

Plug-in compilers and (Embedded) Domain Specific Languages

Sean Seefried

sseefried@cse.unsw.edu.au

4 February, 2005



Problem

Problems in language implementations

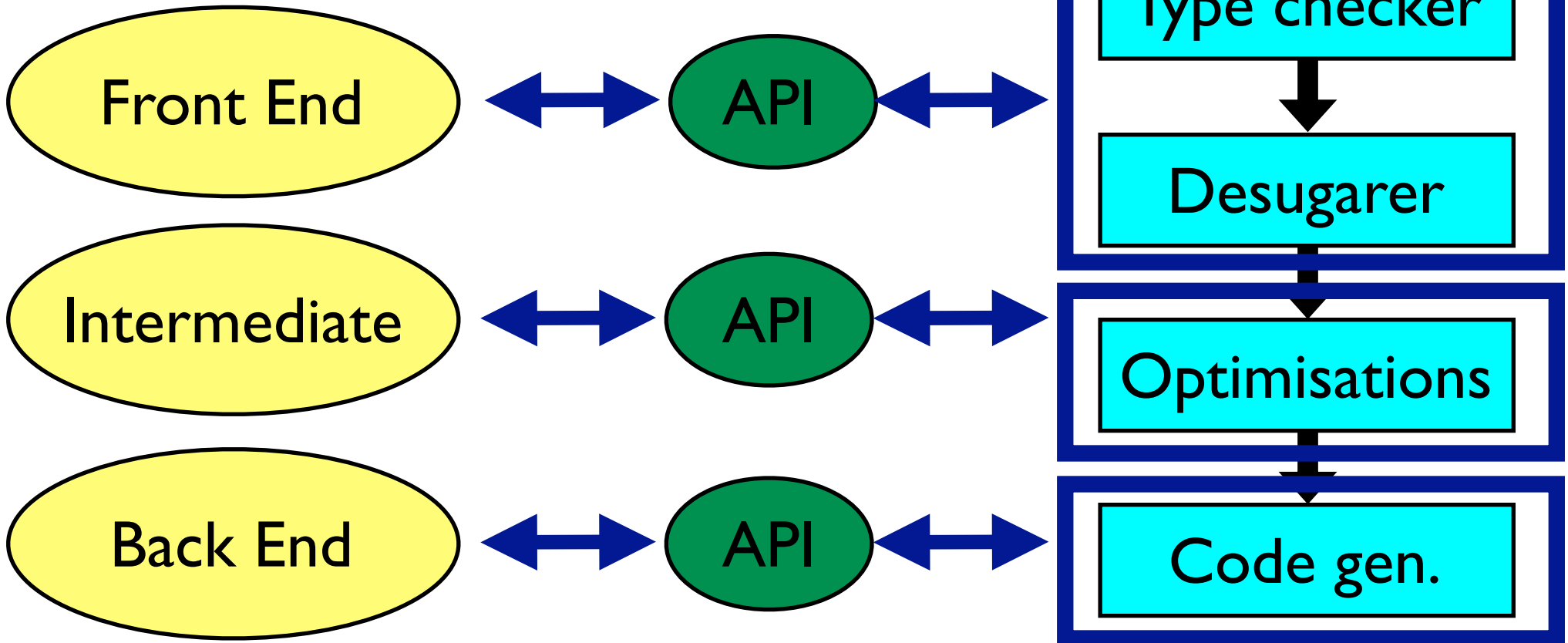
- Language variants that are almost the same
 - triggered by command line switches in a compiler
 - even worse - re-implemented from scratch
- DSLs, although great, are expensive to write

