Extending Alloy to Express and Analyze Optimization Constraints

CS846 Project Presentation

Steven.Stewart@uwaterloo.ca

Overview

- Motivation, Technology, and Context
- Methodology and Implementation
- Demonstration
- Concluding Remarks

Motivation, Technology, and Context

Motivation

- *Feature configuration problem*:
  - How do we obtain the optimal configuration of features for a software product?
- *Solution*: combine lightweight modelling with discrete multiobjective optimization

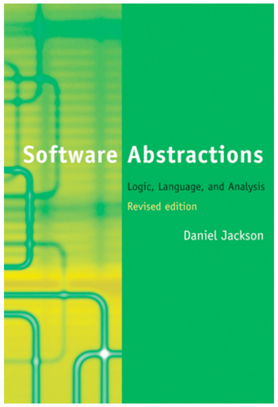
Motivation

- *My objective*: enable the ability to express MOOPs (multiobjective optimization problems) in Alloy

Technology

- Lightweight modelling
  - Alloy (http://alloy.mit.edu/alloy/)
- Multiobjective optimization using Moolloy (http://sdg.csail.mit.edu/moolloy/)
- Feature models & Design Space Modelling

Alloy



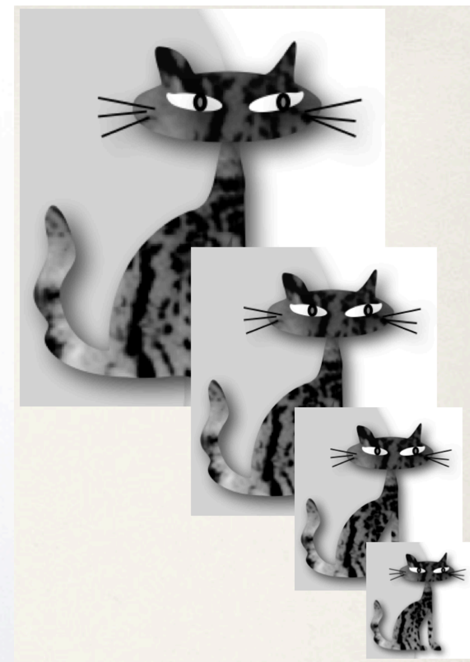
- A Language and Tool for Relational Models (http://alloy.mit.edu/alloy/)
- Logic and Language
  - First-order logic and transitive closure

Alloy

- Analysis
  - Model-finding / Simulation
  - Refutation: check assertions against a huge set (likely billions) of test cases to find a counter-example

