Advancing Declarative Programming

Aleksandar Milicevic
Massachusetts Institute of Technology

May 07, 2015
What is **Declarative** Programming?

> What is declarative programming? 
> 
> It would be very nice to input this description into some suitably programmed computer, and get the computer to translate it automatically into a subroutine.
> 
> - C. A. R. Hoare
>   
> ["An overview of some formal methods for program design", 1987]
What is **Declarative** Programming?

- say **what**, not **how**

- describe what the program is intended to do
- in some terms that are both **expressive** and **easy** to use

*"It would be very nice to input this description into some suitably programmed computer, and get the computer to translate it automatically into a subroutine"*

- C. A. R. Hoare

*"An overview of some formal methods for program design", 1987*
What is **Declarative** Programming?

- say **what**, not **how**
  - describe what the program is intended to do in some terms that are both **expressive** and **easy** to use

  "It would be very nice to input this **description** into some suitably programmed computer, and get the computer to translate it **automatically** into a subroutine”

  - C. A. R. Hoare  
    [“An overview of some formal methods for program design”, 1987]
Spectrum of The Declarative Programming Space

- **spec engine apps**
  - formal logic
  - sophisticated search
  - complex algorithms, constraint solving

- **spec engine apps**
  - DSL
  - translation/compilation
  - domain-specific uses

[ABZ'12, SCP'14, ICSE'15]
[Onward'13]
[ABZ'14]
Spectrum of The Declarative Programming Space

(my previous work)

executable specs for Java
program synthesis

spec engine apps

formal logic
sophisticated search
complex algorithms,
constraint solving

spec engine apps

DSL
translation/compilation
domain-specific uses

• more powerful constraint solver
• capable of solving a whole new category of formal specifications

[Onward’13]

• model-based web framework
• reactive, single-tier, policy-agnostic
• what instead of how

[ABZ’14]

• unified specification & implementation language

[ABZ’12, SCP’14, ICSE’15]
Spectrum of The Declarative Programming Space

(my previous work)

- executable specs for java
- program synthesis

spec
- formal logic
- sophisticated search
- complex algorithms, constraint solving

engine
- squander
- jennisys

apps
- executable specs for java
- program synthesis

spec
- alloy*
- [ABZ'12, SCP'14, ICSE'15]

- more powerful constraint solver
- capable of solving a whole new category of formal specifications

engine
- translation/compilation
- domain-specific uses

apps
- model-based web framework
- reactive, single-tier, policy-agnostic
- what instead of how

[Onward'13]
Spectrum of The Declarative Programming Space

(executable specs for Java)

spec
formal logic

engine
sophisticated search

apps
complex algorithms, constraint solving

Program synthesis

spec
DSL

engine
translation/compilation

apps
domain-specific uses

[ABZ’12, SCP’14, ICSE’15]

[Onward’13]

• more powerful constraint solver
• capable of solving a whole new category of formal specifications

• model-based web framework
• reactive, single-tier, policy-agnostic
• what instead of how
Spectrum of The Declarative Programming Space

Spec engine apps
my previous work
- executable specs for java
- program synthesis

Spec engine apps
- formal logic
- sophisticated search
- complex algorithms, constraint solving

Spec engine apps
ARbY
- unified specification & implementation language
- more powerful constraint solver
- capable of solving a whole new category of formal specifications

Spec engine apps
SUNNY
- model-based web framework
- reactive, single-tier, policy-agnostic
- what instead of how

DSL translation/compilation
domain-specific uses

[ABZ'12, SCP'14, ICSE'15]

[ABZ'14]

[Onward'13]
ALLOY*: Higher-Order Constraint Solving

(my previous work)

- Executable specs for Java
- Program synthesis

spec engine apps
formal logic
sophisticated search
complex algorithms, constraint solving

spec engine apps
DSL translation/compilation
domain-specific uses

[ABZ’12, SCP’14, ICSE’15]

• More powerful constraint solver
• Capable of solving a whole new category of formal specifications

[ARBY]

• Unified specification & implementation language

[ABZ’14]

• Model-based web framework
• Reactive, single-tier, policy-agnostic
• What instead of how

[SUNNY]

[Onward’13]
What is Alloy*:

Alloy*: a more powerful version of the alloy analyzer
What is Alloy*:

Alloy*: a more powerful version of the alloy analyzer

**Typical uses of the alloy analyzer**

- bounded software verification → but no software synthesis
- analyze safety properties of event traces → but no liveness properties
- find a safe full configuration → but not a safe partial config
- find an instance satisfying a property → but no min/max instance
What is ALLOY*?

ALLOY*: a more powerful version of the alloy analyzer

typical uses of the alloy analyzer

- bounded software verification → but no software synthesis
- analyze safety properties of event traces → but no liveness properties
- find a safe full configuration → but not a safe partial conf
- find an instance satisfying a property → but no min/max instance

higher-order
**What is ALLOY\(^\ast\)**

ALLOY\(^\ast\): a more powerful version of the alloy analyzer

**Typical uses of the alloy analyzer**

- bounded software verification → but no software synthesis
- analyze safety properties of event traces → but no liveness properties
- find a safe full configuration → but not a safe partial conf
- find an instance satisfying a property → but no min/max instance

**ALLOY\(^\ast\)**

- capable of automatically solving arbitrary higher-order formulas
First-Order Vs. Higher-Order: clique

**first-order**: finding a graph and a clique in it
- every two nodes in a clique must be connected

![Graph Diagram]

- **n1** (key: 5)
- **n2** (key: 0)
- **n3** (key: 6)
- **n4** (key: 1)

edges

Alloy Analyzer: automatic, bounded, relational constraint solver

A solution (automatically found by Alloy):

\[ clq = \{n_1, n_3\} \]
**First-Order Vs. Higher-Order: clique**

**first-order:** finding a graph and a **clique** in it
- every two nodes in a clique must be connected

```
sig Node { key: one Int }
```

```
run {
some edges: Node -> Node |
some clq: set Node |
clique[edges, clq]
}
```

Alloy Analyzer: automatic, bounded, relational constraint solver

A solution (automatically found by Alloy):
- `clq = {n1, n3}`
**First-Order Vs. Higher-Order: clique**

**first-order:** finding a graph and a **clique** in it
- every two nodes in a clique must be connected

```plaintext
sig Node { key: one Int }

run {
    some edges: Node -> Node |
    some clq: set Node |
    clique[edges, clq]
}
```

```
Alloy Analyzer: automatic, bounded, relational constraint solver
```

- **Alloy Analyzer:**
  - automatically found a solution
  - `clq = {n1, n3}`
**First-Order Vs. Higher-Order: clique**

**first-order**: finding a graph and a clique in it

- every two nodes in a clique must be connected

```plaintext
sig Node { key: one Int }

run {
    some edges: Node -> Node |
    some clq: set Node |
    clique[edges, clq]
}

pred clique[edges: Node->Node, clq: set Node] {
    all disj n1, n2: clq | n1->n2 in edges
}
```

