### Satisfiability Modulo Theories (SMT)
- SMT is the problem of determining satisfiability of formulas modulo background theories.
- Examples of background theories:
  - linear arithmetic: \( x + 1 \leq y \)
  - arrays: \( a[i] = a[j] \)
  - uninterpreted functions: \( f(f(x)) = x \)
  - datatypes: \( \text{cons}(\text{null}, v) = v \)
  - bitvectors: \( \text{rotate}(b, b) = b \)
- Example of formula:
  \[
  i - j + 2, f(i + 3) \neq f(j + 6)
  \]

### Applications of SMT
- Extended Static Checking
- Equivalence Checking (Hardware)
- Bounded Model Checking (e.g., sal-inf-bmc)
- Predicate Abstraction
- Symbolic Simulation
- Test Case Generation (e.g., sal-atg)
- AI Planning & Scheduling
- Embedded in Theorem Provers (e.g., PVS)

### Yices
- Yices is an SMT Solver developed at SRI International.
- It is used in SAL, PVS, and CALO.
- It is a complete implementation of SRI’s previous SMT solvers.
- It has a new architecture, and uses new algorithms.
- Counterexamples and Unsatisfiable Cores.
- Incremental, push, pop, and retract.
- Weighted MaxSAT/MaxSMT.
- Supports all theories in SMT-LIB and much more.

### Supported Features
- Uninterpreted functions
- Linear real and integer arithmetic
- Extensional arrays
- Fixed-size bit-vectors
- Quantifiers
- Scalar types
- Recursive datatypes, tuples, records
- Lambda expressions
- Dependent types

### Using Yices
- Starting yices shell: `./yices -i`
- Batch mode:
  - Yices format: `./yices.ex1.ys`
  - SMT-LIB format: `./yices -smt.ex1.smt`
  - Dimacs format: `./yices -d ex1.cnf`
- Increasing verbosity level:
  - `./yices -v 3 ex1.ys`
- Producing models:
  - `./yices -e ex1.ys`

### Extracting Models
- `./yices -e ex3.ys`

<table>
<thead>
<tr>
<th>Define x::int</th>
<th>Define y::int</th>
</tr>
</thead>
<tbody>
<tr>
<td>Define f::(-&gt; int int)</td>
<td>Define g::(-&gt; int int)</td>
</tr>
<tr>
<td>Assert (= x (- y 4))</td>
<td>Assert (= (f k) (f i))</td>
</tr>
<tr>
<td>Check</td>
<td>Assert (= (f 0) 1)</td>
</tr>
</tbody>
</table>
- `- reset the logical context.
- `(status)` - display the status of the logical context.
- `(echo [string])` - prints the string `[string]`.

### Extracting Unsatisfiable Cores
- `./yices -e ex4.ys`

<table>
<thead>
<tr>
<th>Define f::(-&gt; int int)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Define i::int</td>
</tr>
<tr>
<td>Define j::int</td>
</tr>
<tr>
<td>Assert (= (+ i (* 2 k)) 10)</td>
</tr>
<tr>
<td>Assert (= (+ i (* 2 k)) 10)</td>
</tr>
<tr>
<td>Assert (= (+ i (* 2 k)) 10)</td>
</tr>
<tr>
<td>Check</td>
</tr>
</tbody>
</table>
- `- reset the logical context.
- `(status)` - display the status of the logical context.
- `(echo [string])` - prints the string `[string]`.

### Lemma Learning
- `Yices (and SAT) solvers have a search engine:
  - Case-split
  - Propagate
  - Conflict → Backtrack
  - Each conflict generates a Lemma:
    - It prevents a conflict from happening again.

### Retracting Assertions
- `Assumptions with `assert+` can be retracted.`
- `Yices knows which lemmas are safe to reuse.`
- `Yices knows which lemmas are safe to reuse.`
- `Yices knows which lemmas are safe to reuse.`

### Stacking logical contexts
- `(push)` - saves the current logical context on the stack.
- `(pop)` - restores the context from the top of the stack.

