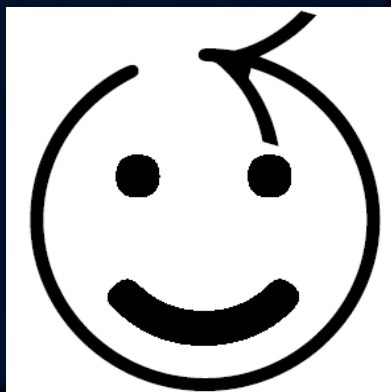
未来言語 Alloy

酒井 政裕



What is Alloy?

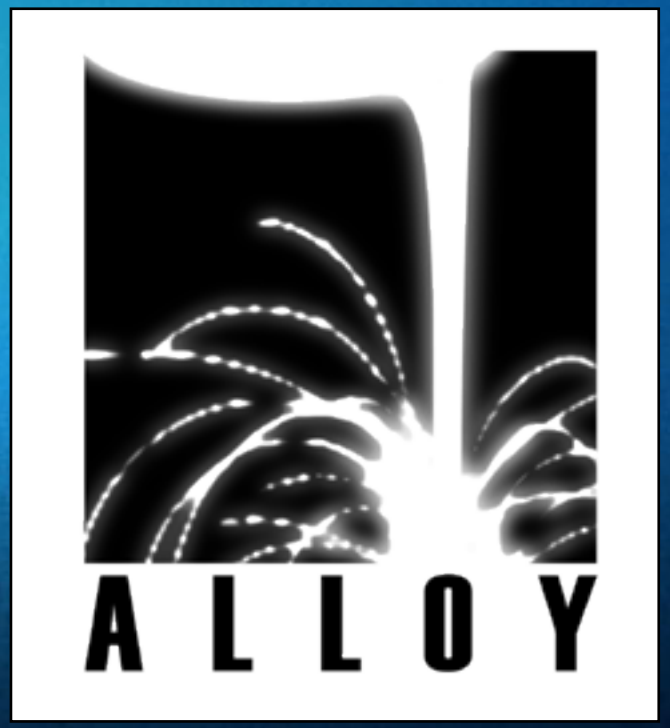
n., v. 合金(する); 混ぜ物(する); 品を落す.

EXCEED英和辞典 より

What is Alloy?

関係論理 のモデル発見器
とその入力言語

- モデル発見器?
- 関係論理?
- それっておいしいの?



http://alloy.mit.edu/alloy4/

モデル発見器 (Model Finder)

- 条件を与えると、その条件を満たす具体例(モデル)を探してくれるもの。
- 例:
  - 条件: 連立方程式
  - モデル: 連立方程式の解
- 例:
  - 条件: 親子関係の制約
  - モデル: 親子関係の具体例 (家系図的なもの)
- モデルを見つけることで、様々な問題を解ける

Alloy言語

- 関係論理 = 一階述語論理 + α
- 集合っぽい
  - 集合・関係演算が存在
  - 式は個体ではなく集合を表し、個体は一点集合で表す
- オブジェクト指向っぽい
  - Javaのインターフェース風に型を定義し、アクセスできる
- 推移閉包演算がある
- 独特だけど、慣れると便利。

親子関係：条件の記述

```
//人の集合
abstract sig Person {
  father: lone Man,
  // 人は高々一人の男を
  // 父として持つ。
  // father は PersonとManの間の
  // 二項関係!
  mother: lone Woman,
}

sig Man extends Person {
  wife: lone Woman
}
```

```
sig Woman extends Person {
  husband: lone Man
}

fact {
  no p: Person |
     p in p.^(mother + father)
  // + は関係の和, ^ は推移閉包
  // 自分自身の祖先にならない

  wife = ~husband
  // ~は逆
}
```