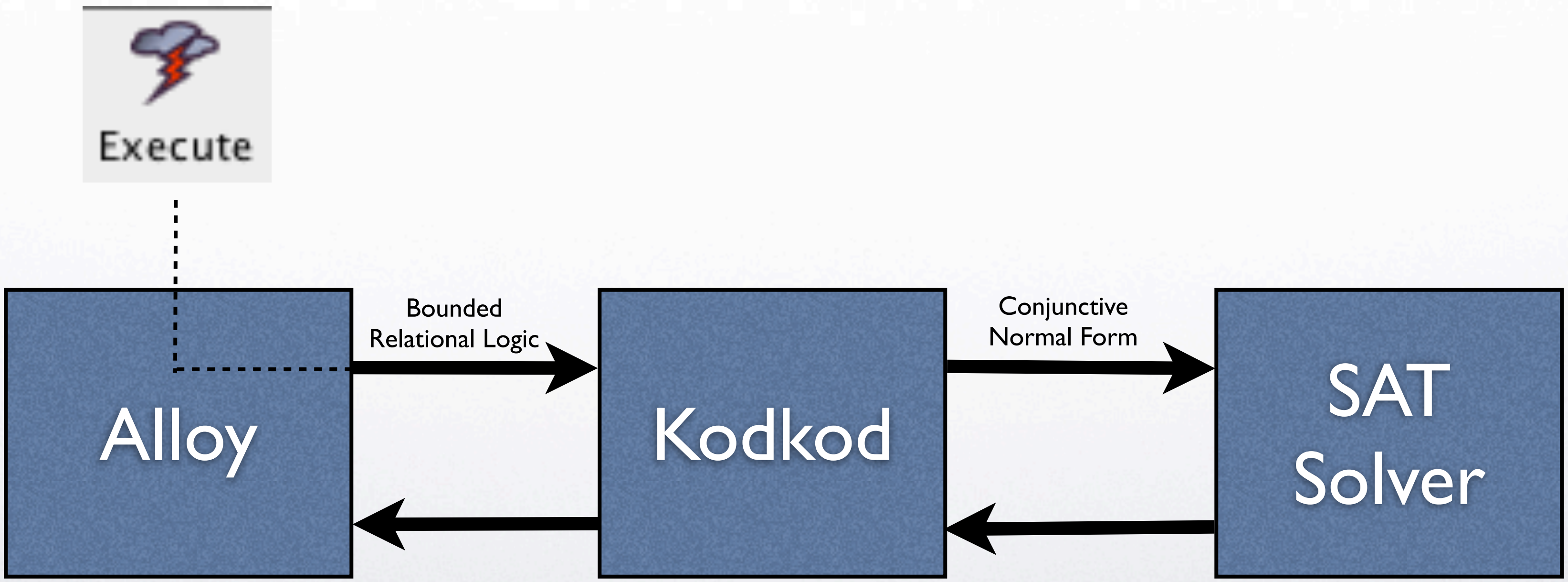
Alloy+Kodkod

- The Alloy compiler translates a specification into a Kodkod formula
- Kodkod[1] (a relational constraint solver) translates its "bounded relational logic" to CNF using novel techniques
- Kodkod passes the formula to a backend SAT solver



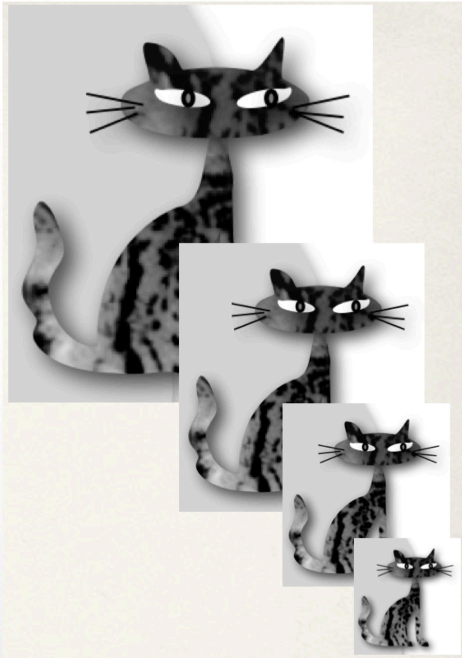
[1]E. Torlak and D. Jackson. "Kodkod: A Relational Model Finder," *Proceedings of 13th International Conference on Tools and Algorithms forConstruction and Analysis of Systems*, Braga, Portugal, April 2007. Lecture Notes in Computer Science, Vol. 4424, Berlin, Springer- Verlag, pp.632-647.

