check oneLocation for 8
```



SMT-based bounded Analysis – *type hierarchy*

- Bitvector sorts for Alloy top-level types
- Membership function for Alloy types
- FOL axioms to enforce subtyping

Alloy Model:

```
abstract sig FSO {
 parent: lone Dir
}
sig Dir extends FSO {
 contents: set FSO
}
sig File extends FSO {}
```

Z3 Spec:

$\lceil \log_2(|FSO|) \rceil$

```
(define-sort () FOS BitVec[X])

(declare-fun isFSO (FSO) Bool)
(declare-fun isDir (FSO) Bool)
(declare-fun isFile (FSO) Bool)

(assert (forall (f FSO)
 (=> (isDir f) (isFSO f))))
(assert (forall (f FSO)
 (=> (isFile f) (isFSO f))))
```

SMT-based bounded Analysis – *type hierarchy*

- Abstract types are the union of their subtypes
- Extension types are disjoint

Alloy Model:

```
abstract sig FSO {
 parent: lone Dir
}
sig Dir extends FSO {
 contents: set FSO
}
sig File extends FSO {}
```

Z3 Spec:

```
(assert (forall (f FSO)
 (=> (isFSO f)
  (or (isDir f)(isFile f)))))

(assert (forall (f FSO)
 (not
  (and (isDir f)(isFile f)))))
```

SMT-based bounded Analysis – *relations*

- Membership function and type-enforcing axioms for each relation
- Uninterpreted functions to enforce multiplicity keywords

parent: FSO -> lone Dir

Alloy Model:

```
abstract sig FSO {
 parent: lone Dir
}
sig Dir extends FSO {
 contents: set FSO
}
sig File extends FSO {}
```

Z3 Spec:

parent: FSO -> FSO -> Bool

```
(declare-fun parent (FSO FSO) Bool)
(declare-fun contents (FSO FSO) Bool)
(assert (forall (f FSO)(g FSO)
 (=> (parent f g)
     (and (isFSO f)(isDir g)))))
(assert (forall (f FSO)(g FSO)
 (=> (contents f g)
     (and (isDir f)(isFSO f)))))
(declare-fun oneParent (FSO) FSO)
(assert (forall (f FSO)(g FSO)
 (=> (parent f g)
     (= g (oneParent f)))))
```

ITP – *type hierarchy*

- Sorts for relations and tuples: `Relation`, `Tuple`
- Membership predicate: `in(Tuple,Relation)`
- Arities of tuples and relations are captured by subtypes: `Tuple2` $\sqsubseteq$ `Tuple`, etc.
- Constructor functions freely generate `Tuple2`, `Tuple3`, etc.:
  `Tuple2 binary(Atom,Atom)`,
  `Tuple3 ternary(Atom,Atom,Atom)`
- Alloy's operators built explicitly – *higher level reasoning*
  `subset(Relation,Relation)`,
  `Rel1 union1(Rel1,Rel1)`, `Rel2 union2(Rel2,Rel2)`
- Reasoning via sequent calculus (KeY):

$$\bigwedge \mathcal{M} \wedge \bigwedge \mathcal{F} \Rightarrow a \;\rightsquigarrow\; [\![\mathcal{M}]\!], [\![\mathcal{F}]\!] \vdash [\![a]\!]$$

ITP – *type hierarchy*

```
\sorts{
  Relation;
  Rel1 \extends Relation;
...
```

KeY Spec:

```
Rel1 FSO; Rel1 Dir; Rel1 File;
Rel2 parent; Rel2 contents;
...
∀Atom this; (in(this,Object)
 ⇒ (in(this,File) | in(this,Dir))),
subset(parent,prod1x1(Object,Dir)),
∀Atom this; (in(this,Object)
 ⇒ lone(join1x2(sin(this),parent))),
subset(File,Object),
subset(Dir,Object),
subset(contents,prod1x1(Dir,Object)),
disj(File,Dir),
⊢
...
```

Alloy Model:

```
abstract sig FSO {
 parent: lone Dir
}
sig Dir extends FSO {
 contents: set FSO
}
sig File extends FSO { }
```

Proving Strategy

KeY's proof strategy extended for efficient relational reasoning

- Expand predicate invocations in the succedent to their definitions: universal quantification is eliminated by skolemization
- Keep predicate invocations in the antecedent; use lemmas to exploit their semantics, e.g.