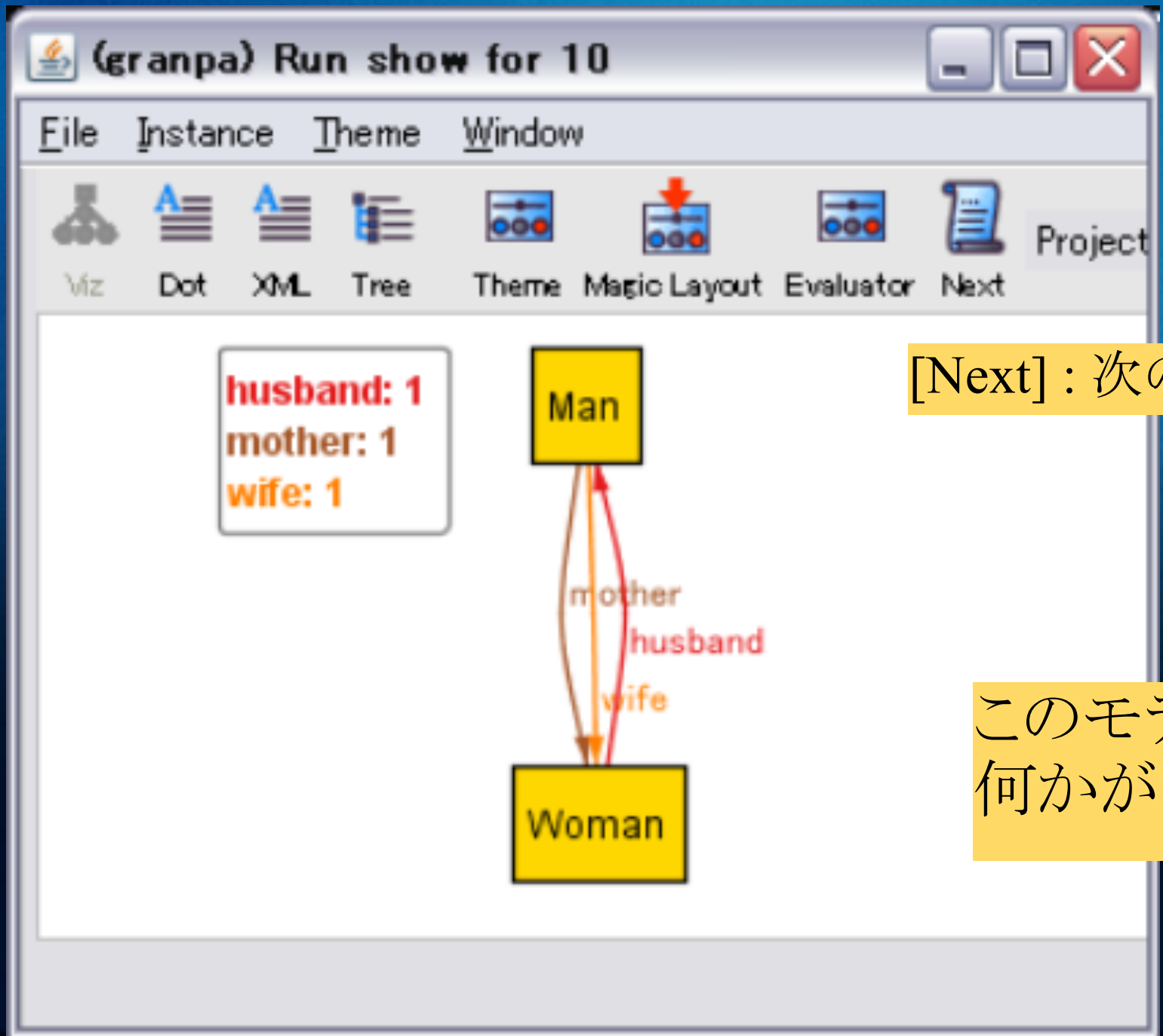
親子関係: モデルの探索

- ソースコードに右の記述を追加してリロード後、[Execute] > [Run show] を選択
- モデルを発見! →
- Instanceをクリックするとモデルが表示される

```
pred show() {}
run show
```

Instance found.
Predicate is consistent.

親子関係: 出てきたモデル



[Next] : 次のモデルを探索

このモデルは
何かがおかしい!?

親子関係: モデルの探索 (2)

```
fun grandpas(p: Person) : set Person {
  p.(mother + father).father
}
pred ownGrandpa(p: Person) {
  p in grandpas[p]
}
run ownGrandpa
```

No instance found
Predicate may be inconsistent.

反例の探索

- 満たされるべき性質を記述して、反例がないことを検査する
- 右の記述を追加後、リロードし、[Execute] > [Check noSelfFather] を選択
- すると……

```
assert noSelfFather {
no m: Man | m = m.father
}
check noSelfFather
```

No counterexample found
Assertion may be valid.

