Part 1:
Engineering Domain-Specific Languages
With FORMULA 2.0

Ethan K. Jackson,
Research in Software Engineering,
Microsoft Research, Redmond, WA, USA
Outline

- Domain-specific languages and core use-cases
  How do these align with the goals of ExCAPE?

- Logic programming as a foundation
  Discuss relationships with other logics and problems with classical logic programming.

- Open-world logic programming for synthesis / verification
  Modify the closed-world assumption of classical LP. Allows for natural specifications of synthesis / verification problems.
Resources

http://formula.codeplex.com/

*Permissive source license (Microsoft Public License)
1.1 Why Domain-Specific Languages?

FORMULA 2.0

Ethan K. Jackson,
Research in Software Engineering,
Microsoft Research, Redmond, WA, USA
Hypothesis

- Precisely capture patterns and practices.
- And scalability of synthesis / verification can be improved.
- This applies to systems in general, not just software.
How to formalize DSLs?

Many ways to define abstractions; we shall use logic.

- Logic is precise.
- In many cases, computers can quickly reason about logical statements.
- With the right logic, many phenomena can be formalized.
A system for specifying DSLs with logic.

- Generic; not specifically designed to model software.
- Specifications are written as “open–world” logic programs.
- FORMULA 2.0 can verify, synthesize, transform, compile and check models all with logic.
Key Use Cases

- **Clean specification language for building abstractions**
  
  Modern language features to build compositions, transformations, etc... with static analysis to detect mistakes.

- **Model synthesis and design space exploration**
  
  Synthesize models satisfying complex constraints. Find many different models satisfying constraints.

- **Axiomatic compilers and verification**
  
  For DSLs were compilers are non-trivial, but whose compilation logic is not too complicated. Write small axiomatic compilers where verification is more scalable.
Some Links

P DSL for verifiable device drivers


Synthesis of biological models

Benjamin Hall, Ethan K. Jackson and Jasmin Fisher (MSR), Fast Analysis of Biological Models Using Open-World Logic Programming, POPL OBT, 2013. (Full paper in progress)

Diverse design space exploration


Reasoning about metamodeling frameworks

Ethan K. Jackson, Tihamer Levendovszky, and Daniel Balasubramanian: Reasoning about Metamodeling with Formal Specifications and Automatic Proofs, MODELS 2011
A type of pico satellite; standard form factor; ~$65K to launch

Underwater-gliders; at sea 100’s of days; ~$100K to build

Apple’s theorized watch; has array of biometric sensors
Figure from top medical reference drawn by a graphic artist; an informal model of a biological system.
Extend Powerpoint with biology abstraction

Shapes have a specific meaning. Can be used to build precise models and synthesize diagrams.
Add some colors to the shapes...
1.2
Formalizing With Logic Programs

FORMULA 2.0
Ethan K. Jackson,
Research in Software Engineering,
Microsoft Research, Redmond, WA, USA
A relation $R$ models $\varphi$ ($R \models \varphi$) if $\varphi$ evaluates to true under $R$.

Let $R = \{1, 2, 3, 4\}$

\[
R \models \forall x \in R. \left( x > 1 \Rightarrow \exists y \in R. x = y + 1 \right)
\]

\[
R \not\models \exists x \in R. \left( x > 1 \Rightarrow \forall y \in R. x = y + 1 \right)
\]
Is this transitive closure?

Claim: 
$E^* \subset N \times N$ is the transitive closure of $E \subset N \times N$

If: 
$\forall x \in E. x \in E^*$

$\forall x, y \in E^*. \pi_2(x) = \pi_1(y) \Rightarrow (\pi_1(x), \pi_2(y)) \in E^*$

Let $E = \{(1, 2), (2, 3)\}$

$\{(1,2), (2, 3), (1, 3)\} \models \varphi$

$\{(1,2), (2, 3), (1, 3), (1, 1), (2, 2)\} \models \varphi$

$\{(1,2), (1,3), (2,3), (3,2), (3,1), (2,1), (1,1), (2,2), (3,3)\} \models \varphi$
Least models

Models are ordered by subset inclusion. Transitive closure is the least model satisfying the predicate.

\{(1,2), (2, 3), (1, 3)\} \subset \{(1,2), (2, 3), (1, 3), (1, 1), (2, 2)\} \subset \{(1,2), (1, 3), (2, 3), (3, 2), (3, 1), (2, 1), (1, 1), (2, 2), (3, 3)\}
Logic Programs

A logic program is a way of writing logic so a least model always exists and can be computed.

