http://maximaximal.pages.sai.jku.at/mbmv23/

Boolean Format Multitool for the Next Generation

Maximilian Heisinger

March 23, 2023

Institute for Symbolic AI, JKU Linz

Propositional Logic, SAT, QBF, and Beyond

Propositional Logic $a \wedge (b \vee c)$

Propositional Logic, SAT, QBF, and Beyond

Propositional Logic $a \wedge (b \vee c)$

SAT Finding solutions (assignments) for propositional logic

Propositional Logic, SAT, QBF, and Beyond

Propositional Logic $a \wedge (b \vee c)$

SAT Finding solutions (assignments) for propositional logic

QBF Extension with quantifiers $\forall$ and $\exists$

Propositional Logic, SAT, QBF, and Beyond

Propositional Logic $a \wedge (b \vee c)$

SAT Finding solutions (assignments) for propositional logic

QBF Extension with quantifiers $\forall$ and $\exists$

QSAT Finding solutions (certificates) for QBF

Propositional Logic, SAT, QBF, and Beyond

Propositional Logic $a \wedge (b \vee c)$

SAT Finding solutions (assignments) for propositional logic

QBF Extension with quantifiers $\forall$ and $\exists$

QSAT Finding solutions (certificates) for QBF

SMT More theories on top of SAT – more complex formulas, more complex solvers

Formula Formats

Propositional Logic Infix like `a & (b | c)`, LaTeX

Formula Formats

Propositional Logic Infix like `a & (b | c)`, LaTeX

SAT *DIMACS* (Conjunctive Normal Form, CNF)

Formula Formats

Propositional Logic Infix like `a & (b | c)`, LaTeX

SAT *DIMACS* (Conjunctive Normal Form, CNF)

QBF *QDIMACS* (CNF) and QCIR (trees of gates)

Formula Formats

Propositional Logic Infix like `a & (b | c)`, LaTeX

SAT *DIMACS* (Conjunctive Normal Form, CNF)

QBF *QDIMACS* (CNF) and QCIR (trees of gates)

SMT *SMT-LIB* (prefix-notation, Common-Lisp compatible), custom languages (CVC5's language, different Python APIs, etc.)

Formula Formats

Propositional Logic Infix like `a & (b | c)`, LaTeX

SAT *DIMACS* (Conjunctive Normal Form, CNF)

QBF *QDIMACS* (CNF) and QCIR (trees of gates)

SMT *SMT-LIB* (prefix-notation, Common-Lisp compatible), custom languages (CVC5's language, different Python APIs, etc.)

Ideas LaTeX?

From Ideas to Formulas to Solutions . . .

1. Limboole for putting ideas into propositional formulas

From Ideas to Formulas to Solutions . . .

1. Limboole for putting ideas into propositional formulas
2. PySAT to put propositional formulas (using Python) to CNF

From Ideas to Formulas to Solutions ...

1. Limboole for putting ideas into propositional formulas
2. PySAT to put propositional formulas (using Python) to CNF
3. Many custom shell-tools

From Ideas to Formulas to Solutions . . .

1. Limboole for putting ideas into propositional formulas
2. PySAT to put propositional formulas (using Python) to CNF
3. Many custom shell-tools
4. Many websites for educational use, reformatting, etc.. Dubios Correctness.

From Ideas to Formulas to Solutions . . .

1. Limboole for putting ideas into propositional formulas
2. PySAT to put propositional formulas (using Python) to CNF
3. Many custom shell-tools
4. Many websites for educational use, reformatting, etc.. Dubios Correctness.
5. Common-Lisp and others for processing SMT-LIB

. . . to More Problems

How to combine tools?

. . . to More Problems

How to combine tools?

How to transform formulas?

. . . to More Problems

How to combine tools?

How to transform formulas?

How to try out new encoding ideas?

