Babelsberg
Unifying Declarative Constraints and Object-oriented Execution

Tim Felgentreff
Software Architecture Group
HPI Research School, UCT Workshop
April 14, 2014

An Extension for Object-oriented Programming Languages

```
always: { image.width * image.height *
            image.depth / 8 == image.data.length }
```

Why change a programming language?

Implicit assumptions inherent to programming

- A function works for certain types
- Memory is freed when it is no longer required
- Data structures in are consistent

Why change a programming language?

Implicit assumptions inherent to programming

- A function works for certain types
- Memory is freed when it is no longer required
- Data structures in are consistent

We can use guidelines and patterns to prevent problems

- Name your arguments appropriately
- Declare and allocate up front, free everything at the end
- Separate input checking and correcting from implementation concerns

Why change a programming language?

Implicit assumptions inherent to programming

- A function works for certain types
- Memory is freed when it is no longer required
- Data structures in are consistent

We can use guidelines and patterns to prevent problems

- Name your arguments appropriately
- Declare and allocate up front, free everything at the end
- Separate input checking and correcting from implementation concerns

Language extensions can support or enforce patterns

- (optional) type systems and checking
- automatic memory management
- ???

What is Babelsberg?

A design that integrates constraints with an object-oriented language. For example:

```
function drawBitmap(image) {

    always: { image.width * image.height *
                  image.depth / 8 == image.data.length }

    surface = new DisplaySurface(image.width,
                                 image.height,
                                 image.depth);
    for (i = 0; i < image.data.length; i++) {
        surface.setPixel(i, image.data[i]);
    }
}
```

What is Babelsberg?

How always different than assert?

```
always: { image.width * image.height *
                  image.depth / 8 == image.data.length }
```

vs

```
assert(image.width * image.height *
                  image.depth / 8 == image.data.length)
```

What is Babelsberg?

How always different than assert?

```
always: { image.width * image.height *
                  image.depth / 8 == image.data.length }
```

vs

```
assert(image.width * image.height *
                  image.depth / 8 == image.data.length)
```

Instead of failing, it describes what the state should look like and gives the system the chance to correct the image

Why change a programming language?

Implicit assumptions inherent to programming

- A function works for certain types
- Memory is freed when it is no longer required
- Data structures in are consistent

We can use guidelines and patterns to prevent problems

- Name your arguments appropriately
- Declare and allocate up front, free everything at the end
- Separate input checking and correcting from implementation concerns

Language extensions can support or enforce patterns

- (optional) type systems and checking
- automatic memory management
- ???

Why change a programming language?

Implicit assumptions inherent to programming

- A function works for certain types
- Memory is freed when it is no longer required
- Data structures in are consistent