Alloy Analyzer: automatic, bounded, relational constraint solver

[a solution (automatically found by Alloy): clique = {n1, n3}]

```
```
**First-Order Vs. Higher-Order: clique**

**first-order**: finding a graph and a clique in it

- every two nodes in a clique must be connected

```alloy
sig Node { key: one Int }

run {
    some edges: Node -> Node |
    some clq: set Node |
    clique[edges, clq]
}

pred clique[edges: Node->Node, clq: set Node] {
    all disj n1, n2: clq | n1->n2 in edges
}
```

**Alloy Analyzer**: automatic, bounded, relational constraint solver
**First-Order Vs. Higher-Order: clique**

**first-order**: finding a graph and a clique in it
- every two nodes in a clique must be connected

**Alloy Analyzer**: automatic, bounded, relational constraint solver

**a solution** (automatically found by Alloy): \( \text{clq} = \{n_1, n_3\} \)
first-order: finding a graph and a **clique** in it
- every two nodes in a clique must be connected

```
sig Node { key: one Int }

run {
    some edges: Node -> Node |
    some clq: set Node |
    clique[edges, clq]
}

pred clique[edges: Node->Node, clq: set Node] {
    all disj n1, n2: clq | n1->n2 in edges
}
```

**Alloy Analyzer**: automatic, bounded, relational constraint solver

a **solution** (automatically found by Alloy): \( \text{clq} = \{n_1, n_3\} \)
**First-Order Vs. Higher-Order:** maxClique

**higher-order:** finding a graph and a maximal clique in it
- there is no other clique with more nodes

![Graph Diagram]

```
maxClique[edges: Node->Node, clq: set Node] {
    clique[edges, clq] all ns:
        not (clique[edges, ns] and #{ns} > #{clq})
}
```

expressible but not solvable in Alloy!

**Definition of higher-order (as in Alloy):**
- quantification over all sets of atoms
**First-Order Vs. Higher-Order: maxClique**

**higher-order**: finding a graph and a *maximal clique* in it
- there is no other clique with more nodes

```alloy
pred maxClique[edges: Node->Node, clq: set Node] {
    clique[edges, clq]
    all ns: set Node |
        not (clique[edges, ns] and #ns > #clq)
}
run {
    some edges: Node -> Node |
    some clq: set Node |
    maxClique[edges, clq]
}
```

expressible but not solvable in Alloy!

**definition of higher-order (as in Alloy):**
- quantification over all sets of atoms
higher-order: finding a graph and a maximal clique in it

- there is no other clique with more nodes

expressible but not solvable in Alloy!
First-Order Vs. **Higher-Order: maxClique**

**higher-order**: finding a graph and a maximal clique in it
- there is no other clique with more nodes

**expressible but not solvable** in Alloy!

---

**definition** of higher-order (as in Alloy):
- quantification over all sets of atoms
# Solving maxClique Vs. Program Synthesis

<table>
<thead>
<tr>
<th>Program Synthesis</th>
<th>maxClique</th>
</tr>
</thead>
<tbody>
<tr>
<td>find some program AST s.t., for all possible values of its inputs its specification holds</td>
<td>find some set of nodes s.t., it is a clique and for all possible other sets of nodes not one is a larger clique</td>
</tr>
<tr>
<td>some program: ASTNode</td>
<td>some clq: set Node</td>
</tr>
<tr>
<td>all env: Var -&gt; Val</td>
<td>clique[clq] and</td>
</tr>
<tr>
<td>spec[program, env]</td>
<td>all ns: set Node</td>
</tr>
<tr>
<td></td>
<td>not (clique[ns] and #ns &gt; #clq)</td>
</tr>
</tbody>
</table>
## Solving maxClique Vs. Program Synthesis

<table>
<thead>
<tr>
<th></th>
<th><strong>program synthesis</strong></th>
<th><strong>maxClique</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>find <em>some</em> program AST s.t.,</td>
<td>find <em>some</em> set of nodes s.t., it is a clique and</td>
</tr>
<tr>
<td></td>
<td>for <em>all</em> possible values of its inputs</td>
<td>for <em>all</em> possible other sets of nodes</td>
</tr>
<tr>
<td></td>
<td>its specification holds</td>
<td>not one is a larger clique</td>
</tr>
<tr>
<td></td>
<td><em>some</em> program: ASTNode</td>
<td><em>some</em> clq: set Node</td>
</tr>
<tr>
<td></td>
<td><em>all</em> env: Var -&gt; Val</td>
<td>_clique[clq] and</td>
</tr>
<tr>
<td></td>
<td><em>spec[program, env]</em></td>
<td>_all ns: set Node</td>
</tr>
<tr>
<td></td>
<td></td>
<td>_not (clique[ns] and #ns &gt; #clq)</td>
</tr>
</tbody>
</table>

### Similarities:
- The same _some/all (∃∀) pattern_
- The _all_ quantifier is higher-order
### Solving \textit{maxClique} Vs. Program \textit{Synthesis}

<table>
<thead>
<tr>
<th><strong>program synthesis</strong></th>
<th><strong>maxClique</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>find \textbf{some} program AST s.t., for all possible values of its inputs its specification holds</td>
<td>find \textbf{some} set of nodes s.t., it is a clique and for all possible other sets of nodes not one is a larger clique</td>
</tr>
<tr>
<td>\textbf{some} program: ASTNode</td>
<td>\textbf{some} clq: set Node</td>
</tr>
<tr>
<td>\textbf{all} env: Var -&gt; Val</td>
<td>clique[clq] and</td>
</tr>
<tr>
<td>spec[program, env]</td>
<td>all ns: set Node</td>
</tr>
<tr>
<td></td>
<td>not (clique[ns] and #ns &gt; #clq)</td>
</tr>
</tbody>
</table>

**similarities:**
- the same \textbf{some/all} (\exists\forall) pattern
- the \textbf{all} quantifier is higher-order

**how do existing program synthesizers work?**
original synthesis formulation

\[
\text{run } \{ \text{some } \text{prog: ASTNode} \mid \text{all } \text{env: Var }\rightarrow \text{Val} \mid \text{spec}[\text{prog}, \text{env}] \}\]

Counter-Example Guided Inductive Synthesis [Solar-Lezama, ASPLOS'06]
original synthesis formulation

\[
\text{run } \{ \text{some prog: ASTNode} \mid \text{all env: Var } \rightarrow \text{Val} \mid \text{spec[prog, env]} \}
\]

Counter-Example Guided Inductive Synthesis [Solar-Lezama, ASPLOS'06]

1. search: find some program and some environment s.t. the spec holds, i.e.,

\[
\text{run } \{ \text{some prog: ASTNode} \mid \text{some env: Var } \rightarrow \text{Val} \mid \text{spec[prog, env]} \}
\]

to get a concrete candidate program $\text{prog}$
CEGIS: A Common Approach for Program Synthesis

original synthesis formulation

\[
\text{run } \{ \text{some prog: ASTNode} \mid \text{all env: Var -> Val} \mid \text{spec}[\text{prog, env}] \} 
\]