E(1, 2).
E(2, 3).
E*(x, y) :- E(x, y).
E*(x, z) :- E*(x, y), E*(y, z).
Logical Meaning

Find the least model $K$ satisfying:

- $E(1, 2) \in K.$
- $E(2, 3) \in K.$
- $\forall E(x, y) \in K. E^*(x, y) \in K.$
- $\forall E^*(x_1, y_1), E^*(x_2, y_2) \in K. y_1 = x_2 \Rightarrow E^*(x_1, y_2) \in K.$

Least model is:

$\{ E(1,2), E(2,3), E^*(1,2), E^*(2,3), E^*(1,3) \}$
Initially let $K = \{\}$. Run the rules until nothing else happens.

1. $K = \{\}$
   - Run: $E(1, 2)$.

2. $K = \{E(1, 2)\}$
   - Run: $E(2, 3)$.

3. $K = \{E(1, 2), E(2, 3)\}$
   - Run: $E^*(x, y) :- E(x, y)$.

4. $K = \{E(1, 2), E(2, 3), E^*(1, 2), E^*(2, 3)\}$
   - Run: $E^*(x, z) :- E^*(x, y), E^*(y, z)$.

5. $K = \{E(1, 2), E(2, 3), E^*(1, 2), E^*(2, 3), E^*(1, 3)\}$
   - Stop: no rule can extend $K$ further.
Review

- Use logic for formal specifications
- Logic programs can be used to write axioms
- Logic programs mean least models, making them more expressive than first-order logic
- Logic programs are also programs; their execution computes least models
- Logic programs can answer queries about their least models
Still missing something...

Most programming languages have data types. What about logic programs?

- We add data types to logic programming
- Our data types are “algebraic”, i.e. they are functions that create data.
- A data constructor always constructs the same value when provided the same arguments
- Two values are the same iff they were constructed by the same constructor with the same arguments
Constructor Declaration

\[
E ::= (\text{src: Integer}, \text{dst: Integer}).
\]
Examples

- Constructs an E-value
  \[ E(1, 2) \]
- Error: tries to constructs an E-value with non-integers
  \[ E(0.5, \text{“hello”}) \]
- \( x \) must be an E-value constructed with identical arguments
  \[ x = E(y, y) \]
- If an \( E(x, y) \) value is in the least model, then \( E(y, x) \) is in the least model.
  \[ E(y, x) :\neg E(x, y). \]
Returning to transitive closure...

\[
\begin{align*}
E & ::= (\text{Natural, Natural}). \\
E^* & ::= (\text{Natural, Natural}). \\
E(1, 2). \\
E(2, 3). \\
E^*(x, y) & : - E(x, y). \\
E^*(x, z) & : - E^*(x, y), E^*(y, z).
\end{align*}
\]
The type U stands for every well-typed E-value, integer or string.
Recursive data types

Pre ::= (val: Natural, tail: List).
List ::= Pre + { NIL }.
Sub ::= (lst: List).
Max ::= (lst: Pre, val: Natural).

Sub(Pre(x, y)) :- Pre(x, y).
Sub(y) :- Sub(Pre(x, y)).
Max(Pre(x, NIL), x) :- Sub(Pre(x, NIL)).
Max(Pre(x, y), m) :- Sub(Pre(x, y)), Max(y, m), m >= x.
Max(Pre(x, y), x) :- Sub(Pre(x, y)), Max(y, m), m < x.

Pre(1, Pre(2, Pre(3, NIL))).
Maximum Elements of Sublists

\[
\begin{align*}
&\text{Max}(\text{Pre}(3, \text{NIL}), 3) \\
&\text{Max}(\text{Pre}(2, \text{Pre}(3, \text{NIL})), 3) \\
&\text{Max}(\text{Pre}(1, \text{Pre}(2, \text{Pre}(3, \text{NIL}))), 3) \\
&\vdots
\end{align*}
\]
1.3
Open–World Logic Programming
Open–World Logic Programming

- **Treat some parts of a program as open.**
  
  The least model of an open program is undefined. Formalizes the dichotomy between domain axioms and instances.

- **Phrase analysis problems over closures.**
  
  Search for closures of the program where some property is satisfied. Call this an OLP query.

