Finding Models in Model-Based Development

Wolfram Schulte

Microsoft Research, Redmond, USA

@

Models 2011

Wellington, NZ

Prelude :
Modeling, Microsoft and Me

Timeline & Work

1999

- Language integrated queries
- Model-based testing
- Design-by-contract
- Unit testing
- Task-parallel library
- Formula

2011

Linq

One language for writing three-tier applications, no marshaling, no security issues

? How to grow a language so that it captures XML, OO, and SQL, with different types, literals and query lang.

```
scoreQuery =
from score in scores where score > 80 select score;
```

- Linq ships since .NET 3.0. -- widely adopted. (Structural sub-typing, new query syntax reduces to generic query operators)

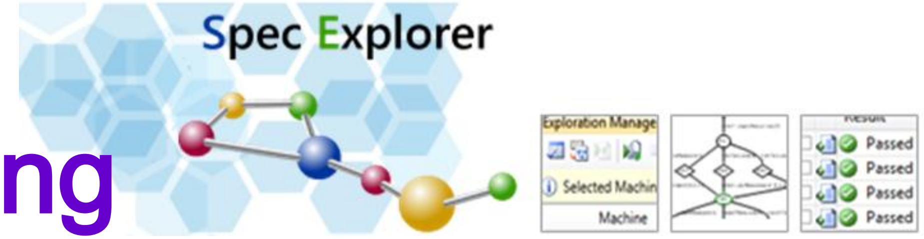
Linq

Lessons learnt

- ✓ Generalize: Translate syntactic sugar to general concepts (higher-order functions)
- ✓ Be pragmatic: Structural typing only within one assembly (no new CLR)
- ✓ Enable ecosystem: Access to query construction at runtime (pLinq, DryadLinq, etc)

See: **The world according to LINQ**. CACM(10): 45-51 (2011)

Model-based Testing

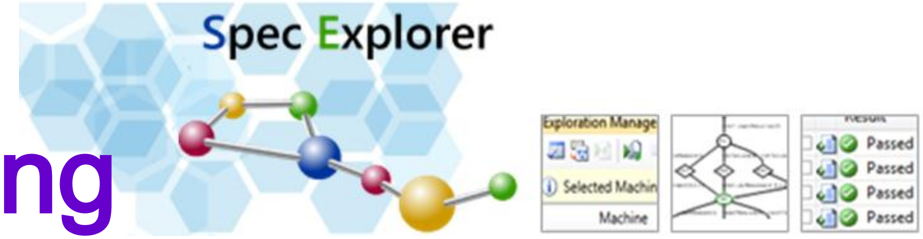


Test interoperability of stateful protocols

? How to describe protocols, what's the conformance notion, how to generate tests for non-deterministic systems, how to make it user friendly

- SpecExplorer has been used for testing 200+ protocols, 50% less cost than manual test (SE uses model checking like Java pathfinder)

Model-based Testing



✓ Minimize adoption: Use existing language (C#)

✓ Embed in existing process: Scenario control

✓ Support debugging: Visualization key

See: Microsoft's Protocol Documentation Program: Interoperability Testing at Scale, CACM 54(7):51-57 (2011)

Design-by-Contract

Capture developer intentions, detect bugs early

? What's the meaning of object invariants in the context of inheritance, aliasing, callbacks, and multi-threading

```
int GetTotal() {
    Contract.Requires(GetItems().Count > 0);
    Contract.Ensures(Contract.Result<int>()>=0);
```

- Spec#/Code-Contracts adopted, ship partly in .NET 4.0, 50k external users (rewriting for runtime checking, verification for extended checking)

Design-by-Contract

- ✓ Pay as you go: From runtime to static checking
- ✓ Push QA upstream: Design time verification, actionable analysis results
- ✓ Separate concerns: verification condition generation and proof (using SMT)

See: **Specification and verification: the Spec# experience**. CACM 54(6):81-91 (2011)

Also: **Satisfiability modulo theories: introduction and applications**. CACM 54(9), 69-77 (2011).

Lessons learnt

For design

- Use succinct, expressive descriptions
- Build on solid foundations

For success

- Solve a real problem
- With as little friction as possible

For analysis

- Give instant feedback
- Find subtle bugs
- Give confidence in correctness

For implementation

- Factor and reuse, reuse, reuse

FORMULA

Modeling Foundations.

