Modeling with ClaferIG

Jia (Jimmy) Liang

University of Waterloo

March 5, 2012

Part I

Quick glance at Clafer

Model vs instances

Clafer 1: Clafer model

```
Processor
    Core 1..2
```

Model vs instances

Clafer 4: Clafer model

```
Processor
    Core 1..2
```

Clafer 5: First instance

```
Processor
    Core
```

Clafer 6: Second instance

```
Processor
    Core1
    Core2
```

Clafers vs constraints

Clafer 7: Model with constraints

```
Person *
    Age : integer
    [Age ≥ 0]
```

Part II

What is the problem?

Satisfiability

$$satisfiable(model) = \begin{cases} true & \exists x \in instances(model) \\ false & otherwise \end{cases}$$

Satisfiability

$$satisfiable(model) = \begin{cases} true & \exists x \in instances(model) \\ false & otherwise \end{cases}$$

Theorem

Satisfiability for first-order logic is undecidable.

Figure: Alonzo Church

Scope

$$satisfiable(model, scope) = \begin{cases} true & \exists x \in instances(model, scope) \\ false & otherwise \end{cases}$$

Scope

$$satisfiable(model, scope) = \begin{cases} true & \exists x \in instances(model, scope) \\ false & otherwise \end{cases}$$

Clafer 9: Fermat's last theorem

```
A 1..*
B 1..*
C 1..*
[ (#A × #A × #A) +(#B × #B × #B) = #C × #C × #C ]
```

Scope

$$satisfiable(model, scope) = \begin{cases} true & \exists x \in instances(model, scope) \\ false & otherwise \end{cases}$$

Clafer 10: Fermat's last theorem

```
A 1..*
B 1..*
C 1..*
[ (#A × #A × #A) +(#B × #B × #B) = #C × #C × #C ]
```

$$scope = \left\{ \begin{array}{ccc} A & \rightarrow & 10 \\ B & \rightarrow & 10 \\ C & \rightarrow & 10 \end{array} \right\}$$

Meaning of true

How to interpret $\mathit{satisfiable}(\mathit{model}, \mathit{scope}) = \mathbf{true}$?

Perfect: accepts all intended instances.

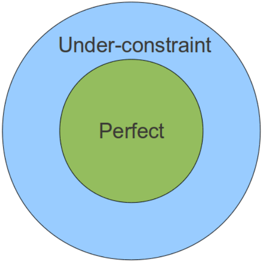


Figure: Satisfiable models

Meaning of true

How to interpret $satisfiable(model, scope) = \textbf{true}$?

Perfect: accepts all intended instances.

Under-constraint: accepts all intended instances plus more.

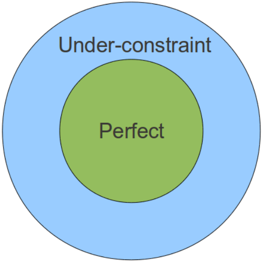


Figure: Satisfiable models

Under-constraint example

Clafer 11: Spouses model

```
abstract Person
    marriedTo → Person ?
    [this = marriedTo.marriedTo]

Alice : Person
Bob : Person
```

$$satisfiable(spouses\ model, scope) = \textbf{true}$$

Under-constraint example

Clafer 12: Spouses model

```
abstract Person
    marriedTo → Person ?
    [this = marriedTo.marriedTo]

Alice : Person
Bob : Person
```

$$\mathit{satisfiable}(\mathit{spouses\ model}, \mathit{scope}) = \mathbf{true}$$

But it's wrong.

Meaning of false

How to interpret $satisfiable(model, scope) = \textbf{false}$?

Over-constraint

Clafer 13: 1 = 2?

```
A : integer
[ A = 1 ]
[ A = 2 ]
[ A ≠ 0 ]
```

Meaning of false

How to interpret $satisfiable(model, scope) = \textbf{false}$?

Over-constraint

Clafer 15: 1 = 2?

```
A : integer
[ A = 1 ]
[ A = 2 ]
[ A ≠ 0 ]
```

Insufficient scope

Clafer 16: Not enough A

```
A 5..*
```

$$scope = \{A \rightarrow 4\}$$

Part III

ClaferIG

Model finding

$$satisfiable(model, scope) = \begin{cases} true & \exists x \in instances(model, scope) \\ false & otherwise \end{cases}$$

$$find(model, scope) = instances(model, scope)$$

Detecting under-constraint

Clafer 17: Spouses model

```
abstract Person
    marriedTo -> Person ?
    [this = marriedTo.marriedTo]

Alice : Person
Bob : Person
```

$$satisfiable(spouses\_model, scope) = \textbf{true}$$

$$find(spouses\_model, scope) = \ldots$$

Detecting under-constraint

Clafer 18: ClaferIG instance of spouses model

```
Alice
    marriedTo1 = Alice
Bob
    marriedTo2 = Bob
```