Counter-Example Guided Inductive Synthesis [Solar-Lezama, ASPLOS'06]

1. search: find some program and some environment s.t. the spec holds, i.e.,
   \[
   \text{run } \{ \text{some prog: ASTNode} \mid \text{some env: Var -> Val} \mid \text{spec}[\text{prog, env}] \} 
   \]
to get a concrete candidate program $prog$

2. verification: check if $prog$ holds for all possible environments:
   \[
   \text{check } \{ \text{all env: Var -> Val} \mid \text{spec}[\text{$prog$, env}] \} 
   \]
Done if verified; else, a concrete counterexample $env$ is returned as witness.
CEGIS: A Common Approach for Program Synthesis

original synthesis formulation

\[
\text{run } \{ \text{some } \text{prog: ASTNode} \mid \text{all } \text{env: Var }\rightarrow\text{Val} \mid \text{spec[prog, env]} \} \]

Counter-Example Guided Inductive Synthesis [Solar-Lezama, ASPLOS’06]

1. search: find some program and some environment s.t. the spec holds, i.e.,
   \[
   \text{run } \{ \text{some } \text{prog: ASTNode} \mid \text{some } \text{env: Var }\rightarrow\text{Val} \mid \text{spec[prog, env]} \} \\
   \]
   to get a concrete candidate program \$prog$

2. verification: check if \$prog$ holds for all possible environments:
   \[
   \text{check } \{ \text{all } \text{env: Var }\rightarrow\text{Val} \mid \text{spec[$prog, env$]} \} \\
   \]
   Done if verified; else, a concrete counterexample \$env$ is returned as witness.

3. induction: incrementally find a new program that additionally satisfies \$env$:
   \[
   \text{run } \{ \text{some } \text{prog: ASTNode} \mid \\
   \text{some } \text{env: Var }\rightarrow\text{Val} \mid \text{spec[prog, env]} \text{ and spec[prog, }$env$] \} \\
   \]
   If UNSAT, return no solution; else, go to 2.
**ALLOY**

**ALLOY** key insight

CEGIS can be applied to solve **arbitrary higher-order** formulas.
generality

- solve arbitrary higher-order formulas
- no domain-specific knowledge needed
**generality**

- solve *arbitrary* higher-order formulas
- no *domain-specific* knowledge needed

**implementability**

- key solver features for *efficient* implementation:
  - *partial instances*
  - *incremental solving*
**ALLOY**

**generality**
- solve arbitrary higher-order formulas
- no domain-specific knowledge needed

**implementability**
- key solver features for efficient implementation:
  - partial instances
  - incremental solving

**wide applicability** (in contrast to specialized synthesizers)
- program synthesis: SyGuS benchmarks
- security policy synthesis: Margrave
- solving graph problems: max-cut, max-clique, min-vertex-cover
- bounded verification: Turán’s theorem
Generality: Nested Higher-Order Quantifiers

```haskell
fun keysum[nodes: set Node]: Int {
    sum n: nodes | n.key
}

pred maxMaxClique[edges: Node->Node, clq: set Node] {
    maxClique[edges, clq]
    all ns: set Node |
    not (maxClique[edges, clq2] and
         keysum[ns] > keysum[clq])
}

run maxMaxClique for 5
```
Semantics: General Idea

- CEGIS: defined only for a single idiom (the $\exists \forall$ formula pattern)
Semantics: General Idea

- CEGIS: defined only for a single idiom (the $\exists\forall$ formula pattern)
- ALLOY*: generalized to arbitrary formulas
Semantics: General Idea

- CEGIS: defined only for a single idiom (the $\exists \forall$ formula pattern)
- ALLOY*: generalized to arbitrary formulas
  1. perform standard transformation: NNF and skolemization
Semantics: General Idea

- **CEGIS**: defined only for a single idiom (the $\exists \forall$ formula pattern)
- **ALLOY**: generalized to arbitrary formulas
  1. perform standard transformation: NNF and skolemization
  2. decompose arbitrary formula into known idioms
     - $\text{FOL}$: first-order formula
     - $\text{OR}$: disjunction
     - $\exists \forall$: higher-order top-level $\forall$ quantifier (not skolemizable)
Semantics: General Idea

- **CEGIS**: defined only for a single idiom (the $\exists \forall$ formula pattern)
- **ALLOY\*:** generalized to arbitrary formulas

1. perform standard transformation: NNF and skolemization

2. decompose arbitrary formula into known idioms
   - $\text{FOL}$ : first-order formula
   - $\text{OR}$ : disjunction
   - $\exists \forall$ : higher-order top-level $\forall$ quantifier (not skolemizable)

3. solve using the following decision procedure
   - $\text{FOL}$ : solve directly with Kodkod (first-order relational solver)
   - $\text{OR}$ : solve each disjunct separately
   - $\exists \forall$ : apply CEGIS
**Caveats**

\[
\text{some } \text{prog}: \text{Node} \mid \text{acyclic}[\text{prog}]
\]

\[
\text{all } \text{eval}: \text{Node} \to (\text{Int} + \text{Bool}) \mid \text{semantics}[\text{eval}] \text{ implies spec}[\text{prog}, \text{eval}]
\]

\[
\exists (\text{conj}: \text{prog in Node and acyclic}[$\text{prog}$], \\
\text{eQuant}: \text{some eval ...}, \\
\text{aQuant}: \text{all eval} ...)
\]
Some prog: Node | acyclic[prog]
all eval: Node -> (Int+Bool) | semantics[eval] implies spec[prog, eval]

\[ \exists (\text{conj}: \ prog \ in \ Node \ and \ acyclic[\ prog], \ eQuant: \ some \ eval \ ... , \ aQuant: \ all \ eval \ ...) \]

1. candidate search

- solve \( \text{conj} \land eQuant \)
- \( \rightarrow \) candidate instance \( \$cand: \) values of all relations except \( eQuant.var \)
A\textsc{LLOY}\textsuperscript{*} Implementation \textbf{Caveats}

\texttt{some prog: Node | acyclic[prog]}
\texttt{all eval: Node -> (Int+Bool) | semantics[eval] implies spec[prog, eval]}
\Rightarrow \exists (\textit{conj}: \ prog \ in \ Node \ and \ acyclic[\ prog],
\textit{eQuant}: \ some \ eval \ ...,
\textit{aQuant}: \ all \ eval \ ...)