Formal Modeling Using Logic Programming and Analysis

Ethan Jackson, Nikolaj Bjørner and **Wolfram Schulte**, RiSE, Microsoft Research

Dirk Seifert, Markus Dahlweid and Thomas Santen, EMIC, Microsoft Research

Formula: Main Ideas

Language ideas

- Abstractions as constraints
- Constraints expressed as logic program
- Logic program encapsulated in domains
- Domains composed/transformed to build new abstractions

Analysis ideas

- Analysis using model finding
- Model finding by fixpoint computation and SMT
- Used for design space exploration, transformation verification, and model checking

An example

Friends and Family

Given

- facebook.com, a social network
- ancestry.com, a US family tree

Build recommender system for facebook.NEXT

- Use ancestry to make more friend recommendations for facebook
- Can we do this without new exploits?

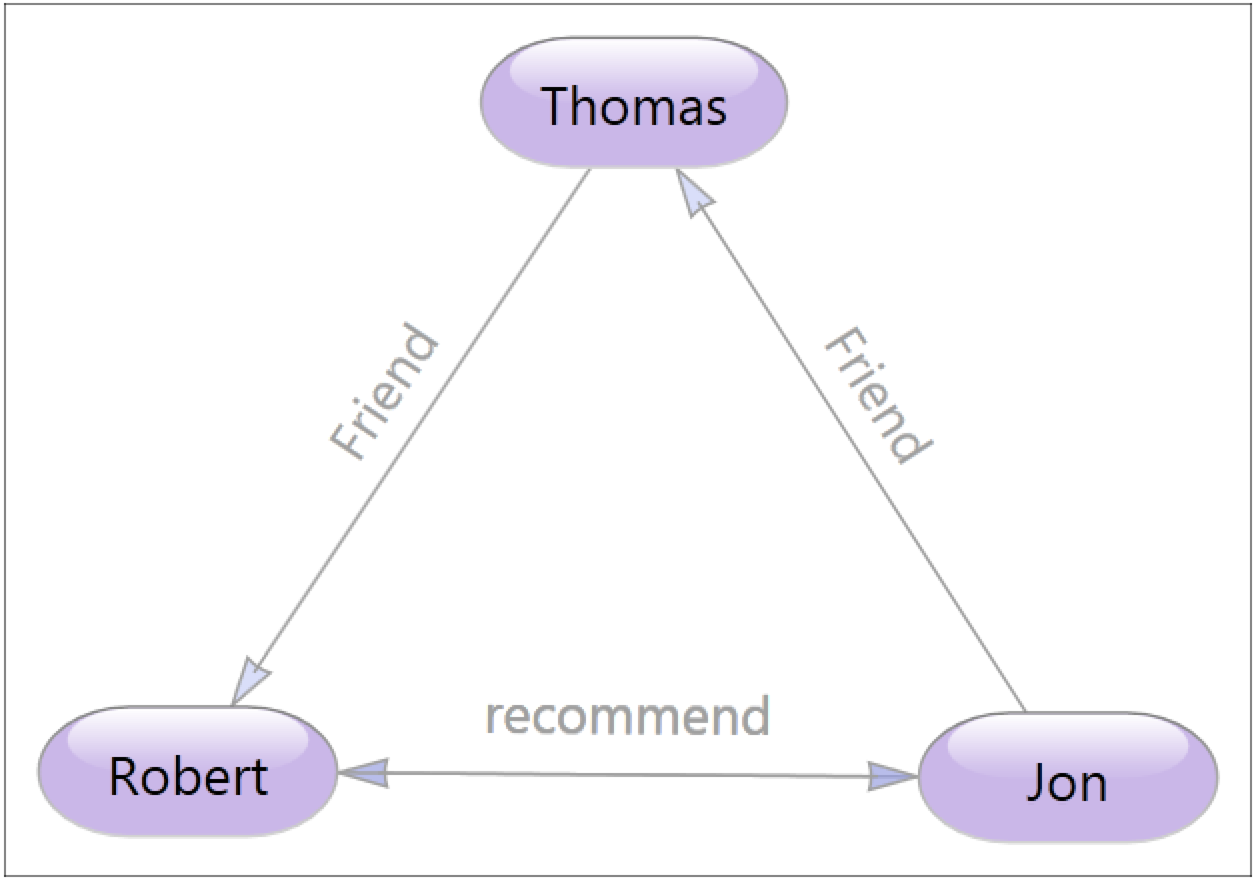
Domains

```
domain Facebook
{
   Gender ::= { male, female }.
   Person ::= (name: String, gender: Gender).
   Friend ::= (me: Person, you: Person).

   friendsFriend(x, y) :- Friend(x, y).
   friendsFriend(x, z) :- friendsFriend(x, y),
                          friendsFriend(y, z).

   recommend(x, y)     :- friendsFriend(x, y),
                          fail Friend(x, y), x != y.
   recommend(y, x)     :- recommend(x, y).

   conforms := fail Friend(x, x).
}
```