Alloy+Kodkod

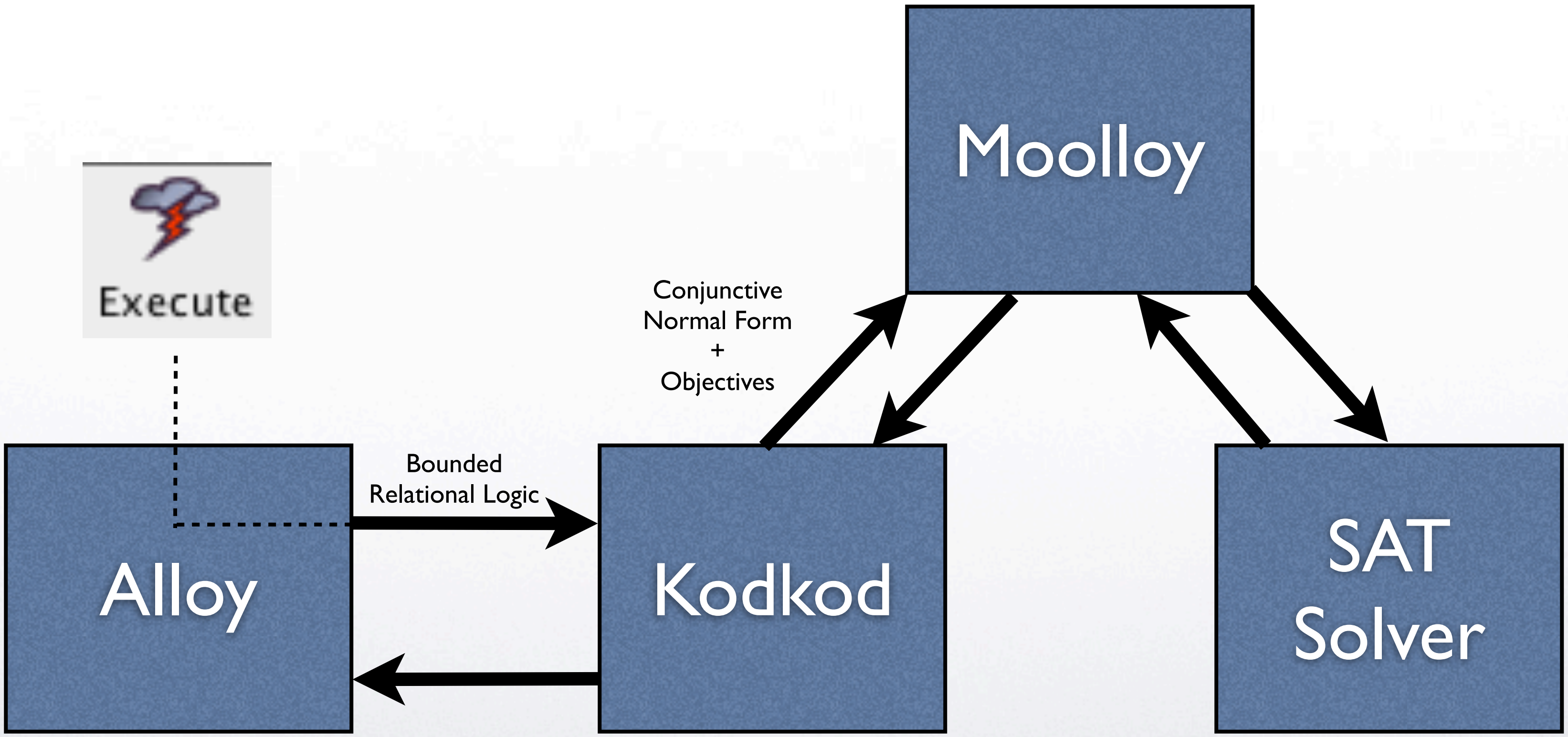


Alloy+MOOP

- Alloy calls Kodkod (as per usual)
- Kodkod, instead, passes control to Moolloy
- Moolloy uses the "Guided Improvement Algorithm" for solving discrete MOOPs