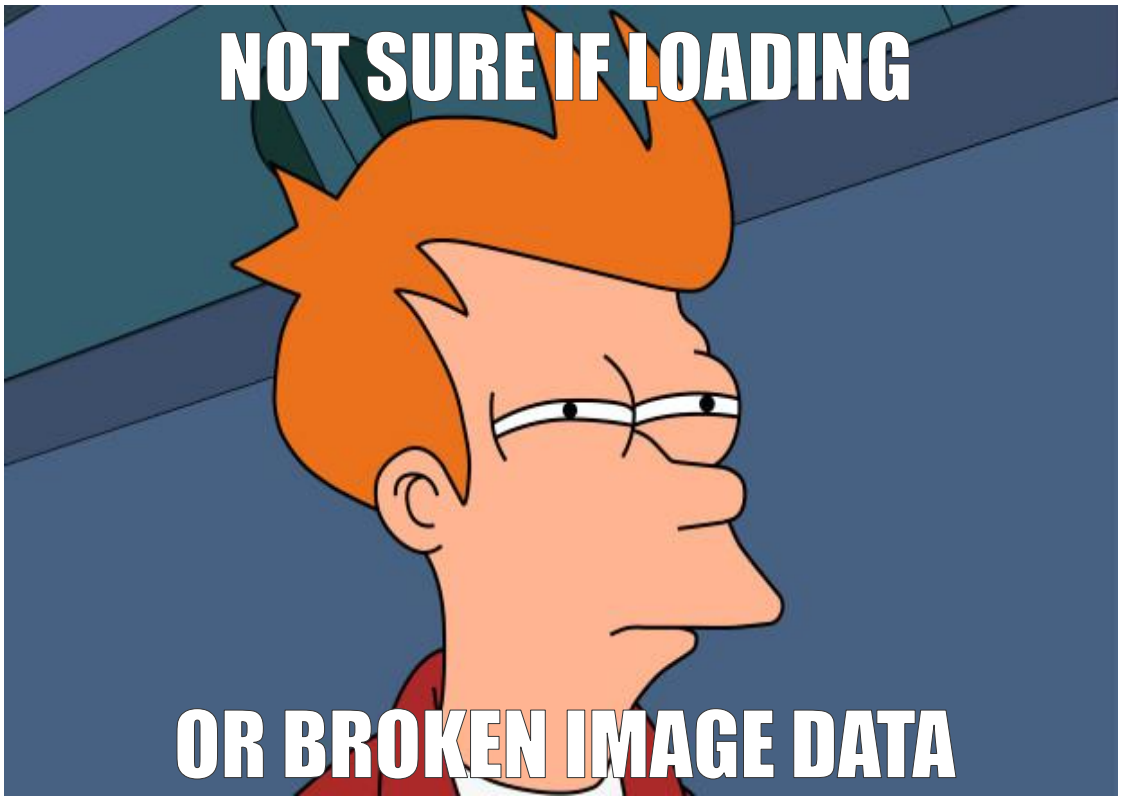
We can use guidelines and patterns to prevent problems

- Name your arguments appropriately
- Declare and allocate up front, free everything at the end
- Separate input checking and correcting from implementation concerns

Language extensions can support or enforce patterns

- (optional) type systems and checking
- automatic memory management
- optional system-maintained assertions – constraints

Integration of Constraints for OO Applications
Use-Case: An Image Viewer

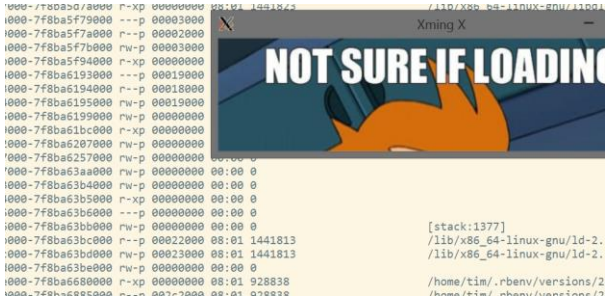


Integration of Constraints for OO Applications
Use-Case: An Image Viewer

- Imperative version may crash or display incomplete data when a broken (e.g. partially downloaded) image is passed (width * height * depth/8 != datasize)

```
image.width * image.height *
   image.depth / 8 == image.data.length
```

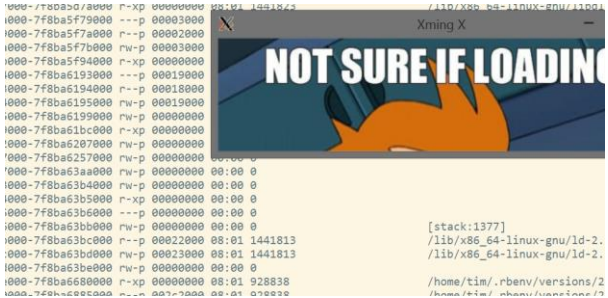
Integration of Constraints for OO Applications
Use-Case: An Image Viewer



- Imperative version may crash or display incomplete data when a broken (e.g. partially downloaded) image is passed (width * height * depth/8 != datasize)