Models and Assertions

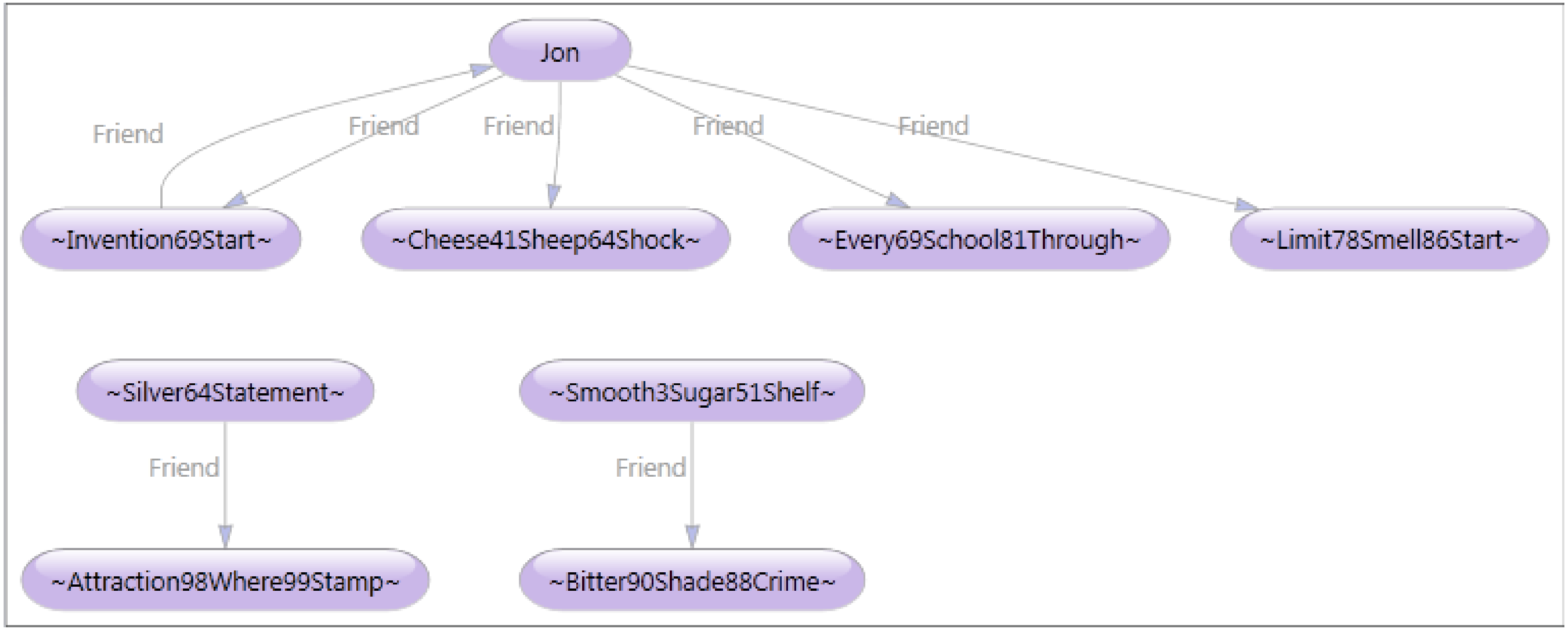
```
[CheckTermsExist(
    recommend(Person("Jon", male), Person("Robert", male)))
]
model MODELS2011 of Facebook
{
   Person("Jon", male).
   Person("Thomas", male).
   Person("Robert", male).
   Friend(Person("Jon", male), Person("Thomas", male)).
   Friend(Person("Thomas", male), Person("Robert", male)).
}
```

Visualization of Relationships



Synthesis and Partial Models

```
domain JonHasFriends extends Facebook
{ conforms := count(Friend(Person("Jon", male),_)) >= 4. }

[Introduce(Friend, 20)] [Introduce(Person, 20)]
partial model FriendlierMODELS2011 of JonHasFriends {}
```



Use constraints to model !