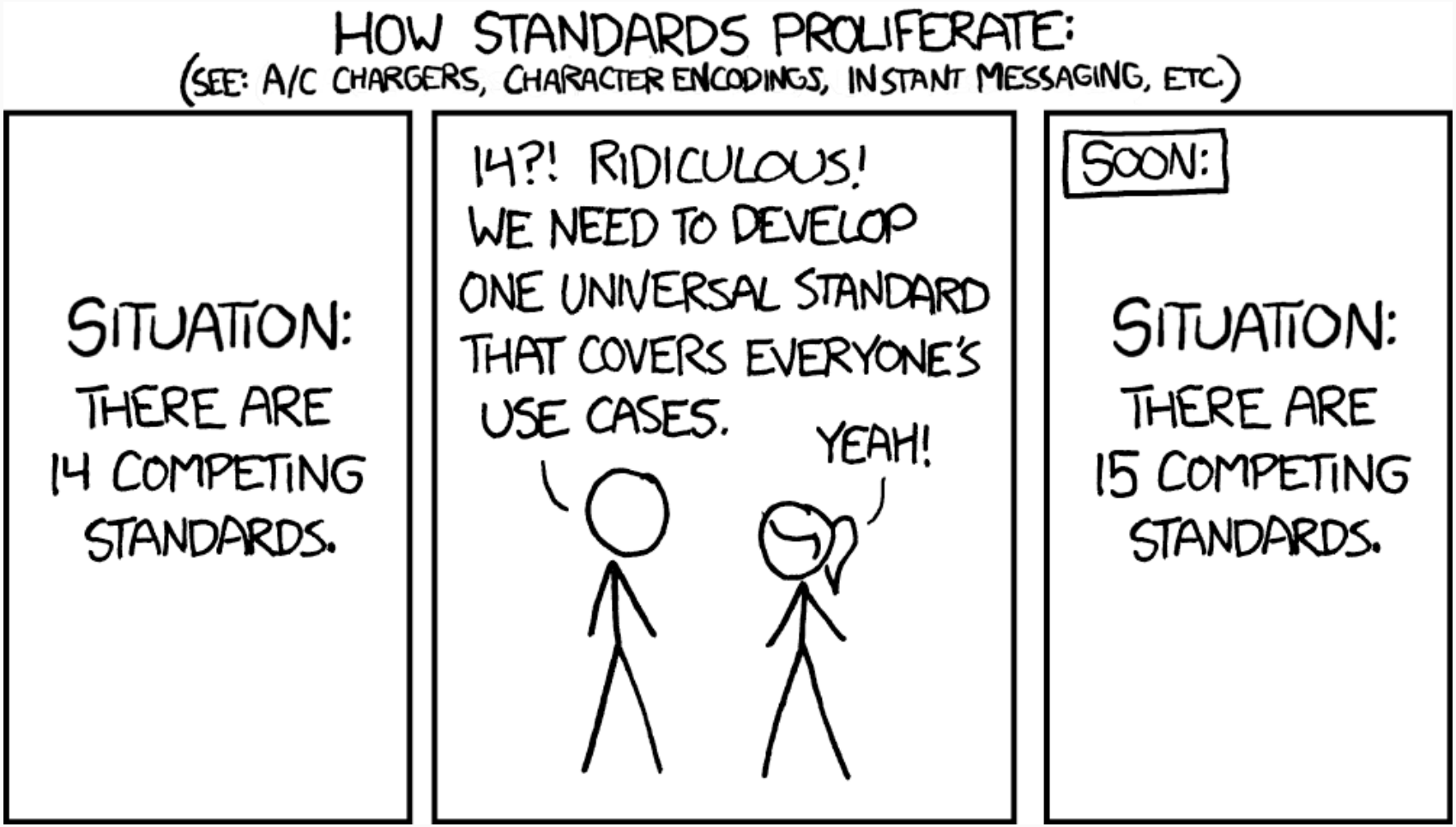
. . . to More Problems

How to combine tools?