1. candidate search

\textbullet\ solve \ \textit{conj} \ \land \ \textit{eQuant} \\
\Rightarrow \ \textit{candidate instance} \ $\texttt{cand}$: \ values \ of \ all \ relations \ except \ \textit{eQuant}.\texttt{var}$

2. verification

\textbullet\ solve \ \lnot \ \textit{aQuant} \ \text{against the} \ \texttt{cand} \ \textit{partial instance} \\
\Rightarrow \ \textit{counterexample} \ $\texttt{cex}$: \ value \ of \ the \ \textit{eQuant}.\texttt{var} \ relation
\textbf{ALLOY* Implementation Caveats}

\begin{verbatim}
some prog: Node | acyclic[prog]  
all eval: Node -> (Int+Bool) |  
semantics[eval] implies spec[prog, eval] 
\end{verbatim}

\begin{itemize}
  \item \textbf{1. candidate search}
    \begin{itemize}
      \item solve \textit{conj} \land \textit{eQuant}
      \begin{align*}
        \text{\textit{candidate instance}} \quad \textit{$cand$}: & \text{values of all relations except } \textit{eQuant}.\text{var} \\
      \end{align*}
    \end{itemize}
  \item \textbf{2. verification}
    \begin{itemize}
      \item solve $\neg \textit{aQuant}$ against the $\textit{cand}$ \textit{partial instance}
      \begin{align*}
        \text{\textit{counterexample}} \quad \textit{$cex$}: & \text{value of the } \textit{eQuant}.\text{var} \text{ relation} \\
      \end{align*}
    \end{itemize}
\end{itemize}

\textbf{partial instance}
\begin{itemize}
  \item partial solution known upfront
  \item enforced using \textit{bounds}
\end{itemize}
**Caveats**

some prog: Node |  
acyclic[prog]  
all eval: Node -> (Int+Bool) |  
semantics[eval] implies spec[prog, eval]  

$\exists(\text{conj}: \text{prog in Node and acyclic[prog]}, \text{eQuant}: \text{some eval} \ldots, \text{aQuant}: \text{all eval} \ldots)$

1. **candidate search**

- solve $\text{conj} \land \text{eQuant}$
- $\rightarrow$ candidate instance $\$\text{cand}$: values of all relations except $\text{eQuant}.\text{var}$

2. **verification**

- solve $\neg \text{aQuant}$ against the $\$\text{cand}$ partial instance
- $\rightarrow$ counterexample $\$\text{cex}$: value of the $\text{eQuant}.\text{var}$ relation

3. **induction**

- use incremental solving to add
  - replace $\text{eQuant}.\text{var}$ with $\$\text{cex}$ in $\text{eQuant}.\text{body}$
  - to previous search condition
ALLOY* Implementation Caveats

some prog: Node | 
acyclic[ prog ]

all eval: Node -> (Int+Bool) | 
semantics[ eval ] implies spec[ prog, eval ]

⇒ ∃( conj: $ prog in Node and acyclic[ $ prog ],

  eQuant: some eval ..., 

  aQuant: all eval ...) 

1. candidate search

  • solve $ conj ∧ eQuant$

  ⇒ candidate instance $ cand: values of all relations except $ eQuant.var $

2. verification

  • solve ¬$ aQuant$ against the $ cand partial instance$

  ⇒ counterexample $ cex: value of the $ eQuant.var relation$

3. induction

  • use incremental solving to add

    replace $ eQuant.var with $ cex in $ eQuant.body$

to previous search condition

  partial instance
  • partial solution known upfront
  • enforced using bounds

  incremental solving
  • continue from prev solver instance
  • the solver reuses learned clauses
**ALLOY\* Implementation Caveats**

\[\text{some prog: Node |} \]
\[\text{acyclic[\text{prog}] \rightarrow} \]
\[\text{all eval: Node -> (Int+Bool) |} \]
\[\text{semantics[eval] implies spec[prog, eval]} \rightarrow \]
\[\exists (\text{conj: \ prog in Node and acyclic[\text{prog}]}, \]
\[\text{\ eQuant: some eval ...}, \]
\[\text{\ aQuant: all eval ...})} \]

1. **candidate search**
   - solve \( \text{conj} \land \text{eQuant} \)
   - \( \rightarrow \) \text{candidate instance} \( \text{$cand$}: \) values of all relations except \( \text{eQuant}.\text{var} \)

2. **verification**
   - solve \( \neg \text{aQuant} \) against the \( \text{$cand$} \) \text{partial instance}
   - \( \rightarrow \) \text{counterexample} \( \text{$cex$}: \) value of the \( \text{eQuant}.\text{var} \) relation

3. **induction**
   - use \text{incremental solving} to add
     - replace \( \text{eQuant}.\text{var} \) with \( \text{$cex$} \) in \( \text{eQuant}.\text{body} \)
     - \text{to previous search condition}

? **what if the increment formula is not first-order**
   - optimization 1: use its weaker “first-order version”

---

**partial instance**
- partial solution known upfront
- enforced using \textit{bounds}

**incremental solving**
- continue from prev solver instance
- the solver reuses learned clauses
2. domain constraints

“for all possible eval, if the semantics hold then the spec must hold” vs. “for all eval that satisfy the semantics, the spec must hold”
2. domain constraints

"for all possible eval, if the semantics hold then the spec must hold" vs. "for all eval that satisfy the semantics, the spec must hold"

- logically equivalent, but, when "for" implemented as CEGIS:
2. domain constraints

"for all possible eval, if the semantics hold then the spec must hold" vs. "for all eval that satisfy the semantics, the spec must hold"

- logically equivalent, but, when "for" implemented as CEGIS:

```alloy
pred synth[prog: Node] {
  all eval: Node -> (Int+Bool) |
  semantics[eval] implies spec[prog, eval]
}

candidate search

some prog: Node |
  some eval: Node -> (Int+Bool) |
  semantics[eval] implies spec[prog, eval]

↓

a valid candidate doesn’t have to satisfy the semantics predicate!
```
2. domain constraints

"for all possible eval, if the semantics hold then the spec must hold" vs. "for all eval that satisfy the semantics, the spec must hold"

- Logically equivalent, but, when "for" implemented as CEGIS:

\[
\text{pred synth[prog: Node] \{}
\begin{align*}
    & \text{all eval: Node } \rightarrow \text{ (Int+Bool)} | \\
    & \text{ semantics[eval] implies spec[prog, eval]}
\end{align*}
\text{\}} \quad \downarrow \quad \text{candidate search}
\]

\[
\text{some prog: Node |}
\begin{align*}
    & \text{some eval: Node } \rightarrow \text{ (Int+Bool)} | \\
    & \text{ semantics[eval] implies spec[prog, eval]}
\end{align*}
\]

- A valid candidate doesn't have to satisfy the semantics predicate!

\[
\text{pred synth[prog: Node] \{}
\begin{align*}
    & \text{all eval: Node } \rightarrow \text{ (Int+Bool)} | \\
    & \text{ semantics[eval] implies spec[prog, eval]}
\end{align*}
\text{\}} \quad \downarrow \quad \text{candidate search}
\]

\[
\text{some prog: Node |}
\begin{align*}
    & \text{some eval: Node } \rightarrow \text{ (Int+Bool)} | \\
    & \text{ semantics[eval] implies spec[prog, eval]}
\end{align*}
\]

- A valid candidate must satisfy the semantics predicate!
evaluation goals
evaluation goals

1. scalability on classical higher-order graph problems
   ? does ALLOY* scale beyond “toy-sized” graphs
evaluation goals

1. scalability on classical higher-order graph problems
   - does ALLOY* scale beyond “toy-sized” graphs