Concept relationships

- is, has, friend, transition, etc

Temporal relationships,

- Time constraints (intervals), a scheduler

Spatial relationships

- Location (regions), placement

Logic

Logic – the foundation of Formula

Denotation

- LP program is first order logic (FOL) with fixpoints

Evaluation

- Compute logical consequence operator (bottom-up)

Model finding

- Search for facts that satisfy query

LP Syntax

- **Type**

```
Friend ::= (me: Person, you: Person).
```

- **Facts**

```
Friend(Jon, Thomas).
Friend(Thomas, Robert).
```

- **Rule**

```
friendsFriend(x, y) :- Friend(x, y).
friendsFriend(x, z) :- friendsFriend(x, y),
                       friendsFriend(y, z).
```

- **Query**

```
conforms := fail Friend(x, x).
```

Logical Semantics

Reason over the **least knowledge *K*** satisfying …

- Fact axioms

$Friend(Thomas, Robert) \in K$

$Friend(Jon, Thomas) \in K$

- Rule axioms

$\forall x, y.\, Friend(x, y) \in K \Rightarrow friendsFriend(x, y) \in K$

$$\forall\, x, y, z.\, friendsFriend(x, y) \in K \wedge friendsFriend(y, z) \in K \Rightarrow friendsFriend(x, z) \in K$$

- Axioms from Clark completion

$$\forall x, z.\, friendsFriend(x, z) \in K \Rightarrow (\exists y.\, friendsFriend(x, y) \in K \wedge friendsFriend(y, z) \in K) \vee Friends(x, z) \in K$$

Logical Semantics

Negation tests for absence of knowledge

- Rule axioms

$$\forall x, y.\, friendsFriend(x,y) \in K \wedge Friend(x,y) \notin K \Rightarrow recommend(x,y) \in K$$

$$\forall x, y.\, recommend(x,y) \in K \Rightarrow recommend(y,x) \in K$$

- Axioms from Clark completion

$$\forall x, y.\, recommend(x,y) \in K \Rightarrow$$
$$(friendsFriend(x,y) \in k \wedge Friend(x,y) \notin K) \vee recommend(y,x) \in K$$

- Axioms from Queries

$$conforms \Leftrightarrow \forall x.\, Friend(x,x) \notin K$$

Fixpoint Semantics for Evaluatio

K_0

$$\{\, Friend(Jon, Thomas),\ Friend(Thomas, Robert)\}$$

$$\forall x, y.\ Friend(x, y) \in K \Rightarrow friendsFriend(x, y) \in K$$

K_1

$$\{Friend(Jon, Thomas),\ Friend(Thomas, Robert)\ friendsFriend(Jon, Thomas),\ friendsFriend(Thomas, Robert)\}$$

$$\forall\, x, y, z.\ friendsFriend(x, y) \in K \wedge friendsFriend(y, z) \in K \Rightarrow friendsFriend(x, z) \in K$$

K_2

$$\{Friend(Jon, Thomas),\ Friend(Thomas, Robert)\ friendsFriend(Jon, Thomas),\ friendsFriend(Thomas, Robert),\ friendsFriend(Jon, Robert)\}$$

Model finding

Can you **find a finite set of facts W** where Jon has 4 friends?

$$\mathbf{W} \left\{ \begin{array}{c} \exists p_1, p_2 . Friend(p_1, p_2) \in K \\ \vdots \\ \exists p_i, p_{i+1} . Friend(p_i, p_{i+1}) \in K \end{array} \right.$$

- Close the program with facts W. We call W a world
- Determine the values for p_i using symbolic execution and SMT

Comparisons

Evaluation

	LP	CLP	ASP	Datalog	Formula
	SLD (top down)	SLD with Constraints	Stable Model Computation	Bottom-up	Bottom-up