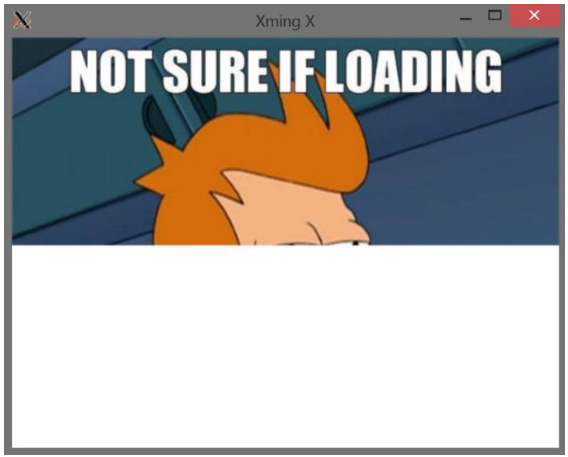
```
always: { image.width * image.height *
          image.depth / 8 == image.data.length }
```

Integration of Constraints for OO Applications
Use-Case: An Image Viewer



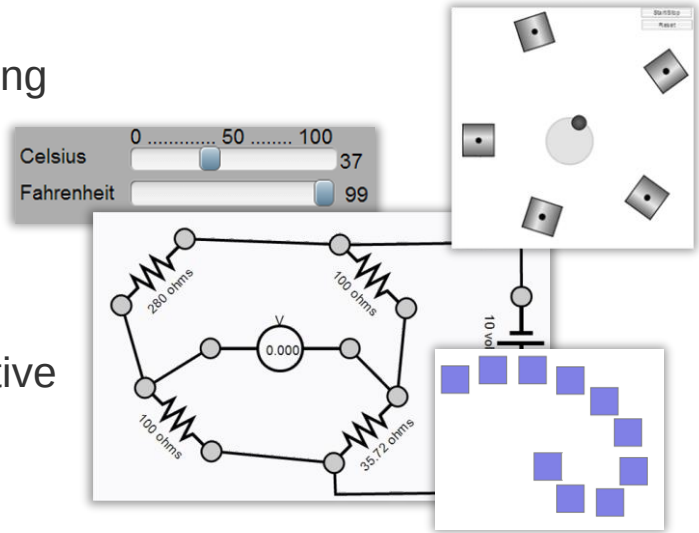
- Imperative version may crash or display incomplete data when a broken (e.g. partially downloaded) image is passed (width * height * depth/8 != datasize)

- The single line constraint allows the system to correct the incorrect image, and display as much data as there is



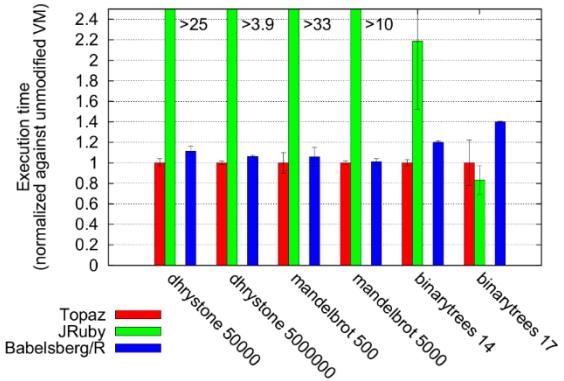
Integration of Constraints for OO Applications

- User Interface Layouting
- Circuit Simulation
- Data conversion and consistency in interactive applications
- Interactive animation



A Performant Implementation

- In related approaches, the performance impact of the language extension is many orders of magnitude. Current JIT optimization techniques can alleviate this problem.



Execution of purely OO code. Blue is Babelsberg/R
[Felgentreff, Borning & Hirschfeld 2014]

Reads	Objects	Constraints
1,000x	3.31s ± 0.289	8.57s ± 1.08
10,000x	20.4s ± 0.694	29.8s ± 1.82
100,000x	189s ± 5.83	241s ± 15.9

Access of pure OO object attributes versus constrained attributes [Felgentreff, Borning, Hirschfeld et.al. 2014]

Current Work: An Operational Semantics
Disclaimer

- I didn't like semantics
  - Formal = scary
  - I understand code better
- **But:** they really are not that scary!
- **And:** they are actually useful to really understand what is going on!