2. applicability to program synthesis
   - expressiveness: how many SyGuS benchmarks can be written in ALLOY*
   - power: how many SyGuS benchmarks can be solved with ALLOY*
   - scalability: how does ALLOY* compare to other synthesizers
ALLOY* Evaluation

evaluation goals

1. scalability on classical higher-order graph problems
   ? does ALLOY* scale beyond “toy-sized” graphs

2. applicability to program synthesis
   ? expressiveness: how many SyGuS benchmarks can be written in ALLOY*
   ? power: how many SyGuS benchmarks can be solved with ALLOY*
   ? scalability: how does ALLOY* compare to other synthesizers

3. benefits of the two optimizations
   ? do ALLOY* optimizations improve overall solving times
Evaluation: **Graph Algorithms**

![Graph Algorithms Evaluation Graph](image-url)

- **x-axis**: Number of nodes (# Nodes)
- **y-axis**: Solving time (s)

Lines represent:
- **Blue**: Max clique
- **Red**: Max cut
- **Green**: Max independent set
- **Purple**: Min vertex cover
**Evaluation: Program Synthesis**

**expressiveness**
- We extended Alloy to support bit vectors
- We encoded 123/173 benchmarks, i.e., all except “ICFP problems”
  - Reason for skipping ICFP: 64-bit bit vectors (not supported by Kodkod)
  - (Aside) Not one of them was solved by any of the competition solvers

**power**
- **Alloy**\* was able to solve all different categories of benchmarks
  - Integer benchmarks, bit vector benchmarks, let constructs, synthesizing multiple functions at once, multiple applications of the synthesized function

**scalability**
- Many of the 123 benchmarks are either too easy or too difficult
  → Not suitable for scalability comparison
- We primarily used the integer benchmarks
- We also picked a few bit vector benchmarks that were too hard for all solvers
Evaluation: Program Synthesis

scalability comparison (integer benchmarks)

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Alloy*</th>
<th>Enumerative</th>
<th>Stochastic</th>
<th>Symbolic</th>
<th>Sketch</th>
</tr>
</thead>
<tbody>
<tr>
<td>max-2</td>
<td>0.01</td>
<td>0.1</td>
<td>1</td>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td>max-3</td>
<td>0.1</td>
<td>1</td>
<td>10</td>
<td>100</td>
<td>1000</td>
</tr>
<tr>
<td>max-4</td>
<td>0.01</td>
<td>0.1</td>
<td>1</td>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td>max-5</td>
<td>0.1</td>
<td>1</td>
<td>10</td>
<td>100</td>
<td>1000</td>
</tr>
<tr>
<td>array-2</td>
<td>0.01</td>
<td>0.1</td>
<td>1</td>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td>array-3</td>
<td>0.1</td>
<td>1</td>
<td>10</td>
<td>100</td>
<td>1000</td>
</tr>
<tr>
<td>array-4</td>
<td>0.01</td>
<td>0.1</td>
<td>1</td>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td>array-5</td>
<td>0.1</td>
<td>1</td>
<td>10</td>
<td>100</td>
<td>1000</td>
</tr>
</tbody>
</table>

benchmarks – parity-AIG-d1: full parity circuit using AND and NOT gates
- parity-NAND-d1: full parity circuit using AND always followed by NOT

Custom tweaks in Alloy*: create and use a single type of gate
- impose partial ordering between gates

parity-AIG-d1
- sigAIG extends BoolNode {left, right: one BoolNode; invLhs, invRhs, invOut: one Bool;}
- pred aig_semantics[eval: Node->(Int+Bool)] {all n: AIG | eval[n] = ((eval[n.left] ^ n.invLhs) && (eval[n.right] ^ n.invRhs)) ^ n.invOut}
- run synth for 0 but -1..0 Int, exactly 15 AIG solving time w/ partial ordering: 20s
- solving time w/o partial ordering: 80s

parity-NAND-d1
- sigNAND extends BoolNode {left, right: one BoolNode;}
- pred nand_semantics[eval: Node->(Int+Bool)] {all n: NAND | eval[n] = !(eval[n.left] && eval[n.right])}
- run synth for 0 but -1..0 Int, exactly 23 NAND solving time w/ partial ordering: 30s
- solving time w/o partial ordering: ∞
scalability comparison (select bit vector benchmarks)

- parity-AIG-d1: full parity circuit using AND and NOT gates
- parity-NAND-d1: full parity circuit using AND always followed by NOT
scability comparison (select bit vector benchmarks)

- benchmarks
  - parity-AIG-d1: full parity circuit using AND and NOT gates
  - parity-NAND-d1: full parity circuit using AND always followed by NOT

- all solvers (including ALLOY*) time out on both (limit: 1000s)
Evaluation: Program Synthesis

**scalability comparison** (select bit vector benchmarks)

- benchmarks
  - parity-AIG-d1: full parity circuit using AND and NOT gates
  - parity-NAND-d1: full parity circuit using AND always followed by NOT

- all solvers (including ALLOY*) time out on both (limit: 1000s)

- custom tweaks in ALLOY* synthesis models:
  - create and use a single type of gate
  - impose partial ordering between gates
scalability comparison (select bit vector benchmarks)

- parity-AIG-d1: full parity circuit using AND and NOT gates
- parity-NAND-d1: full parity circuit using AND always followed by NOT

all solvers (including ALLOY*) time out on both (limit: 1000s)

custom tweaks in ALLOY* synthesis models:
- create and use a single type of gate
- impose partial ordering between gates

<table>
<thead>
<tr>
<th>parity-AIG-d1</th>
<th>parity-NAND-d1</th>
</tr>
</thead>
</table>
| **sig AIG extends** BoolNode {    
  left, right: **one** BoolNode    
  invLhs, invRhs, invOut: **one** Bool    
} | **sig NAND extends** BoolNode {    
  left, right: **one** BoolNode    
} |
| **pred aig_semantics**[eval: Node->(Int+Bool)] {    
  all n: AIG    
  eval[n] = ((eval[n.left] ^ n.invLhs) &&    
  (eval[n.right] ^ n.invRhs)    
  ) ^ n.invOut} | **pred nand_semantics**[eval: Node->(Int+Bool)] {    
  all n: NAND    
  eval[n] = !(eval[n.left] &&    
  eval[n.right])    
} |
| **run synth for 0 but -1..0 Int, exactly 15 AIG** | **run synth for 0 but -1..0 Int, exactly 23 NAND** |
**Evaluation: Program Synthesis**

**scalability comparison** (select bit vector benchmarks)

- **benchmarks**
  - **parity-AIG-d1**: full parity circuit using AND and NOT gates
  - **parity-NAND-d1**: full parity circuit using AND always followed by NOT

- **all solvers (including ALLOY*) time out on both** (limit: 1000s)

- **custom tweaks in ALLOY* synthesis models:**
  - create and use a single type of gate
  - impose partial ordering between gates