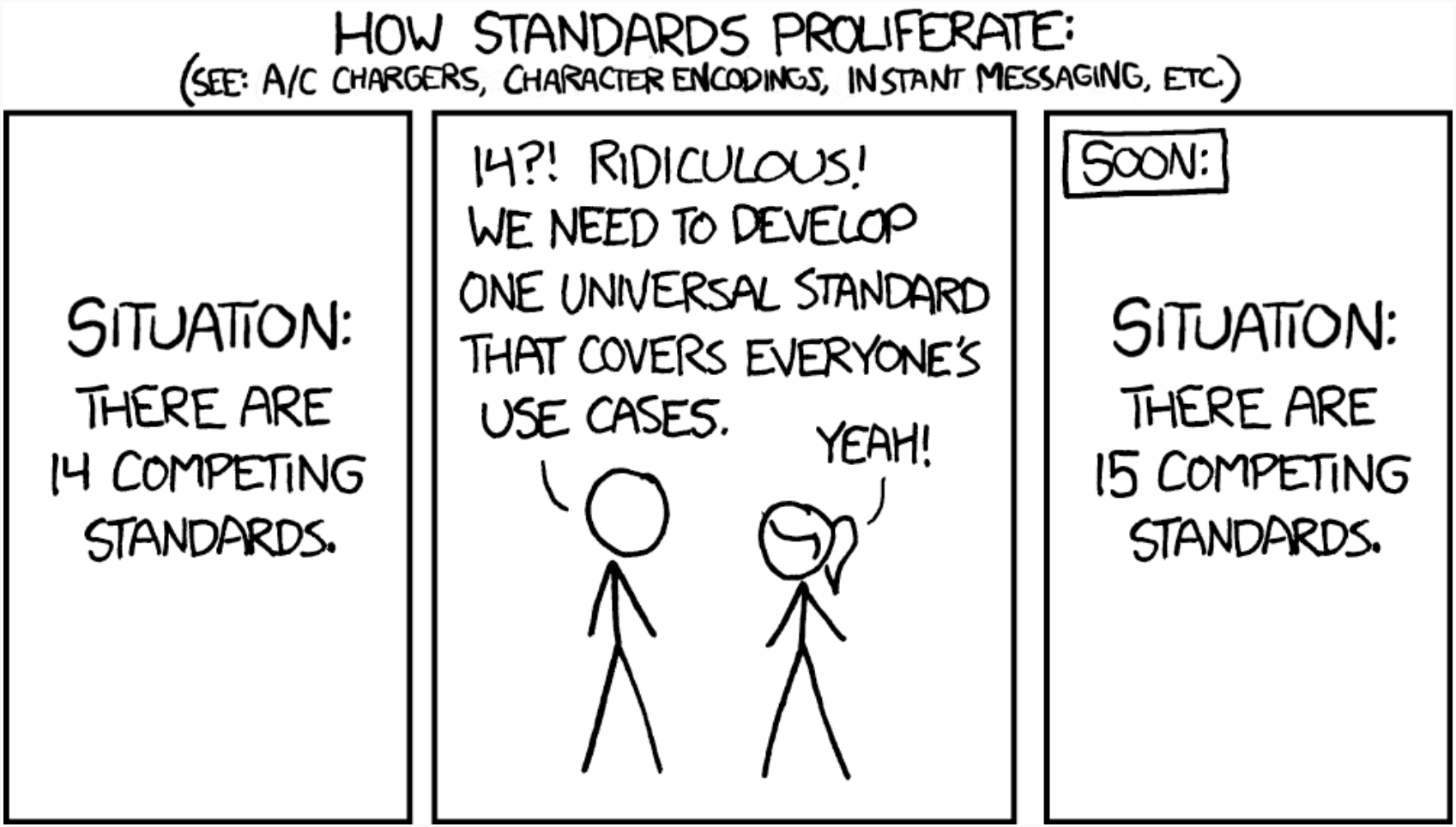
How to transform formulas?