Alloy+MOOP

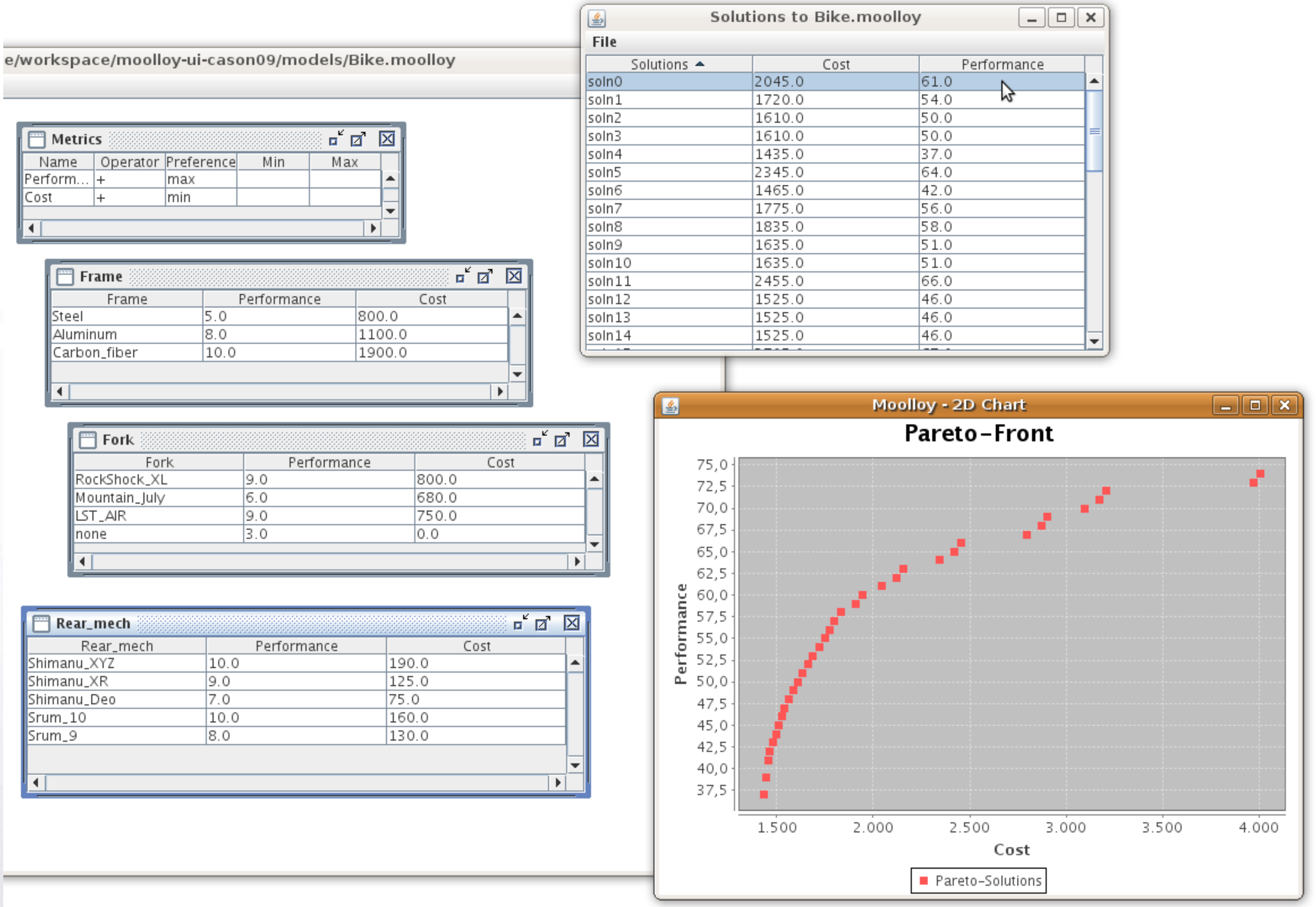

Execute
Moolloy
Conjunctive
Normal Form
+
Objectives
Bounded
Relational Logic
Alloy
Kodkod
SAT
Solver

Optimization

- Single objective
  - 0-1 Knapsack: maximize the value of the contents of your knapsack subject to a weight restriction
  - one optimal solution

Optimization

- Multiobjective
  - maximize performance; minimize cost; maximize stability; minimize energy use...
  - one or more optimal solutions representing the trade-offs among the objectives



D. Rayside and H.-C. Estler. "A Spreadsheet-like User Interface for Combinatorial Multi-Objective Optimization," CASCON'09.

Moolloy+GIA

- The "guided improvement algorithm" (GIA[1]) is used by Moolloy to solve MOOPs
- It repeatedly adjusts a Kodkod formula to ask for better and better solutions
- When no better solution exists, then an optimal solution has been identified on the Pareto Front
- Moolloy provides *exact* solutions

[1]D. Rayside, H.-C. Estler and D. Jackson. "A Guided Improvement Algorithm for Exact, General Purpose, Many-Objective Combinatorial Optimization," MIT-CSAIL-TR-2009-033.