Model Finding

	LP	CLP	ASP	Datalog	Formula
	No	No	Brave/cautious reasoning	No	Yes, using OWA

Language

Language – making it usable

Core: LP with Open World Assumption

Types: Semantic Subtyping and Type inference

Modules: Domains, Composition and Transformations

Regular Types

For

- Type checking, i.e. early bug detection
- Constraint solving, i.e. restricting possible solutions
- Efficient symbolic execution, i.e. fewer terms to match

Composing Abstractions

Inclusion – textual

```
domain B includes A {..}
```

Renaming – deep copy

```
domain C includes A as X {…}
```

Extension – preserve semantics

```
domain D extends A {…}
```

i.e. `D.conforms = A.conforms + …`

Cont'd: FamilyTree

```
domain FamilyTree {
    Status::= { married, divorced }.
    Gender   ::= { male, female }.
    Person   ::= (name: String, gender: Gender).
    Child    ::= (name: Person, mother: Person, father: …).
    Marriage ::= (p1: Person, p2: Person, st: Status).

    //////// Data computed about a family tree.
    ancestor ::= (p1: Person, p2: Person).
    bioRel   ::= (p1: Person, p2: Person).
    lawRel   ::= (p1: Person, p2: Person, st: Status).
    show     ::= (p1: Person, p2: Person).
    //// ....
    show(p, p0)       :- lawRel(p, p0, married).
    show(p, p0)       :- Marriage(p, p0, married).
    show(p0, p)       :- show(p, p0).
}
```

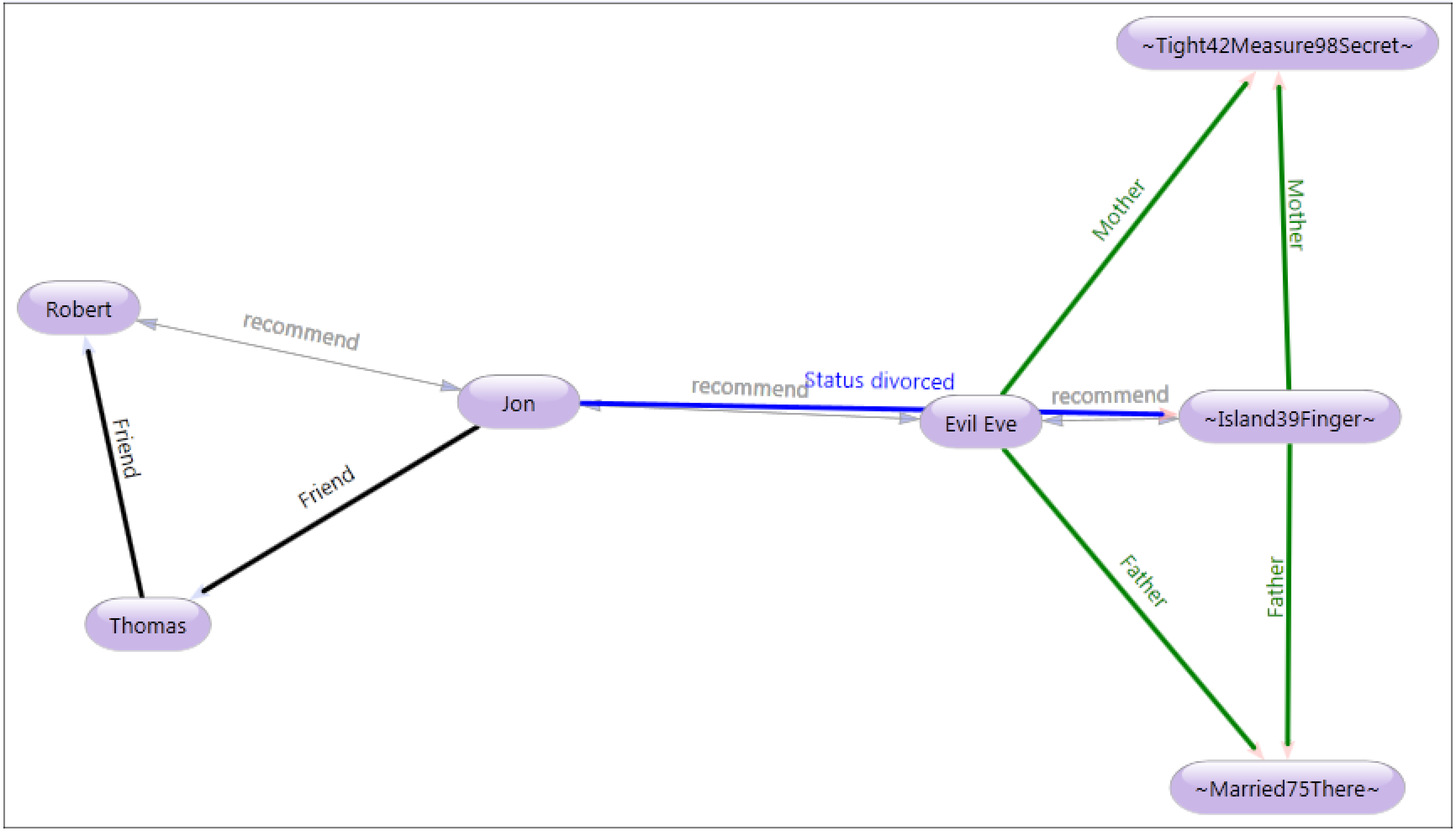
FacebookNext's New Recommenda

```
domain FacebookNext extends Facebook, FamilyTree {
    recommend(x, y) :- lawRel(x, y, _).
    recommend(x, y) :- bioRel(x, y).
}
```