How to try out new encoding ideas?

How to then use these new ideas somewhere else?


HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC.)
SITUATION:
THERE ARE
14 COMPETING
STANDARDS.
14?! RIDICULOUS!
WE NEED TO DEVELOP
ONE UNIVERSAL STANDARD
THAT COVERS EVERYONE'S
USE CASES.
YEAH!
SOON:
SITUATION:
THERE ARE
15 COMPETING
STANDARDS.

XKCD 927.



XKCD 927. Fortunately, the charging one has been solved now that we've all standardized on mini-USB. Or is it micro-USB? Shit.

Combining Standards

Let's create a conversion tool between standards with nice UX.

It worked for USB-C!

Boolean Formulas

The Next Generation

Booleguru

Goals:

1. Enough extension-points to allow adding new input and output formats 📝
    - Propositional Formulas à la Limboole ✓(Input & Output)
    - (Q)DIMACS ✓(Input & Output)
    - SMT-LIB 📝(Bit of Input)
    - QCIR 📝(Output)

Booleguru

Goals:

1. Enough extension-points to allow adding new input and output formats 
2. Efficient internal formula representation ✓
   - All nodes saved in `std::vector`
   - Built by hashing new nodes and finding already existing ones (using the *Ankerl* hash-map)
   - Internal references as 32-bit integers, smaller memory footprint than pointers
   - Nodes have available user-data fields

Booleguru

Goals:

1. Enough extension-points to allow adding new input and output formats 
2. Efficient internal formula representation ✓
3. Programmable and extendable ✓
    - Embedded *Lua* runtime and *fennel* (A Lua-based Lisp)
    - New operations can be defined in scripts, loaded at runtime
    - Composable in CLI and formulas
    - Parameterized formulas as functions for better formula re-use

Booleguru

Goals:

1. Enough extension-points to allow adding new input and output formats 
2. Efficient internal formula representation ✓
3. Programmable and extendable ✓
4. Nice Codebase for Collaboration ✓(?)
    - Everything split into well-separated internal sub-libraries
    - Extension-Points are clearly defined
    - Less toe-stepping (we hope)
    - Already collaborating SAI-internally on the tool

Booleguru

Goals:

1. Enough extension-points to allow adding new input and output formats 
2. Efficient internal formula representation ✓
3. Programmable and extendable ✓
4. Nice Codebase for Collaboration ✓(?)
5. User-friendly Web Interface 
    - Like Jupyter Notebook, but for Logic
    - Fully Client-side
    - Created by Alexander Mayr as a Bachelor's Thesis
    - Online at: logic.pizza
    - Including other cells is very WIP, rest is WIP.

Live Demonstration

Input Format Limboole-like: `a & !(b | c) -> d | (e <-> f)`

Embedded Code `a & f(v "b")` - uses Fennel-Lang (a Lua-Lisp)

Distribute to CNF `((a & b) | (c & d))`
`:(distribute-to-cnf *last-op*)`

Tseitin `((a & b) | (c & d)) :(tseitin *last-op*)`

Visualize `((?a #b a & b) | (?b #a a & b)) :(dotter *last-op*)`

Unquantified `(?a a & b) :(unquantified *last-op*)`

CLI Features

CLI-Support more advanced currently, supports loading Lua code from files.

`BOOLEGURU_LUA_PATH` adds load-paths to Lua.

Example:

```
export BOOLEGURU_LUA_PATH=lua
booleguru --eval formula.boole
booleguru f1.boole --and --not f2.dimacs --qdimacs
booleguru f1.boole --and f2.dimacs
':(*last-op*:rename "1" "a")'
```

Where to find Booleguru

gitlab.sai.jku.at/booleguru/booleguru/

logic.pizza

http://maximaximal.pages.sai.jku.at/mbmv23/

Boolean Format Multitool for the Next Generation

Maximilian Heisinger

March 23, 2023

Institute for Symbolic AI, JKU Linz