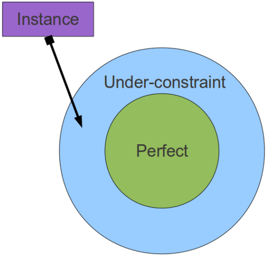


Figure: Proof of under-constraint

UNSAT

Clafer 19: 1 = 2?

```
A : integer
[ A = 1 ]
[ A = 2 ]
[ A ≠ 0 ]
```

$$satisfiable(model, scope) = \textbf{false}$$

Cannot simultaneously satisfy

- $[A = 1]$
- $[A = 2]$

UNSAT core

$$unsatCore(model, scope) = minimal(c)$$

where

$$c \subseteq constraints(model) \wedge unsatisfiable(c, scope)$$

Debugging over-constraint

Clafer 20: Over model

```
A : integer
[ A = 1 ]
[ A = 2 ]
[ A ≠ 0 ]
```

$$unsatCore(over\ model, scope) = \ldots$$

Clafer 21: ClaferIG UNSAT core of over model

```
The following set of constraints cannot be
satisfied in the current scope.
  1) A = 1
  2) A = 2
```

Fixes

Clafer 22: Over model

```
A : integer
[ A = 1 ]
[ A = 2 ]
[ A ≠ 0 ]
```

Clafer 23: ClaferIG fix for over model

```
Altering the following constraints produced
a counterexample.
  1) removed A = 1
A = 2
```

Scope

$$minimalScope(model) = minimal(scope) \land satisfiable(model, scope)$$

Clafer 24: Fermat's last theorem

```
A 1..*
B 1..*
C 1..*
[ (#A × #A × #A) +(#B × #B × #B) = #C × #C × #C ]
```

Scope heuristic

$$minimalScope(removeConstraints(model))$$

Clafer 25: Mesopotamian automotive systems

```
Chariot             --scope(Chariot) = 1
  Wheel 2..4        --scope(Wheel) = scope(Chariot) × 2 = 2
    Spoke 10..16 --scope(Spoke) = scope(Wheel) × 10 = 20
```

Scope dependency

Definition

B depends on A if the scope analysis of B requires the scope analysis of A.

Parent-child dependency

Clafer 26: Parents

```
A
    B
        C
    D

E
    F
```

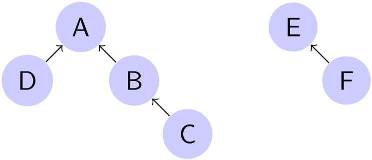


Figure: Dependency graph of parents model

Scope dependency

Subtype dependency

Clafer 27: Subtypes

```
abstract A
    B

C
    D : A 2..*
```

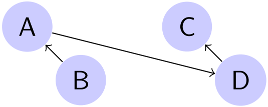


Figure: Dependency graph of subtypes model

Reference dependency

Clafer 28: References

```
A
    B

C
    D → A 2..*
```

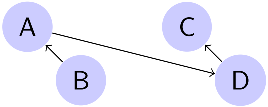
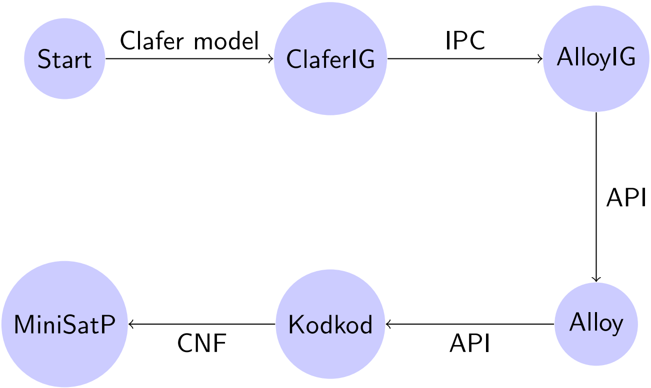


Figure: Dependency graph of references model

Implementation

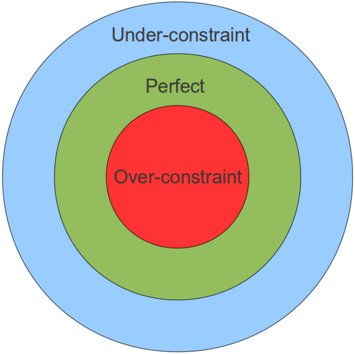


Part IV

Future work

Over-constraint satisfiable models

- Detecting under-constraint is easy
- Detecting over-constraint is hard



Smart fixes

Clafer 29: Three's company

```
Crowd
    Member → Person 3..*

abstract Person

Jack : Person
Janet : Person
```

Tighten scope

Analyze the constraints for a tighter bound.

Clafer 30: Processor feature

```
Processor
    VideoCard ?

[ some VideoCard ]
```

Isomorphism

Clafer 31: Leadership v1

```
Party
    Person1
        Age = 3
    Person2
        Age = 4
```

Clafer 32: Leadership v2

```
Party
    Person1
        Age = 4
    Person2
        Age = 3
```

Can ClaferIG detect isomorphic instances?

Canonical form

$$canonical(model1) = canonical(model2) \iff model1 \cong model2$$

$$canonical(/home/zan/../jayna) = /home/jayna$$
$$canonical(/home/jayna) = /home/jayna$$

canonical function for Clafer instances?

Open source

https://github.com/gsdlab/claferIG