<table>
<thead>
<tr>
<th>parity-AIG-d1</th>
<th>parity-NAND-d1</th>
</tr>
</thead>
</table>
| **sig** AIG extends BoolNode {
  left, right: one BoolNode
  invLhs, invRhs, invOut: one Bool
} | **sig** NAND extends BoolNode {
  left, right: one BoolNode
} |
| **pred** aig_semantics[eval: Node->(Int+Bool)] { |
  all n: AIG |
  eval[n] = ((eval[n.left] ^ n.invLhs) &&
  (eval[n.right] ^ n.invRhs)) ^ n.invOut} | **pred** nand_semantics[eval: Node->(Int+Bool)] { |
  all n: NAND |
  eval[n] = !(eval[n.left] &&
  eval[n.right])} |
| **run** synth for 0 but -1..0 Int, exactly 15 AIG | **run** synth for 0 but -1..0 Int, exactly 23 NAND |

- solving time w/ partial ordering: 20s
- solving time w/o partial ordering: 80s
- solving time w/ partial ordering: 30s
- solving time w/o partial ordering: ∞
## Evaluation: Benefits of ALLOY* Optimizations

<table>
<thead>
<tr>
<th></th>
<th>base</th>
<th>w/ optimizations</th>
</tr>
</thead>
<tbody>
<tr>
<td>max2</td>
<td>0.4s</td>
<td>0.3s</td>
</tr>
<tr>
<td>max3</td>
<td>7.6s</td>
<td>0.9s</td>
</tr>
<tr>
<td>max4</td>
<td>t/o</td>
<td>1.5s</td>
</tr>
<tr>
<td>max5</td>
<td>t/o</td>
<td>4.2s</td>
</tr>
<tr>
<td>max6</td>
<td>t/o</td>
<td>16.3s</td>
</tr>
<tr>
<td>max7</td>
<td>t/o</td>
<td>163.6s</td>
</tr>
<tr>
<td>max8</td>
<td>t/o</td>
<td>987.3s</td>
</tr>
<tr>
<td>array-search2</td>
<td>140.0s</td>
<td>1.6s</td>
</tr>
<tr>
<td>array-search3</td>
<td>t/o</td>
<td>4.0s</td>
</tr>
<tr>
<td>array-search4</td>
<td>t/o</td>
<td>16.1s</td>
</tr>
<tr>
<td>array-search5</td>
<td>t/o</td>
<td>485.6s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>base</th>
<th>w/ optimizations</th>
</tr>
</thead>
<tbody>
<tr>
<td>turan5</td>
<td>3.5s</td>
<td>0.5s</td>
</tr>
<tr>
<td>turan6</td>
<td>12.8s</td>
<td>2.1s</td>
</tr>
<tr>
<td>turan7</td>
<td>235.0s</td>
<td>3.8s</td>
</tr>
<tr>
<td>turan8</td>
<td>t/o</td>
<td>15.0s</td>
</tr>
<tr>
<td>turan9</td>
<td>t/o</td>
<td>45.0s</td>
</tr>
<tr>
<td>turan10</td>
<td>t/o</td>
<td>168.0s</td>
</tr>
</tbody>
</table>
**Conclusion**

**Alloy** is a general purpose constraint solver capable of efficiently solving arbitrary higher-order formulas sound & complete within given bounds.

Higher-order and Alloy historically bit-blasting higher-order quantifiers: attempted, deemed intractable previously many ad hoc mods to Alloy:
- Aluminum, Razor, staged execution...

Why is this important?
- Accessible to wider audience, encourages new applications
- Potential impact: abundance of tools that build on Alloy/Kodkod, for testing, program analysis, security, bounded verification, executable specifications...
**Alloy**

**Conclusion**

**Alloy** is
- general purpose constraint solver
- capable of efficiently solving **arbitrary higher-order** formulas
- **sound & complete** within given bounds

**Higher-order and Alloy historically**
- bit-blasting higher-order quantifiers: attempted, deemed intractable
- previously many ad hoc mods to alloy
  - aluminum, razor, staged execution, ...
**Conclusion**

**Alloy** is:
- *general* purpose constraint solver
- capable of efficiently solving *arbitrary higher-order* formulas
- *sound & complete* within given bounds

**Higher-order and alloy historically**
- bit-blasting higher-order quantifiers: attempted, deemed intractable
- previously many ad hoc mods to alloy
  - aluminum, razor, staged execution, ...

**Why is this important?**
- accessible to wider audience, encourages new applications
- potential *impact*
  - abundance of tools that build on Alloy/Kodkod, for testing, program analysis, security, bounded verification, executable specifications, ...
**SUNNY**: Model-Based Reactive Web Framework

(my previous work)

- executable specs for java
- program synthesis

**spec engine apps**
- formal logic
- sophisticated search
- complex algorithms, constraint solving

---

- more powerful constraint solver
- capable of solving a whole new category of formal specifications

---

**spec engine apps**
- DSL translation/compilation
- domain-specific uses

---

- unified specification & implementation language

---

- model-based web framework
- reactive, single-tier, policy-agnostic
- what instead of how

---

- [ABZ’12, SCP’14, ICSE’15]
- [ABZ’14]
- [Onward’13]
A simple web app: Sunny IRC

custom-tailored internet chat relay app
A simple web app: **SUNNY IRC**

custom-tailed internet chat relay app
A simple web app: **SUNNY IRC**

custom-tailored internet chat relay app
Conceptually **simple**, **but** in practice...
Conceptually **simple, but** in practice...

- **distributed system**
  - concurrency issues
  - keeping everyone updated
Conceptually **simple**, but in practice...

- **distributed system**
  - concurrency issues
  - keeping everyone updated

- **heterogeneous environment**
  - rails + javascript + ajax + jquery + ...
  - html + erb + css + sass + scss + bootstrap + ...
  - db + schema + server config + routes + ...
Conceptually **simple, but** in practice...

- **distributed system**
  - concurrency issues
  - keeping everyone updated

- **heterogeneous environment**
  - rails + javascript + ajax + jquery + ...
  - html + erb + css + sass + scss + bootstrap + ...
  - db + schema + server config + routes + ...

- **abstraction gap**
  - high-level problem domain
  - low-level implementation level
Conceptually **simple, but** in practice...

- **distributed system**
  - concurrency issues
  - keeping everyone updated

- **heterogeneous environment**
  - rails + javascript + ajax + jquery + ...
  - html + erb + css + sass + scss + bootstrap + ...
  - db + schema + server config + routes + ...