- **Transparent integration with modern solvers.**
  
  OLP queries can be solved by generating constraint subproblems dispatched to modern SMT solvers.
Definition (Predicate Maze). An \( n \times n \) predicate maze is a pair \( M_{n \times n} \overset{\text{def}}{=} (s, p) \) where \( s \in [0, \ldots, n-1]^2 \) is the starting location and \( p \) is the wall predicate. Wall predicates satisfy the grammar:

\[
p ::= \neg p \mid p \land p \mid v \leq v \mid v \geq v \mid \text{true} \mid \text{false}.
\]

\[
v ::= x \mid y \mid 0 \mid 1 \mid 2 \mid \ldots \mid n.
\]

A solution to a maze is a sequence of locations \( l_0, \ldots, l_m \) such that:

1. \( l_0 = s \) and \( l_m = (n-1, n-1) \).
2. For every \( l_i = (a, b) \) and \( l_{i+1} = (c, d) \) then \( a = c \) or \( b = d \).
3. For every \( l_i = (a, b) \) then \( p[x \backslash a, y \backslash b] \) evaluates to \text{false}.

A maze is solvable if there is a solution.
Data Types Define Structure

Walls ::= \textit{new} (\texttt{any} \mid \texttt{Pred}).
Start ::= \textit{new} (\texttt{any} \texttt{Loc}).
Not ::= \textit{new} (\texttt{any} \texttt{Pred}).
Val ::= \texttt{Coord} + \{ \texttt{XC}, \texttt{YC} \}.
Coord ::= \{ 0..3 \}.
And ::= \textit{new} (\texttt{any} \texttt{Pred}, \texttt{any} \texttt{Pred}).
GEq ::= \textit{new} (\texttt{any} \texttt{Val}, \texttt{any} \texttt{Val}).
LEq ::= \textit{new} (\texttt{any} \texttt{Val}, \texttt{any} \texttt{Val}).
Loc ::= \textit{new} (\texttt{any} \texttt{Coord}, \texttt{any} \texttt{Coord}).
Pred ::= \textit{Not} + \textit{And} + \textit{GEq} + \textit{LEq} + \texttt{Boolean}.

Rch ::= (\texttt{Loc}). Hrz ::= (\texttt{Loc}). Sub ::= (\texttt{Pred} + \texttt{Val}).
EvlPred ::= (\texttt{Loc}, \texttt{Pred}, \texttt{Boolean}). EvlVal ::= (\texttt{Loc}, \texttt{Val}, \{ 0..3 \}).

All data types are algebraic; markers indicate open parts of program
Rules Provide The Axioms (I)

\[
\begin{align*}
\text{Hrz}(\text{Loc}(x, y)) & \implies \text{Start}(\text{Loc}(x, y)). \\
\text{Hrz}(\text{Loc}(x', y')) & \implies \text{Rch}(\text{Loc}(x, y)), x' = x + 1, x' : \text{Coord}, y' = y. \\
\text{Hrz}(\text{Loc}(x', y')) & \implies \text{Rch}(\text{Loc}(x, y)), x' = x - 1, x' : \text{Coord}, y' = y. \\
\text{Hrz}(\text{Loc}(x', y')) & \implies \text{Rch}(\text{Loc}(x, y)), y' = y + 1, y' : \text{Coord}, x' = x. \\
\text{Hrz}(\text{Loc}(x', y')) & \implies \text{Rch}(\text{Loc}(x, y)), y' = y - 1, y' : \text{Coord}, x' = x.
\end{align*}
\]

Rules for computing the horizon in \( \Pi_{4 \times 4} \).

\[
\begin{align*}
\text{Rch}(\text{Loc}(x, y)) & \implies \text{Walls}(p), \text{Hrz}(\text{Loc}(x, y)), \text{EvalPred}(\text{Loc}(x, y), p, \text{FALSE}). \\
\text{Sub}(p) & \implies \text{Walls}(p); \text{Sub}(\text{Not}(p)). \\
\text{Sub}(p), \text{Sub}(p') & \implies \text{Sub}(\text{And}(p, p')); \text{Sub}(\text{GEq}(p, p')); \text{Sub}(\text{LEq}(p, p')).
\end{align*}
\]

/// More rules for evaluating subexpressions.

Reachability and subexpression rules for \( \Pi_{4 \times 4} \).
Rules Provide The Axioms (II)

\[ \text{EvlVal}(\text{Loc}(x, y), v, c) :\]
\[ \text{Hrz}(\text{Loc}(x, y)), \text{Sub}(v), v : \text{Coord}, c = v; \]
\[ \text{Hrz}(\text{Loc}(x, y)), \text{Sub}(v), v = \text{XC}, c = x; \]
\[ \text{Hrz}(\text{Loc}(x, y)), \text{Sub}(v), v = \text{YC}, c = y. \]

