Current State of ABS

Rudi Schlatte, UiO

May 28, 2018

http://www.sirius-labs.no
Last year . . .

- Semi-regular releases
  - Broke some things, need better release checklist
- Test Suite is approaching usability (down to 9 failures)
  - Continuous integration soon
- More contributors and users
- Website: needs to be re-done, with existing tutorials transferred
Breaking Changes

• Removed automatic \texttt{Rat} \rightarrow \texttt{Int} truncation (v1.5.0)
Features and Fixes

• Stdlib: `takeMaybe` (v1.5.0), `elements` (v1.5.2), `entries` (v1.5.3)
• `foreach` statement (v1.5.1)
• Parameters for exceptions (v1.5.2)
• Deadlines, user-defined schedulers for Erlang backend (v1.5.2)
• Second-order functions and anonymous function parameters (v1.5.3)
  • `map`, `filter`, `foldl`, `foldr` (v1.5.3)
• Lists as input parameters for model api (v1.5.4)
• Floating-point numbers (master)
Plans for next year

- Zero-failure test suite, continuous integration set up
- Web site and tutorials remade
- Some tool integrations
- Some new features
Repeatability

- Output compiler version from `absc --version`
- Output compiler version from compiled model, e.g., `gen/erl/run -v` (v1.5.3)
- Wanted: a way of packaging simulation data with model
• Add data to models
  • Repeatable data-driven models: need version of compiler (done), run with same data
  • Give data file as compile-time parameter, embed in model
  • “Fake” function: read data file line-by-line

• Also: Integrate custom visualization (index.html + JS libraries) with model compilation

• All this is possible already, but in a hacky way
Adding data to models

[External] def List<Int> f() = [1, 2, 3];

absc model.abs --external f=./data.csv

- For static analysis / development: have sample data in model
- For simulation: read (possibly big) data file when starting model
Publish the tool suite?

- JOSS (http://joss.theoj.org/)
  - check authorship of code
  - clarify licenses
  - Remove dead code
  - Make test suite pass

- Cons:
  - The citation refers to a specific version of the toolchain
  - We might be out of scope – but all required changes are beneficial anyway
Backwards-incompatible changes

- ? remove null ?
- ? Run main blocks for all modules ?
- ? Make new asynchronous ? (new local stays synchronous)

```typescript
Fut<I> fo = new C();
await fo?;
I o = fo.get;
```
```typescript
I o = await new C();
```
- Remove the current eclipse plugin
Infrastructure work

- Other frameworks (Ecore, Rasqal)?
  - JastAdd works so far but is slightly non-deterministic
  - It’s possible to do step-by-step migration to new framework
New features

- module-level constants
  - currently done via parameterless functions
- Return value for main block
  - sets shell error code
- throw expression, try / catch expression
- operator overloading?

plus everything in github project “new features”
Thank you!