使い方の流れ

- sig と fact を使って公理を記述
- run を使って、公理の元で条件を満たすモデルが存在することを確認
- assertを使って、満たされるべき性質を記述
- checkを使って反例が存在しないかを検査

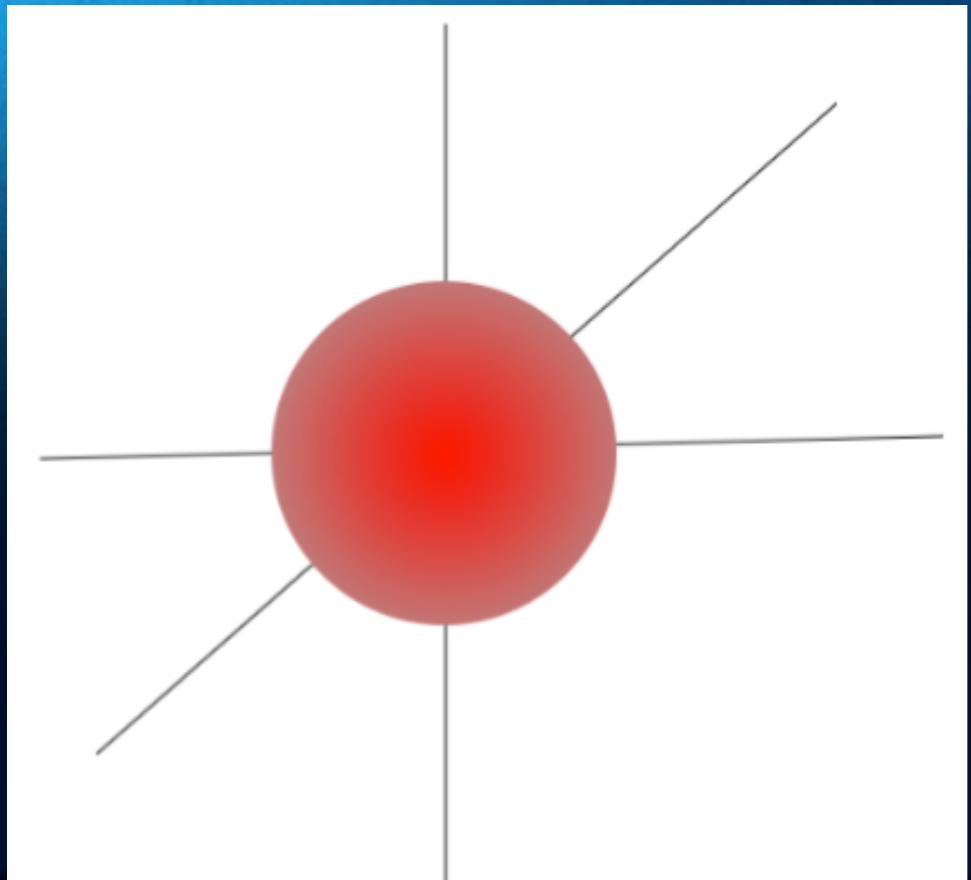
探索範囲

- Runでは
  - Instance found. Predicate is consistent.
  - No instance found. Predicate may be inconsistent.
- Checkでは
  - Counterexample found. Assertion is invalid.
  - No counterexample found. Assertion may be valid.
- 「may」って……

探索範囲

- 実は、あらかじめ指定されたサイズ(インスタンス数)以下のモデル(反例)だけを探索している。
  - 探索範囲を限定することで自動化
- その範囲外にモデルが存在する可能性が残されている。
  だから may

- 「run show for 6 Person」などとしてサイズを指定。



Alloyで出来ること

- Alloy言語を使ったモデリング
  - 一階述語論理+αの表現力
- モデルの探索 (run)
  - あらかじめサイズの上限を指定しての探索
- 性質の検査 / 反例の探索 (check)
  - あらかじめサイズの上限を指定しての探索
- モデルや反例を見易く表示

Alloy Analyzerの仕組み

- 関係論理のエンジンKodKod
  - 関係論理の論理式をSAT問題にエンコードして、SATソルバを利用して解く。

SAT問題

- SAT = SATisfiability
- 命題論理の論理式を、各変数に真偽を割り当てて、全体を真にできるか?
- 例
  - 問題: (x1∨x2)∧(x1∨¬x2)∧(¬x1∨¬x2)
  - 解: x1=T, x2=F のとき真になる。
- NP完全だけど、最近のソルバーは速くて、結構大きな問題も解ける。