An Operational Semantics for Babelsberg

- Based on normal imperative semantics

$$\frac{E \vdash e \Downarrow v \quad E'(x) = v}{\langle E | x := e \rangle \longrightarrow \langle E' | skip \rangle}$$

An Operational Semantics for Babelsberg

- Successful assignment requires constraint solving

$$\frac{E \vdash e \Downarrow v \quad E' \vDash (C \wedge \mathrm{stay}(E) \wedge x = v)}{\langle E|C|x := e\rangle \longrightarrow \langle E'|C|\mathrm{skip}\rangle}$$

An Operational Semantics for Babelsberg

- But constraint solving can fail

$$\frac{E \vdash e \Downarrow v \quad \not\models (C \wedge stay(E) \wedge x = v)}{\langle E|C|x := e\rangle \longrightarrow \langle E|C|\mathrm{unsat}\rangle}$$

Summary

- Babelsberg is a backwards compatible integration of constraints into OO languages for added expressiveness
  - Allow the system to correct errors rather than fail
- The prototypes are performant and do not rely on VM modifications
  - It can be implemented like a library with no runtime cost for code that does not use constraints
- Support for a wide range of constraint solver-specific features
  - Solvers designed or optimized for specific domains can be added with little work

References

- Ingalls, D., Palacz, K., Uhler, S., Taivalsaari, A., & Mikkonen, T. (2008). The lively kernel a self-supporting system on a web page. In Self-Sustaining Systems (pp. 31-50). Springer Berlin Heidelberg.
- Sannella, M., Maloney, J., Freeman-Benson, B., & Borning, A. (1993). Multi-way versus one-way constraints in user interfaces: Experience with the DeltaBlue algorithm. Software: Practice and Experience, 23(5), 529-566.
- Borning, A., Freeman-Benson, B., & Wilson, M. (1992). Constraint hierarchies. LISP and symbolic computation, 5(3), 223-270.
- Badros, G. J., Borning, A., & Stuckey, P. J. (2001). The Cassowary linear arithmetic constraint solving algorithm. ACM Transactions on Computer-Human Interaction (TOCHI), 8(4), 267-306.
- Felgentreff, T., Borning, A., Hirschfeld, R., Lincke, J., Freudenberg, B., Ohshima, Y., Krahn, R. (2014). Babelsberg/JS - A Browser-based Implementation of an Object Constraint Language. In Proceedings of the 28th European Conference on Object-Oriented Programming (ECOOP). Springer (to appear)
- McCarthy, J. (1962). A basis for a mathematical theory of computation. Computer programming and formal systems, 33-70.

Babelsberg

- Babelsberg is a district of Potsdam, in which HPI is located
- The Tower of Babel was an attempt at building a tower to heaven by a humanity united under one language
- It also refers to Studio Babelsberg, the historical center of the German film industry where e.g. Fritz Lang's Metropolis was filmed

Line	Function	Statements	Lines	Comment Lines	Comment %	Branches	Depth	Cyclomatic Complexity	Halstead Volume	Halstead Potential	Program Level	MI
1	[[code]]	44	36	13	36.11%	0	0	1	0	0	0	
1	doStep	11	19	4	21.05%	1	1	2	211	4.75	0.0225	108.50
6	doStep.(Anonymous1)	2	12	3	25%	0	0	1	346	11.6	0.0335	113.49
22	movePiston	29	15	9	60%	6	2	7	831	8.00	0.00963	111.44

Line	Function	Statements	Lines	Comment Lines	Comment %	Branches	Depth	Cyclomatic Complexity	Halstead Volume	Halstead Potential	Program Level	MI
1	[[code]]	13	25	2	8%	0	0	1	0	0	0	
1	setupConstraints	8	21	2	9.52%	0	0	1	221	11.6	0.0525	104.97
7	setupConstraints.(Anonymous1)	0	3	0	0%	1	0	2	144	8.00	0.0556	135.75
17	setupConstraints.(Anonymous2)	1	4	0	0%	1	0	2	151	11.6	0.0768	130.92
24	step	1	2	0	0%	0	0	1	87.6	4.75	0.0542	144.24