- **abstraction gap**
  - high-level problem domain
  - low-level implementation level
MDD: how far can it get us?

exercise:

sketch out a model (design, spec) for the Sunny IRC application
**Sunny IRC: data model**

- **user class User**
  
  `# inherited: name, email: Text`
  
  `salute: () -> "Hi #{this.name}"`

- **record class Msg**
  
  `text: Text`
  
  `sender: User`
  
  `time: Val`

- **record class ChatRoom**
  
  `name: Text`
  
  `members: set User`
  
  `messages: compose set Msg`

- **record**: automatically persisted objects with typed fields
- **user**: special kind of record, assumes certain fields, auth, etc.
- **set**: denotes non-scalar (set) type
- **compose**: denotes ownership, deletion propagation, etc.
Sunny IRC: **machine model**

- **client** class: Client
  - user: User
- **server** class: Server
  - rooms: compose set ChatRoom

- **client**: special kind of record, used to represent client machines
- **server**: special kind of record, used to represent the server machine
event class SendMsg
from: client: Client
to: server: Server

params:
  room: ChatRoom
  msgText: Text

requires: () ->
  return "must log in!" unless this.client?.user
  return "must join room!" unless this.room?.members.contains(this.client.user)

ensures: () ->
  this.room.messages.push Msg.create(sender: this.client.user
text: this.msgText
time: Date.now())
Modeling done. **What next?**

**challenge**

how to **make the most** of this **model**?
Modeling done. **What next?**

**challenge**

how to make the most of this model?

**goal**

make the model **executable** as much as possible!
Traditional **MVC** Approach

- write a matching DB schema
- turn each record into a resource (model class)
- turn each event into a controller and implement the CRUD operations
- configure URL routes for each resource

**Aesthetics:**
- design and implement a nice looking HTML/CSS presentation
- make it interactive:
  - decide how to implement server push
  - keep track of who's viewing what
  - monitor resource accesses
  - push changes to clients when resources are modified
- implement client-side Javascript to accept pushed changes and dynamically update the DOM
Traditional **MVC** Approach

- boilerplate:
  - write a matching **DB schema**
  - turn each record into a **resource** (model class)
  - turn each event into a **controller** and implement the CRUD operations
  - configure URL **routes** for each resource
Traditional **MVC** Approach

- **boilerplate:**
  - write a matching **DB schema**
  - turn each record into a **resource** (model class)
  - turn each event into a **controller** and implement the CRUD operations
  - configure URL **routes** for each resource

- **aesthetics:**
  - design and implement a nice looking **HTML/CSS presentation**
Traditional **MVC** Approach

- **boilerplate:**
  - write a matching **DB schema**
  - turn each record into a **resource** (model class)
  - turn each event into a **controller** and implement the CRUD operations
  - configure URL **routes** for each resource

- **aesthetics:**
  - design and implement a nice looking **HTML/CSS presentation**

- **to make it interactive:**
  - decide how to implement **server push**
  - keep **track** of who’s **viewing** what
  - **monitor** resource **accesses**
  - push changes to clients when resources are modified
  - implement client-side Javascript to accept pushed changes and dynamically update the **DOM**
Traditional **MVC** Approach

- **Boilerplate:**
  - write a matching DB schema
  - turn each record into a resource (model class)
  - turn each event into a controller and implement the CRUD operations
  - configure URL routes for each resource

- **Aesthetics:**
  - design and implement a nice looking HTML/CSS presentation

- **To make it interactive:**
  - decide how to implement server push
  - keep track of who’s viewing what
  - monitor resource accesses
  - push changes to clients when resources are modified
  - implement client-side Javascript to accept pushed changes and dynamically update the DOM
demo: responsive GUI without messing with javascript
GUIs in Sunny: **dynamic templates**

- like standard templating engine with data bindings
- automatically re-rendered when the model changes
GUIs in SUNNY: **dynamic templates**

- like standard **templating engine with data bindings**
- automatically **re-rendered** when the model changes

```html
online_users.html

```<div>
{{#each Server.onlineClients.user}}
  {{> user_tpl user=this}}
{{/each}}
</div>
```
GUIs in **SUNNY**: **binding to events**

```html
<div>
  <div>
    <input type="text" name="text" {\{SendMsg_msgText\}} {\{sunny_trigger\}} />
  </div>
  <button {\{sunny_trigger\}}>
    Send
  </button>
</div>
```

- html5 data attributes specify event type and parameters
- dynamically discovered and triggered asynchronously
- no need for any Ajax requests/responses
- the data-binding mechanism will automatically kick in
GUIs in Sunny: **binding to events**

**room_tpl.html**

```html
<div>{{SendMsg room=this.room}}
  
  <div>
    <input type="text" name="text"
      placeholder="Enter message"
      {{SendMsg_msgText}}
      {{sunny_trigger}} />
  </div>

  <button>{{sunny_trigger}}>Send</button>

</div>
```

Data attributes specify event type and parameters dynamically discovered and triggered asynchronously. No need for any Ajax requests/responses – the data-binding mechanism will automatically kick in.
html5 data attributes specify event type and parameters

dynamically discovered and triggered asynchronously

no need for any Ajax requests/responses

- the data-binding mechanism will automatically kick in
Adding New Features: **adding a field**

implement user status messages
Adding New Features: **adding a field**

implement user status messages

- **all it takes:**

```jsx
user class User

status: Text

<p>{{editableField obj=this.user fld="status"}}
  {{this.user.status}}
</p>
```
Adding New Features: adding a field

implement user status messages

- all it takes:

```html
user class User
  status: Text

<p>{{editableField obj=this.user fld="status"}}</p>

{{this.user.status}}
</p>
```

demo
Security/Privacy: **write** policies

**forbid changing other people’s data**
- by default, all fields are public
- **policies** used to specify access restrictions
Security/Privacy: **write** policies

**forbid changing other people’s data**
- by default, all fields are public
- **policies** used to specify access restrictions

```javascript
policy User,
  update:
    "*": (usr, val) ->
      return this.allow() if usr.equals(this.client?.user)
      return this.deny("can’t edit other people’s data")
```
Security/Privacy: **write** policies

**forbid changing other people’s data**
- by default, all fields are public
- **policies** used to specify access restrictions

```javascript
policy User,
  update:
    "*": (usr, val) ->
      return this.allow() if usr.equals(this.client?.user)
      return this.deny("can’t edit other people’s data")
```

- **declarative** and **independent** from the rest of the system
- automatically **checked** by the system at each **field access**
Security/Privacy: **read** & **find** policies

**hide avatars** unless the two users share a room
Security/Privacy: **read** & **find** policies

**hide avatars** unless the two users share a room

```javascript
policy User,
read:
  avatar: (usr) ->
    clntUser = this.client?.user
    return this.allow() if usr.equals(clntUser)
    if (this.server.rooms.some (room) -> room.members.containsAll([usr, clntUser]))
      return this.allow()
    else
      return this.deny()
```

**read denied** → empty value returned instead of raising exception
Security/Privacy: **read** & **find** policies

**hide avatars** unless the two users share a room

```javascript
policy User,
    read:
    avatar: (usr) ->
        clntUser = this.client?.user
        return this.allow() if usr.equals(clntUser)
        if (this.server.rooms.some (room)->room.members.containsAll([usr, clntUser]))
            return this.allow()
        else
            return this.deny()
```

- **read denied** → empty value returned instead of raising exception

**invisible users**: hide users whose status is “busy”
Security/Privacy: read & find policies

hide avatars unless the two users share a room