Alloyの背景

- ソフトウェアは抽象化(Abstraction)に基づいて構築される
  - うまい抽象化ができれば、小さくて単純なインターフェースが実現でき、機能の追加も容易になる。
  - 抽象化の仕方がまずいと、簡単な変更ですら大規模な変更が必要になったりする。
- 正しい抽象化・設計は重要だけど、設計を検証するにはどうしたらよい?
  - 設計はコードと違ってテスト出来ない（＞＜）

Alloyの背景(2)

- 設計を検証するにはどうしたらよい?
- 形式仕様記述: Zとか
  - 複雑
  - 検証は定理証明によって行われる。定理証明は大変。
- Alloy
  - 簡単
  - スコープを限定しての自動検証が可能
    - 小スコープ仮説「ほとんどの欠陥は小さな範囲内に反例を持つ」⇒ スコープを限定しての検証でも、ほとんどの欠陥は発見可能なはず。
    - Lightweight Formal Method

Alloyを使って色々やってみる

- 数独
- ミニライフゲーム

数独

http://ja.wikipedia.org/wiki/数独 より

- 空いているマスに1～9のいずれかの数字を入れる。
- 縦・横の各列及び、太線で囲まれた3×3のブロック内に同じ数字が複数入ってはいけない。

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

数独: ルール

```
abstract sig Number { data:
Number->Number }
abstract sig Region1, Region2,
Region3 extends Number {}
one sig N1, N2, N3
   extends Region1 {}
one sig N4, N5, N6
   extends Region2 {}
one sig N7, N8, N9
   extends Region3 {}

pred complete(rows, cols: set
Number) { Number in cols.
(rows.data) }
```

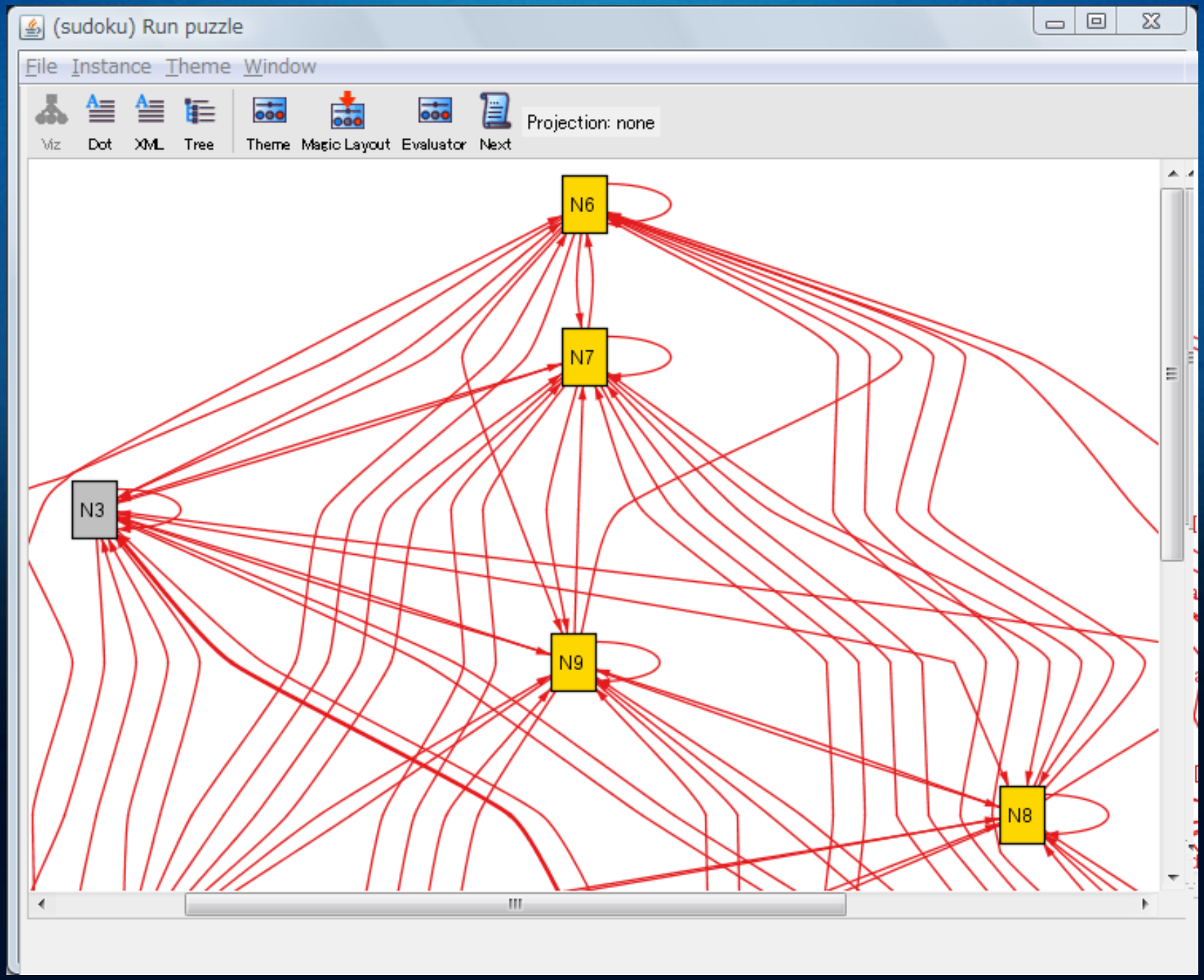
```
fact {
  all x, y: Number |
    lone y.(x.data)
  all row: Number |
    complete[row, Number]
  all col: Number |
    complete[Number, col]
  complete[Region1, Region1]
  complete[Region1, Region2]
  …
  complete[Region3, Region3]
}
```

