Safe, Purely Functional APIs to Low-Level Imperative Libraries: *A Programming Challenge*

Koen Lindström Claessen,
Chalmers University of Technology

WG2.8, Park City, Utah, June 2008

Motivating Example

- SAT-solvers …
- … solve SAT-problems
  - x v y v z
  - ~x v ~y
  - ~x v ~z
  - ~y v ~z

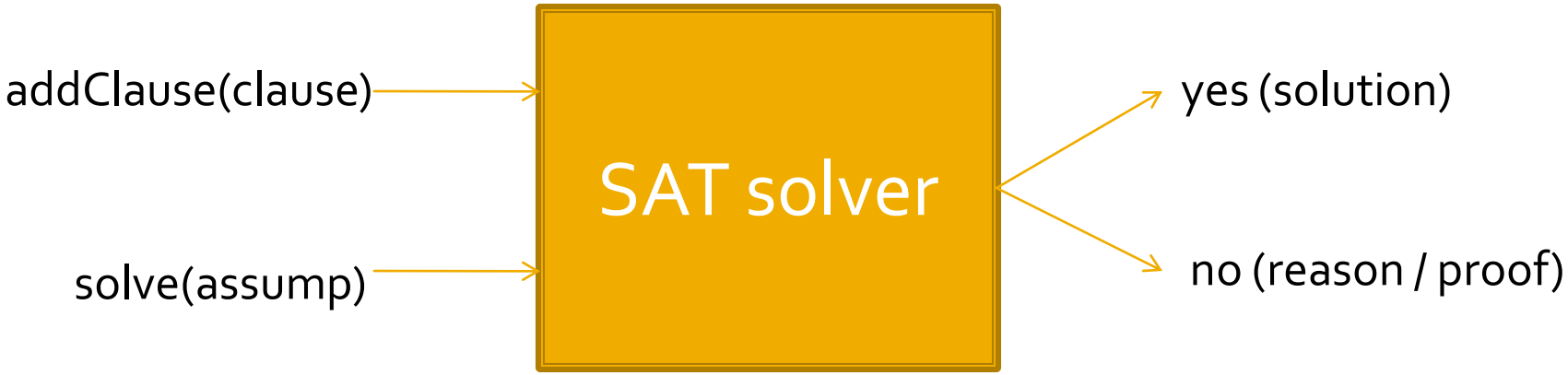
boolean variables

clauses: disjunctions of literals

a possible solution:
x=1, y=0, z=0

Modern SAT-solvers

- Dynamic software component
  - Incremental solving API
  - Feedback
  - Used in algorithms as sub-component



Lo

A Solver object

"MiniSat", implemented in C; Haskell API through FFI

A Literal

```
type Solver
type Lit

newSolver  :: IO Solver

newLit     :: Solver -> IO Lit
neg        :: Lit -> Lit

addClause  :: Solver -> [Lit] -> IO ()
solve      :: Solver -> [Lit] -> IO Bool

modelValue :: Solver -> Lit -> IO Bool
```

Problems With Low-Level API

- Mixing up Lits from different Solvers
  - Create a literal in one solver…
  - … use it in another solver
  - … use literals from different solvers in one clause
- Once in IO, you stay in IO
  - Calls to the API are imperative
  - …but the SAT-solver is deterministic
  - …and has no observable side effects
  - Want to create pure functions

The Challenge: Summing Up

- A low-level API
  - Creating unbounded number of "factory" objects
  - A factory can create reference objects…
  - … that are only valid if used with the original factory object
- The challenge
  - Design a method for building APIs that…
  - …avoids mixing reference objects from different factories
  - …with which pure functions can be created