### Weighted MaxSAT
- `./yices -e ex5.ys`

| Assert (= (+ i (* 2 k)) 10) 10 |
| Assert (= (+ j (* 2 k)) 20) |
| Assert (= (f k) (f i)) |
| Assert (= (f k) (f i)) |
| Assert (= (f k) (f i)) |
| ([max-sat]) |
- `- reset the logical context.
- `(status)` - display the status of the logical context.
- `(echo [string])` - prints the string `[string]`.

---

**Leonardo de Moura** (joint work with Bruno Dutertre)

*Computer Science Laboratory*
*SRI International*
*Menlo Park, CA*

---

*SMT is the problem of determining satisfiability of formulas modulo background theories.*

*Examples of background theories:*

- **linear arithmetic:** \( x + 1 \leq y \)
- **arrays:** \( a[i] = a[j] \)
- **uninterpreted functions:** \( f(f(x)) = x \)
- **datatypes:** \( \text{cons}(\text{null}, v) = v \)
- **bitvectors:** \( \text{rotate}(b, b) = b \)

*Example of formula:*

\[
(i - j + 2, f(i + 3) \neq f(j + 6))
\]
Function (Array) Theory

- Yices (like PVS) does not make a distinction between arrays and functions.
- Function theory handles:
  - Function updates.
  - Lambda expressions.
  - Extensionality

Example:
```plaintext
./yices f1.ys
(define A1::(-> int int))
(define A2::(-> int int))
(define x::int) (define y::int)
(assert (= (update A1 x v) A2))
(assert (= (update A1 y w) A2))
(check) → unsat
```

Lambda expressions

Example:
```plaintext
./yices -e f2.ys
(define f::(-> int int))
(define x::int)
(define g::(-> (-> int int) int))
(assert (= f (g (lambda 0 2)))
(assert (= f (g (lambda 1 3)))
(check) → sat
```

Quantifiers

- Yices may return unknown for quantified formulas.
- The model should be interpreted as a “potential model”.
- The model is implemented as a “satellite theory”.

Fixed-size bit-vectors

- It is implemented as a satellite theory.
- Straightforward implementation.
- Simplification rules.
- Bit-blasting for all bit-vector operators but equality.
- “Bridge” between bit-vector terms and the boolean variables.

Example:
```plaintext
./yices -e bv.ys
(define b::(bitvector 4))
(assert (= b (bv-add 0b0010 0b0011)))
(check) → unsat
```

Recursive datatypes

- Similar to PVS and SAL datatypes.
- Useful for defining lists, trees, etc.

Example:
```plaintext
./yices dt.ys
(define-type list
  (datatype (cons car::int cdr::list) nil))
(define l1::list)
(define l2::list)
(assert (not (nil? l2)))
(assert (not (nil? l1)))
(assert (= (car l1) (car l2)))
(assert (= (cdr l1) (cdr l2)))
```

Dependent types

- Useful for stating properties of uninterpreted functions.
- Alternative to quantifiers.

Example:
```plaintext
./yices -e d.ys
(define x::real)
(define y::int)
(define floor::(-> x::real (subtype (r::int) (and (>= x r) (< x (+ r 1)))))
(assert (and (> x 5) (< x 6)))
(assert (= y (floor x)))
(check) → sat
```

C API

- Yices distribution comes with a C library.
- Two different APIs:
  - yices.h
  - yices.c.h (Lite version).

Quantifiers: example

Example:
```plaintext
./yices q.ys
```

Conclusion

- Yices is an efficient and flexible SMT solver.
- Yices supports all theories in SMT-LIB and much more.
- It is being used in SAL, PVS, and CALO.
- Yices is not ICS.
- Yices is freely available for end-users.
  - http://yices.csl.sri.com
- Supported Platforms:
  - Linux
  - Windows: Cygwin & MinGW
  - Mac OSX