数独: 具体的な問題の記述

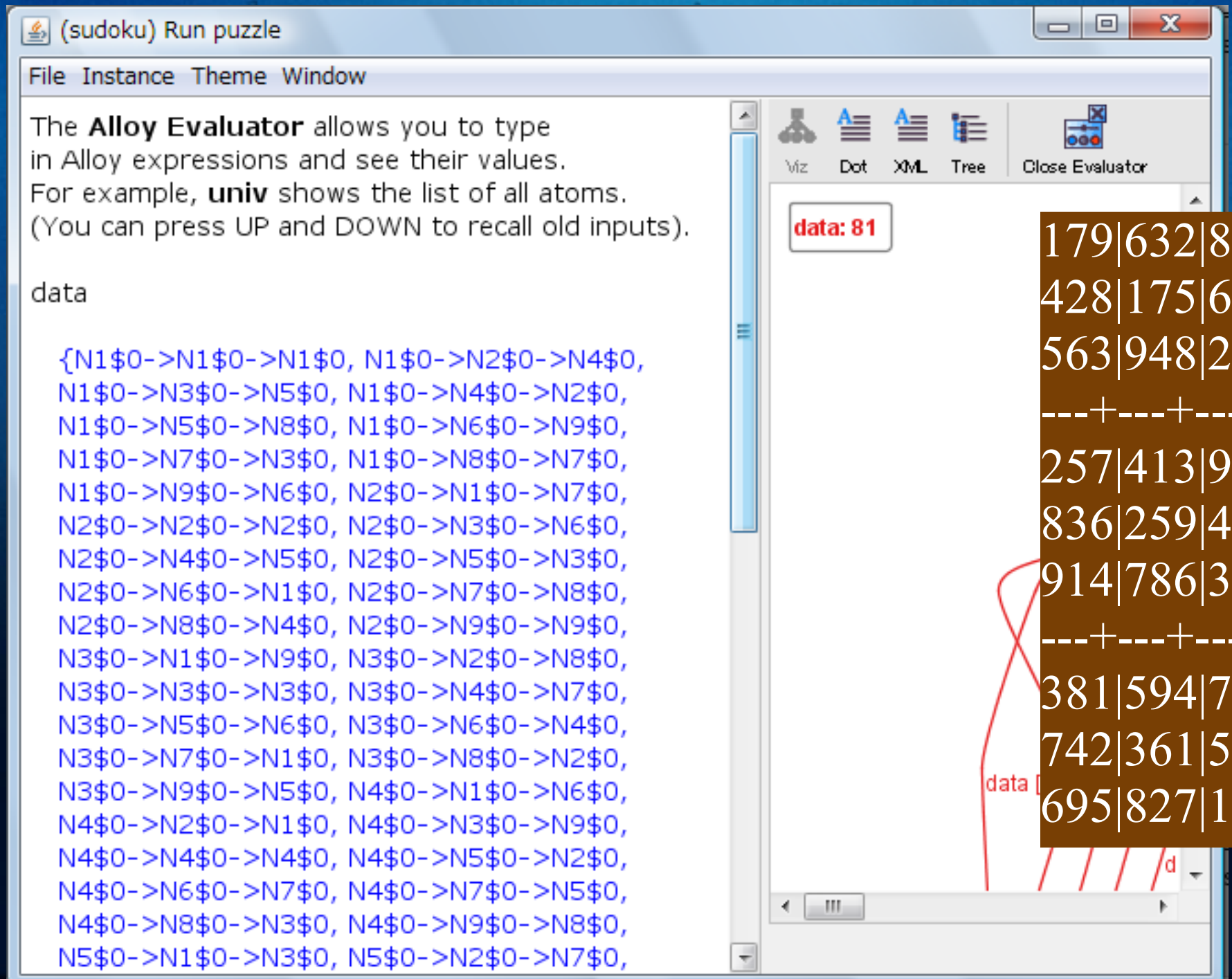
```
pred puzzle() {
  N1->N1->N1 + N1->N4->N2
+ N1->N7->N3 +
  …
  N9->N3->N1 + N9->N6->N2
+ N9->N9->N4
  in data
}

run puzzle
```

```
1  |6  |8
 2 | 7 | 9
  3|  8|  1
---+---+---
2  |4  |9
 3 | 5 | 1
  4|  6|  2
---+---+---
3  |5  |7
 4 | 6 | 8
  5|  7|  3
```