\[ \text{EvlPred}(\text{Loc}(x, y), p, \text{TRUE}) :\]
\[ \text{Hrz}(\text{Loc}(x, y)), p = \text{TRUE}; \]
\[ \text{Hrz}(l), \text{Sub}(p), l = \text{Loc}(x, y), p = \text{LEq}(u, v), \]
\[ \text{EvlVal}(l, u, c), \text{EvlVal}(l, v, d), u \leq v; \]
\[ \text{Hrz}(l), \text{Sub}(p), l = \text{Loc}(x, y), p = \text{GEq}(u, v), \]
\[ \text{EvlVal}(l, u, c), \text{EvlVal}(l, v, d), u \geq v; \]
\[ \text{Sub}(p), p = \text{Not}(p'), \text{EvlPred}(\text{Loc}(x, y), p', \text{FALSE}); \]
\[ \text{Sub}(p), l = \text{Loc}(x, y), p = \text{And}(p', p''), \]
\[ \text{EvlPred}(l, p', \text{TRUE}), \text{EvlPred}(l, p'', \text{TRUE}). \]
\begin{align*}
\text{EvlPred}(\text{Loc}(x, y), p, \text{FALSE}) & :- \\
\text{Hrz}(\text{Loc}(x, y)), p = \text{FALSE};
\end{align*}

\begin{align*}
\text{Hrz}(l), \text{Sub}(p), l = \text{Loc}(x, y), p = \text{LEq}(u, v), \\
\text{EvlVal}(l, u, c), \text{EvlVal}(l, v, d), u > v;
\end{align*}

\begin{align*}
\text{Hrz}(l), \text{Sub}(p), l = \text{Loc}(x, y), p = \text{GEq}(u, v), \\
\text{EvlVal}(l, u, c), \text{EvlVal}(l, v, d), u < v;
\end{align*}

\begin{align*}
\text{Sub}(p), p = \text{Not}(p'), \text{EvlPred}(\text{Loc}(x, y), p', \text{TRUE});
\end{align*}

\begin{align*}
\text{Sub}(p), l = \text{Loc}(x, y), p = \text{And}(p', _), \text{EvlPred}(l, p', \text{FALSE});
\end{align*}

\begin{align*}
\text{Sub}(p), l = \text{Loc}(x, y), p = \text{And}(_ ,p'), \text{EvlPred}(l, p', \text{FALSE}).
\end{align*}
Open–World Queries

- **P ? G**
  
  Find a closure of the program by ground facts where a goal is satisfied.

- **$\Pi_{4 \times 4} ? Rch(Loc(3, 3))$**
  
  \[
  \{ \text{Start}(Loc(3, 3)). \text{Walls}(\text{FALSE}). \} \\
  \{ \text{Start}(Loc(0, 0)). \text{Start}(Loc(3, 3)). \text{Walls}(\text{And}(\text{GEq}(\text{XC}, 3), \text{LEq}(\text{XC}, 2))). \}
  \]

- **P[F]**
  
  Partially close P with facts F and remove “new” marking from all associated data types.

- **$\Pi_{4 \times 4}[\text{Start}(Loc(0, 0)).] ? \text{no } Rch(Loc(3, 3))$**
  
  For a closed starting location (0,0), is there a wall predicate that blocks all routes to the finish?
Maze Solving

A maze is a closure. Is there a route to the end from the starting location? Solvable by executing the closure.

Symbolic Model Checking

For a closed wall predicate, is there a starting location that solves the maze? Query for a closure of the starting point. Solved like a symbolic MC problem.

Program Synthesis

For a closed starting location, is there a wall predicate that blocks all routes to the finish?
Closed and Open Queries (II)

\[ \text{Walls} \]
\[
\text{Not}(\text{And}(
\text{Not}(\text{And}(
\text{And}(\text{LEq}(1, \text{XC}), \text{LEq}(\text{XC}, 2)),
\text{And}(\text{LEq}(1, \text{YC}), \text{GEq}(1, \text{YC})))))
\text{Not}(\text{And}(
\text{And}(\text{LEq}(1, \text{YC}), \text{LEq}(\text{YC}, 3)),
\text{And}(\text{LEq}(2, \text{XC}), \text{GEq}(2, \text{XC}))))).
\]

\[ \Pi_{4 \times 4}[^{\text{Start}(\text{Loc}(0, 0))}.] \quad \text{no} \quad \text{Rch}(\text{Loc}(3, 3)) \]
Solving and Search

Use state-of-the-art **satisfiability modulo theories** (SMT) solver Z3 to solve quantifier-free formulas.

1. **FORMULA Specification**
2. Symbolic Execution → **SMT Formula**
3. Add symmetry breaking → **Z3 Solver**
4. Get solution, reconstruct model
5. Guess symbolic world
6. Encode solution region
7. Try something new
8. Pick next region
Questions?

FORMULA 2.0
Ethan K. Jackson,
Research in Software Engineering,
Microsoft Research, Redmond, WA, USA