Can Eve Exploit New Recommendat

```
domain EvesExploits extends FacebookNext {
    conforms :=
    // Eve wants to get recommended to be a friend of Jon
        recommend(pEve, pJon),
        pEve = Person("Evil Eve", female),
        pJon = Person("Jon", male),
    // But, she cannot directly add a friend link
        fail friendsFriend(pEve, _),
        fail friendsFriend(_, pEve),
    // And she cannot modify any ones family tree
    // in a way they can observe.
        fail show(pJon, _),
        fail show(Person("Robert", male), _),
        fail show(Person("Thomas", male), _).
}
```

Visualization of Exploit



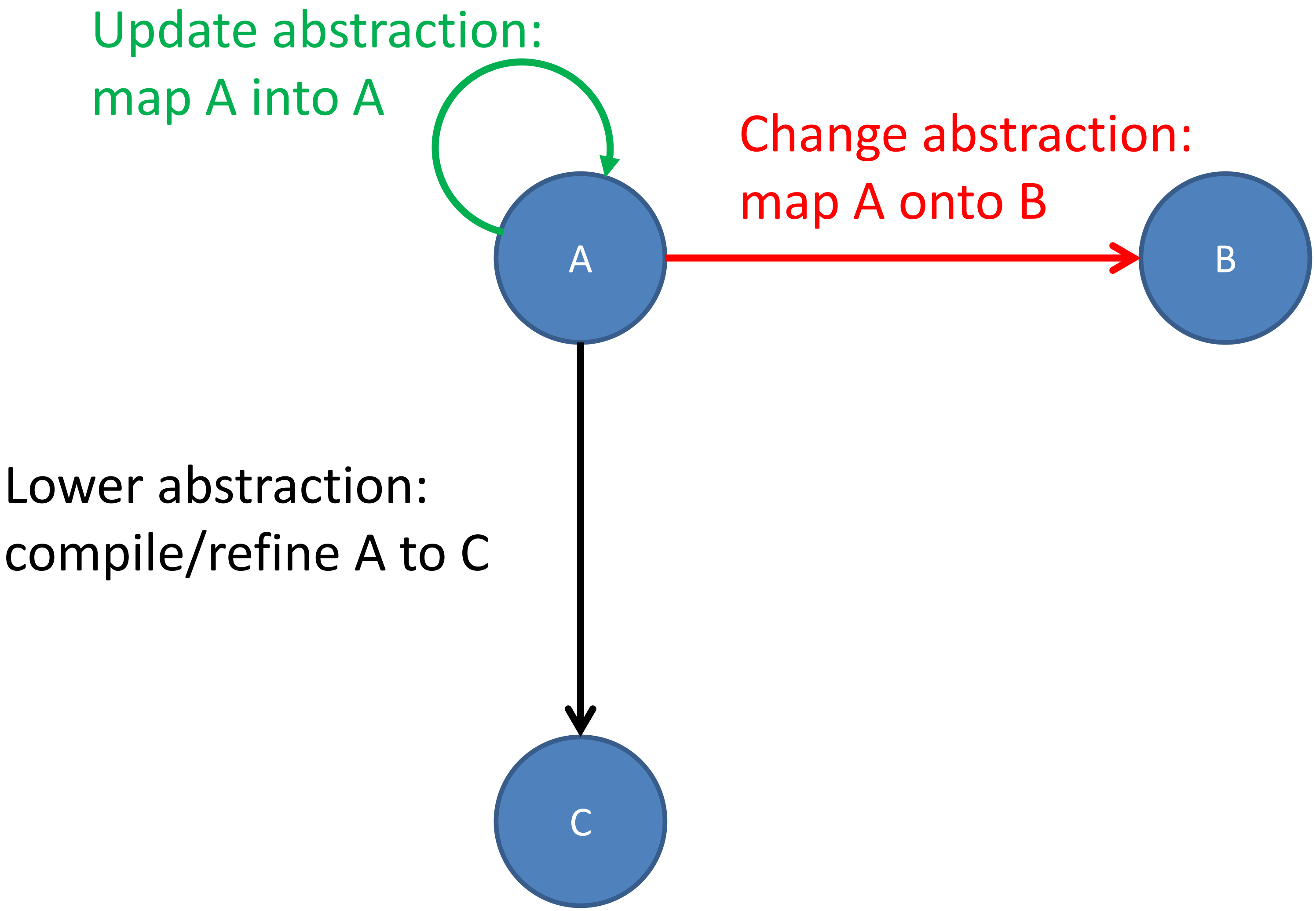
From Domains to Transformatior

Domains have no states or mutation

Behavior can be introduced by introducing time, e.g. state updates increase time

Alternatively use a transformation

Transformations change abstractions



Transforms

…are big step operations. They

- take models and return models
- are expressed using Formula’s core
- execute until a fixpoint is reached

Deleting a Person in Facebook

```
transform DeletePerson <name: String> from Facebook as i
                                        to  Facebook as o
{
   o.Person(n, g) :- i.Person(n, g), n != name.
   o.Friend(x, y) :- i.Friend(x, y),
                     x.name != name, y.name != name.

   notDeleted := o.friendsFriend(Person(name, _),_).
   notDeleted := o.friendsFriend(_,Person(name, _)).
}
```

Goodies

The stuff that's not in the language

- Pre-solving: Compute cardinalities
- Post-solving: Compute non-isomorphic solutions

- Language extensibility: Custom attributes
- System reuse: Public API for everything

- Debugging: Visualization for free