数独：解


(sudoku) Run puzzle
File Instance Theme Window
Viz
Dot
XML
Tree
Theme
Magic Layout
Evaluator
Next
Projection: none
N6
N7
N3
N9
N8

数独：解



179|632|845
428|175|693
563|948|271
---+---+---
257|413|968
836|259|417
914|786|352
---+---+---
381|594|726
742|361|589
695|827|134

Mini Life Game

『4日で学ぶモデル検査（初級編）』より。

- 窓辺に1列、8個の花が並んでいて、一日単位で咲いたり閉じたりする。
- 各花はご近所(自分と左右の花)に咲いている数に応じて、次の日に咲くかを決める
  - 0なら、明日は閉じている。
  - 1なら、明日は開いている。
  - 2なら、明日は閉じるか咲くか。
  - 3なら、明日は閉じる。
- 左端の一つが咲いている状態から初めて、すべての花が開くことはあるか?

Mini Life Game : ルール

```
sig Flower {
  next : lone Flower,
}

fact {
  no x : Flower | x in x.^next
  one f : Flower |
     Flower = f.*next
}

fun neighber(x : Flower)
  : set Flower {
  x + x.next + x.~next
}
```

```
sig State {
  inBloom : set Flower,
  next : lone State,
}

fact {
  all s1 : State, s2 : s1.next,
       x : Flower {
     let n = neighber[x],
         bn = s1.inBloom & n {
       no bn => !(x in s2.inBloom)
       one bn => x in s2.inBloom
       #bn=3 =>
           !(x in s2.inBloom)
     } } }
```

Mini Life Game モデル（続き）

```
pred show() {}
run show for exactly 8 Flower,
4 State
```

```
pred solve(s1 : State, s2 : State)
{
  s1.inBloom =
      {x : Flower | no x.~next }
  s2.inBloom = Flower
  s2 in s1.*next
  s1.*next = State
}
run solve for exactly 8 Flower,
8 State
```

Mini Life Game 解

State0 10000000
State1 11000000
State2 01100000
State3 10110000
State4 10101000
State5 10101100
State6 10110110
State7 11111111