```
\assumes (subset(r,s) ⊢)
\find (in(a,r) ⊢)
\add (in(a,s) ⊢)
```

- Lemmas capture simple properties of transitive closure, e.g.

```
\find (in(binary(a,b),transClos(r)) ⊢)
\add (∃Atom c; in(binary(c,b),r) ⊢)
```

- Simplification lemmas are applied greedily, e.g.

```
\find (union1(r,r))
\replacewith (r)
```

- $\sim$ 500 provided and proved lemmas

Evaluation

		AA		bounded Z3		unbounded Z3		KeY	
Property	Scope	Time	res	Time	Res	Time	Res	Step	Res
delUndoesAdd -Buggy	16	0.8	CE	0.4	CE	–	NA	–	NA
	32	58	CE	1.0	CE				
delUndoesAdd	32	150	BV	0.0	BV	0.0	FV	–	NA
	64	TO	UK	0.0	BV				
lookupYields	8	101	BV	147	BV	TO	UK	122	FV
	16	TO	UK	TO	UK				

		AA		bounded Z3		unbounded Z3		KeY	
Property	Scope	Time	res	Time	Res	Time	Res	Time	Res
BuggyCOM *Theorem 1*	16	427	CE	3.6	CE	–	NA	–	NA
	17	TO	CE	1.9	CE				
COM *Theorem 1*	16	451	BV	0.0	BV	0.0	FV	–	NA
	17	TO	UK	0.3	BV				
mark sweep *Soundness 1*	9	140	BV	17	BV	TO	UK	10	FV
	10	TO	UK	107	BV				

CE: counterexample, **BV:** bounded valid, **FV:** fully valid, **UK:** unknown, **NA:** not applicable, **TO:** time out ($>$ 10 *min*.)

Related work

- Dynamite [Frias, Pombo, Moscato, 2007] uses PVS
- Prioni [Arkoudas, Khurshid, Marinov, Rinard, 2003] uses Athena
    - Based on interactive theorem provers
    - Interactive, regardless of the complexity of the problem
    - No support of integer or cardinality expressions
    - (Dynamite) Reduction to binary relations results in additional proof obligations
- SMT-based engine [El Ghazi, Taghdiri, 2011] uses Z3
    - Fully automatic proofs
    - support of integer and cardinality expressions
    - No completeness guarantee

Conclusions

- A dual engine capable of proving and refuting Alloy assertions
  - Via tool chain – *from fully automatic to interactive proving*
  - Bounded SMT engine that improves AA – *counterexample/confidence*
  - Unbounded SMT engine – *fully automatic proof capability*
  - ITP framework in KeY– *interactive but complete*
- Reduces cost and increases flexibility
  - Cost should depend on problem complexity
  - In earlier software development stages, flaws are expected

References I

K. Arkoudas, S. Khurshid, D. Marinov, and M. Rinard.
Integrating model checking and theorem proving for relational reasoning.
RELMICS, pages 21–33, 2003.

M. F. Frias, C. G. L. Pombo, and M. M. Moscato.
Alloy Analyzer+PVS in the analysis and verification of alloy specifications.
In *TACAS*, pages 587–601, 2007.

Gerhard Gentzen.
Untersuchungen über das logische Schließen.
Mathematische Zeitschrift, 39:176–210, 405–431, 1935.

A. A. El Ghazi and M. Taghdiri.
Analyzing Alloy constraints using an SMT solver: A case study.
In *AFM*, Edinburgh, United Kingdom, 2010.

References II

- Aboubakr Achraf El Ghazi and Mana Taghdiri.
  Relational reasoning via SMT solving.
  In *17th International Symposium on Formal Methods (FM)*, 2011.
- T. Lev-ami, N. Immerman, T. Reps, M. Sagiv, and et al.
  Simulating reachability using first-order logic.
  IN CADE-20, pages 99–115, 2005.
- R. Leino and R. Monahan.
  Reasoning about comprehensions with first-order SMT solvers.
  In *SAC*, pages 615–622, 2009.
- P. Suter, R. Steiger, and V. Kuncak.
  Sets with cardinality constraints in satisfiability modulo theories.
  In *VMCAI*, 2011.