```javascript
policy User,
read:
  avatar: (usr) ->
    clntUser = this.client?.user
    return this.allow() if usr.equals(clntUser)
    if (this.server.rooms.some (room) -> room.members.containsAll([usr, clntUser]))
      return this.allow()
    else
      return this.deny()
```

- read denied → empty value returned instead of raising exception

invisible users: hide users whose status is “busy”

```javascript
policy User,
find: (users) ->
  clntUser = this.client?.user
  return this.allow(filter users, (u) -> u.equals(clntUser) || u.status != "busy")
```

- find policies → objects entirely removed from the client-view of the data
Demo: defining **access policies** independently

no GUI templates need to change!
Policy Checking in SUNNY

access control style

- policies attached to fields
- implicit principal: client which issued current request
- evaluate against the dynamic state of the program
- policy code executes in the current client context
  - circular dependencies resolved by allowing recursive operations
access control style

- policies attached to fields
- implicit principal: client which issued current request
- evaluate against the dynamic state of the program
- policy code executes in the current client context
  - circular dependencies resolved by allowing recursive operations

- policy execution creates reactive server-side dependencies

- Alice’s client doesn’t contain Bob’s status field at all
- nevertheless, it automatically reacts when Bob changes his status!
### Related Work: Reactive + Policies

<table>
<thead>
<tr>
<th>Checking Policies</th>
<th>Enforcing Policies</th>
<th>Reactive</th>
</tr>
</thead>
<tbody>
<tr>
<td>UI Frameworks (.NET, XAML, Backbone.js, AngularJS, ...)</td>
<td>Traditional IF (Resin, Jiff, Dytan, ...)</td>
<td>Reactive Web (Ur/Web, Elm, Flapjax, Meteor, ...)</td>
</tr>
</tbody>
</table>
## Related Work: Reactive + Policies

<table>
<thead>
<tr>
<th>UI Frameworks</th>
<th>checking policies</th>
<th>enforcing policies</th>
<th>reactive</th>
</tr>
</thead>
<tbody>
<tr>
<td>(.NET, XAML, Backbone.js, AngularJS, ...)</td>
<td>×</td>
<td>×</td>
<td>✔️</td>
</tr>
</tbody>
</table>
## Related Work: Reactive + Policies

<table>
<thead>
<tr>
<th></th>
<th>Checking Policies</th>
<th>Enforcing Policies</th>
<th>Reactive</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>UI Frameworks</strong></td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>(.NET, XAML, Backbone.js, AngularJS, ...)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Traditional IF</strong></td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>(Resin, Jiff, Dytan, ...)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Related Work: Reactive + Policies

<table>
<thead>
<tr>
<th></th>
<th>checking policies</th>
<th>enforcing policies</th>
<th>reactive</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>UI Frameworks</strong> (.NET, XAML, Backbone.js, AngularJS, ...)</td>
<td>✗</td>
<td>✗</td>
<td>✔️</td>
</tr>
<tr>
<td><strong>Traditional IF</strong> (Resin, Jiff, Dytan, ...)</td>
<td>✔️</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td><strong>Reactive Web</strong> (Ur/Web, Elm, Flapjax, Meteor, ...)</td>
<td>✔️</td>
<td>✗</td>
<td>✔️</td>
</tr>
</tbody>
</table>
## Related Work: Reactive + Policies

<table>
<thead>
<tr>
<th></th>
<th>checking policies</th>
<th>enforcing policies</th>
<th>reactive</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>UI Frameworks</strong></td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>(.NET, XAML, Backbone.js, AngularJS, ...)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Traditional IF</strong></td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>(Resin, Jiff, Dytan, ...)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Reactive Web</strong></td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>(Ur/Web, Elm, Flapjax, Meteor, ...)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Enforcing Policies</strong></td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>(Jeeves, Hails/LIO, ...)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Related Work: Reactive + Policies

<table>
<thead>
<tr>
<th></th>
<th>Checking Policies</th>
<th>Enforcing Policies</th>
<th>Reactive</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>UI Frameworks</strong></td>
<td>![X]</td>
<td>![X]</td>
<td>![✓]</td>
</tr>
<tr>
<td>(.NET, XAML, Backbone.js, AngularJS, ...)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Traditional IF</strong></td>
<td>![✓]</td>
<td>![X]</td>
<td>![X]</td>
</tr>
<tr>
<td>(Resin, Jiff, Dytan, ...)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Reactive Web</strong></td>
<td>![✓]</td>
<td>![X]</td>
<td>![✓]</td>
</tr>
<tr>
<td>(Ur/Web, Elm, Flapjax, Meteor, ...)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Enforcing Policies</strong></td>
<td>![✓]</td>
<td>![✓]</td>
<td>![X]</td>
</tr>
<tr>
<td>(Jeeves, Hails/LIO, ...)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Sunny</strong></td>
<td>![✓]</td>
<td>![✓]</td>
<td>![✓]</td>
</tr>
</tbody>
</table>
Example SUNNY Apps

gallery of applications

- internet relay chat
  + implement invisible users with policies

- party planner
  + intricate and interdependent policies for hiding sensitive data

- social network
  + highly customizable privacy settings

- photo sharing
  + similar to “social network”, but in the context of file sharing

- mvc todo
  + from single- to multi-user with policies
SUNNY: the big picture

Centralized unified model

Single-tier uncluttered focus on essentials: what the app should do

My contribution: functionality

Separation of main concerns: data, events, GUI, policies

Going forward:
- Optimizations: scalable/parallelizable back ends
- Clever data partitioning
- Declarative model-based cloud apps

Visualization:
- Flexible model-based GUI builder
- Generic & reusable widgets

Thank You!
SUNNY: the big picture

**declarative nature of SUNNY**
- centralized **unified** model
- **single-tier**
- uncluttered focus on **essentials**: **what** the app should do
**SUNNY:** the big picture

**declarative nature of SUNNY**
- centralized **unified** model
- **single-tier**
- uncluttered focus on **essentials:** what the app should do

**my contribution:** functionality
- separation of main concerns: data, events, GUI, policies
**SUNNY**: the big picture

**declarative nature of SUNNY**
- centralized *unified* model
- *single*-tier
- uncluttered focus on *essentials*: what the app should do

**my contribution**: functionality
- separation of main concerns: data, events, GUI, policies

**going forward:**
- optimizations
  - scalable/parallelizable back ends
  - clever data partitioning
  - declarative model-based cloud apps
- visualization
  - flexible model-based GUI builder
  - generic & reusable widgets
Acknowledgements
Acknowledgements

advisor

thesis committee

UROPs

doctoral students

co-authors/collaborators
**SUNNY**: the big picture

**declarative nature of SUNNY**
- centralized unified model
- single-tier
- uncluttered focus on essentials: what the app should do

**my contribution**: functionality
- separation of main concerns: data, events, GUI, policies

**going forward**: 
- optimizations
  - scalable/parallelizable back ends
  - clever data partitioning
  - declarative model-based cloud apps
- visualization
  - flexible model-based GUI builder
  - generic & reusable widgets

---

Thank You!