Alloy + Partial Instances

- Alloy generates tuples for each relation bounded by a specified scope
- The ability to specify *partial instances* is a pre-requisite for Alloy+MOOP, because we need relations that map *specific* features to their metric values

```
inst inventory {
   exactly 1 Product,    --explore possible configs of one product
   6 Int,                --large enough integers for our metrics

   --inventory of options for each feature
   F1 = F1O1 + F1O2 + F1O3,
   F2 = F2O1 + F2O2 + F2O3,

   --assignment of values to metrics for each option
   m1 = F1O1->10 + F1O2->15 + F1O3->5 +
           F2O1->4 + F2O2->16 + F2O3->8,
   m2 = F1O1->5 + F1O2->7 + F1O3->3 +
           F2O1->8 + F2O2->5 + F2O3->2
}
```

Context

- Design Space Modelling and Analysis (Cai and Sullivan)
- MOOPs may arise when we consider possible decisions in software design
- Make optimal design decisions in terms of algorithm and data structure selection
- Minimize the impact of changes on other modules

Context

- Feature-oriented software development[1]
  - A *feature* is a unit of functionality that satisfies some requirement
- Software systems are decomposed into their features
- Software Product Lines (SPLs) are generated from a set of features (i.e., configurations)

[1]S. Apel and C. Kastner. An overview of feature-oriented software development. Journal of Object Technology, 8, 2009.

Context

- What if we have limited resources?

  (i.e., CPU speed, memory, battery)

- Select features that satisfy stakeholder requirements within this constrained context

  "optimal configurations"

Context

- The Alloy Analyzer can potentially allow us to step-through, and explore optimal configurations in an SPL
- We're essentially turning Alloy into a discrete MOOP solver

Methodology and Implementation

Methodology

- Update Alloy compiler
  - JFlex -- lexical analyzer
  - JavaCUP -- parser generator
  - Add new classes for AST

Methodology

- Translation of Alloy to Kodkod
  - Objectives must be translated and passed to Kodkod
- Update Kodkod to interact with Moolloy
- Alloy GUI is oblivious to backend changes

Implementation

- Updating JFlex (lex file)
  - Add new keywords:

    objectives

    maximize and **minimize**

    optimize

Alloy+MOOP - Grammar

- Alloy 4 grammar

```
paragraph ::= sigDecl | factDecl | predDecl | funDecl
| assertDecl | cmdDecl
```

- Alloy+MOOP

```
paragraph ::= sigDecl | factDecl | predDecl | funDecl
| assertDecl | cmdDecl | instDecl | objDecl
```

```
objectives myGoals {
    minimize energy,
    maximize performance,
    minimize memoryUse,
    maximize stability
}
```

inst block

run myPredicate **for** config
optimize myGoals

objectives block

Demonstration

Demonstration

- Now we can use Alloy to solve problems such as 0-1 Knapsack
- Imagine: "executable declarative specifications"
- In fact, "Squander" uses an Alloy-like language for executing declarative syntax in Java (performs well on NP-complete problems)

Demonstration

- Product configuration
  - A product is specified as having a set of features
  - For each feature, we can specify a value for the metrics we are interested in
- Alloy+MOOP will solve for optimal configurations

Mandatory			Optional		
FeatureA	**m1**	**m2**	**FeatureB**	**m1**	**m2**
FAV1	5	4	FBV1	3	5
FAV2	4	2	FBV2	3	2

Configurations

FeatureA	**FeatureB**	**totalM1**	**totalM2**	**Optimal?**
FAV1	none	5	4	no
FAV2	none	4	2	yes
none	FBV1	3	5	no
none	FBV2	3	2	no
FAV1	FBV1	8	9	no
FAV1	FBV2	8	6	yes
FAV2	FBV1	7	7	
FAV2	FBV2	7	4	yes

Concluding Remarks

Results

- The Alloy syntax has been extended to support the specification of optimization constraints
- The extension has enabled the ability to express and solve MOOPs via Alloy
- Rafael Olaechea will present his work (next) on translating Clafer to Alloy for MOOPs

Conclusions

- The new syntax enables the exploration of optimal configurations of software products

- This ability enables us to use Alloy as a MOOP solver, with the full-capabilities of Moolloy at our disposal

Lessons Learned

- Making changes to the Alloy compiler was *difficult*
  - Very little documentation
  - Some questionable design decisions
- Reminds us of the benefits of applying best-practices in our academic work

Future Work

- Additional work is currently underway to improve and better-define how the scope of relational variables are computed
- The problem of discontiguous integers leading to increased formula generation and solving time is still being addressed
- As part of their 4th-year design project, students are working on how to improve the visualization of Pareto-optimal solutions
- A group of nano students are writing Alloy models for discrete multiobjective optimization problems