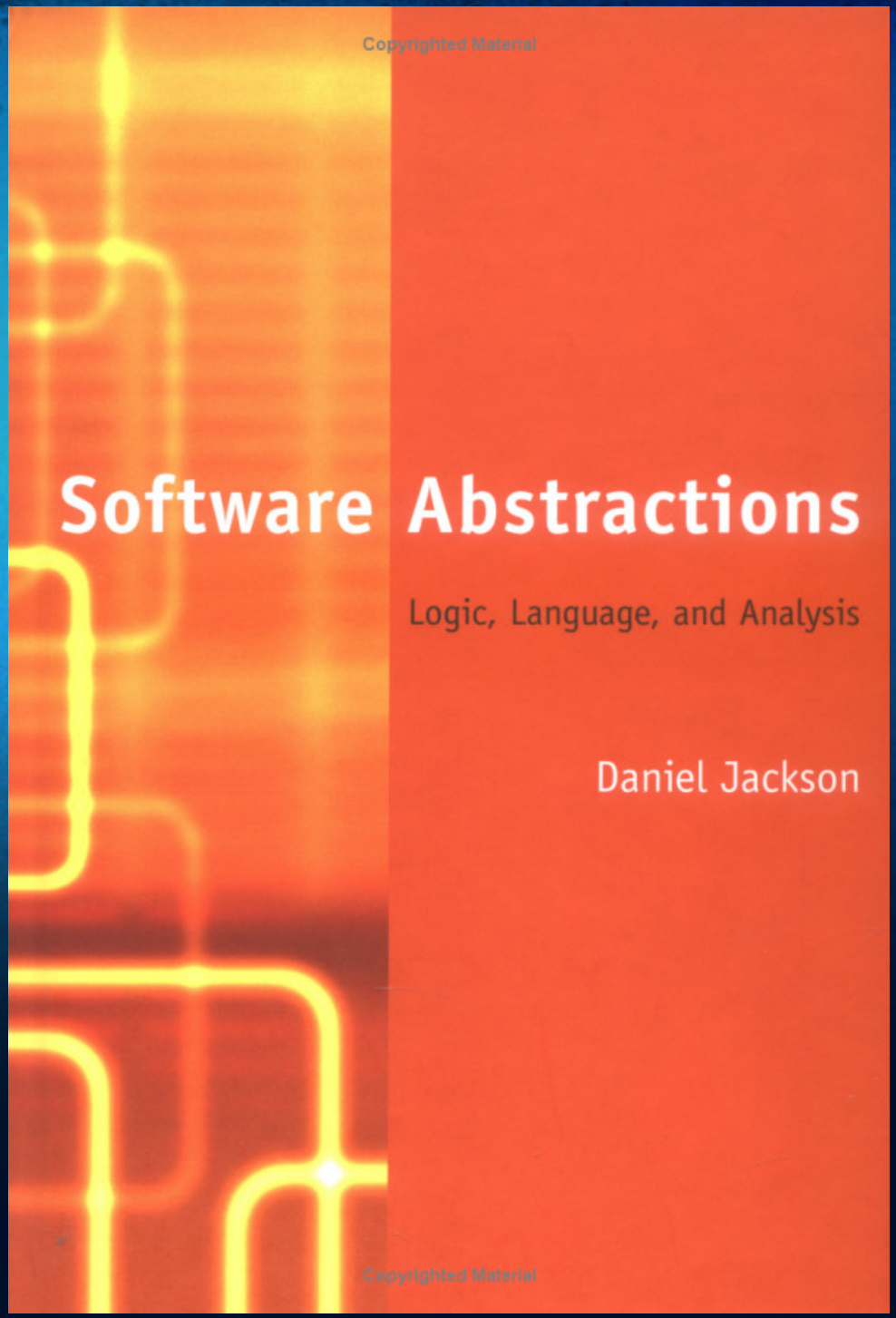
応用とか

- プログラムの検査 (Bounded Model Checking)
  - Forge, JForge, CForge
- セキュリティ・リスクアセスメント
  - Risk-driven architectural decomposition
- セマンティックウェブ・オントロジーの推論
  - Reasoning support for Semantic Web ontology family languages using Alloy
- MIT course scheduler
  - 卒業に必要な単位取得計画のプランニング
- などなど

教科書

- “Software Abstractions” Daniel Jackson
- Alloyの唯一の教科書

- 現在、鋭意翻訳中



参考文献

- 何はともあれ本家サイト
  http://alloy.mit.edu/
- 中島震、鵜林尚靖:Alloy – 自動解析可能なモデル規範形式仕様言語、コンピュータソフトウェア、2009年8月号
- Alloy Analyzer でググるといいかも

まとめ：未来

- Alloyみたいな技術がソフトウェアの信頼性をあげてくれるといいなぁ。
- パズルのルールを書いたら、適当に解いてほしいなぁ。

- 未来って何だっけ?