Some applications

Micro case studies

Synchronous Dataflow Languages (300 lines)

- Semantics given by interpreter defined using domains
- Compiler defined via transforms
- Translation validation via model finding

Timed Automata (200 lines)

- State as domains
- Transitions defined via transforms
- Checking Trace behavior via LTL model checking

First external adopters

Declarative configuration: Configure and debug security group settings (MSFT: Group policies)

- Compute configurations for new resources which obey 200k+ existing policies; debug erratic behavior; Formula used as intermediate language

Design space exploration: Synthesize software architecture guaranteeing time constraints (Automobile company: under NDA)

- Compute schedule as data flow graph (1000+ nodes) over time intervals; Formula used to capture constraints and compute schedule

Large scale integration: Semantic anchoring for large scale model integration (DARPA: Meta)

- Models (here: mechanical, electrical, thermo, software, hardware) from Adaptive Vehicle Make. Formula as glue for compos. and integrity check.

Wrap-up

Future/ongoing work

- Parametric optimization of models
- Finding root cause for unsatisfiability
- Composing and model checking of transformations
- Optimization for design space exploration

Reflections

- Language design is hard
- Solver behavior sometimes unpredictable
- Robust tools require a lot of effort
- Opportunities for modeling abound
- Modeling works
- Work with us

For more info…

Formula community, downloads, tutorial

- http://research.microsoft.com/formula

For more info about RiSE team

- http://research.microsoft.com/rise

Tools to experiment with

- http://www.rise4fun.com