Monolithic nature of compilers is mostly to blame

- too much “baked in”
- not extensible by users

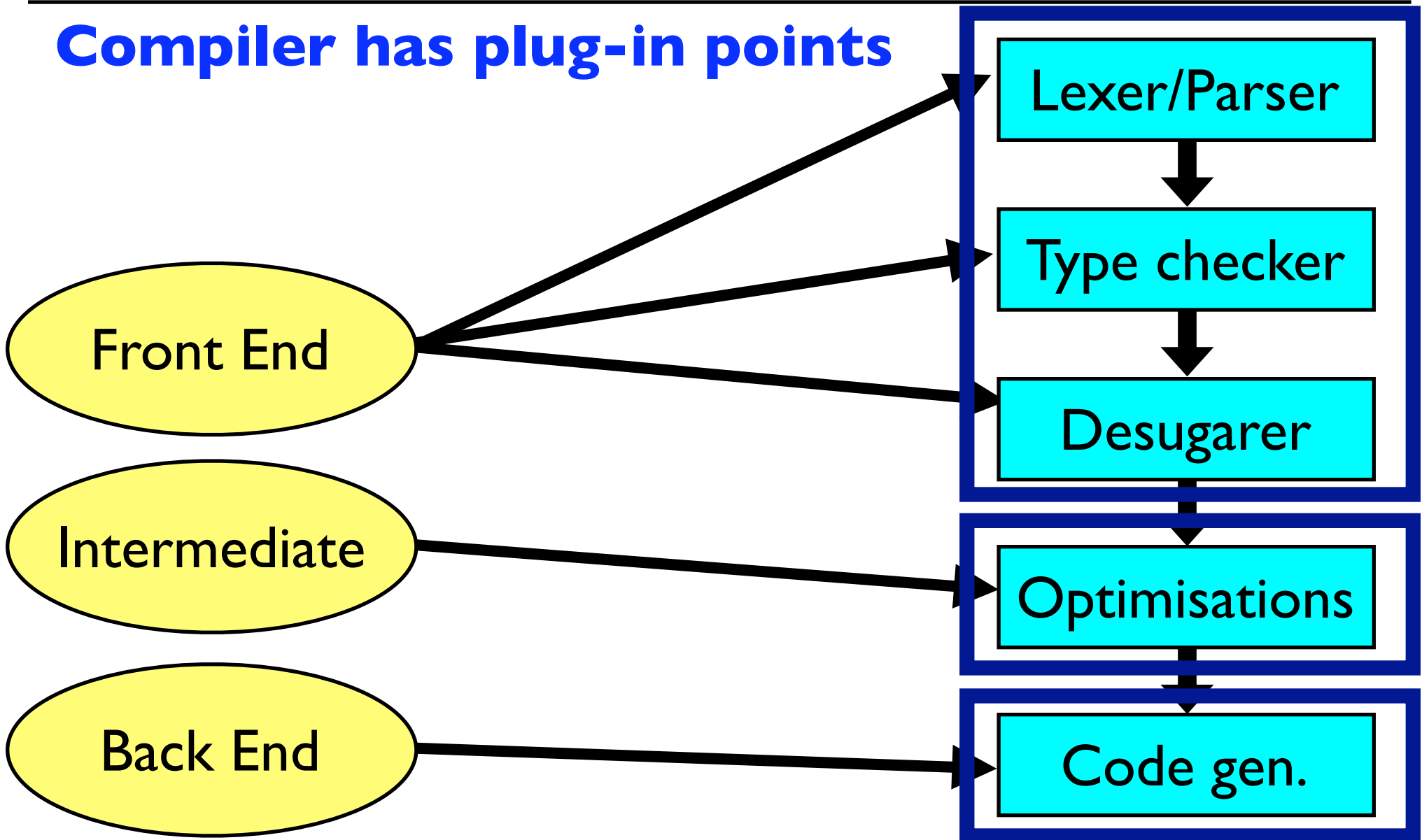
Solution: plug-in compiler

API exposes internals of compiler



Solution: plug-in compiler

Compiler has plug-in points



Solution: plug-in compiler

Each has a different purpose

Front End

Syntax extension, syntactic analysis

Intermediate

Optimisations, translation to diff.
intermediate langs

Back End

Code generation

Obvious implications

Benefits

- Smaller code base
 - Quicker build times
- Core language separated from “syntactic sugar”
- Syntax extension
- Optimisations
 - custom made for data structures
 - can be inserted at *any* point within pipe-line
 - not necessarily *universally* applicable
- Variants of language can stay more closely in step

Wider implications

Unexpected directions

- Reasoning easier.
 - Holy grail: verifiable compilers
- Optimisations distributed *with* libraries
- Micro-compiler
 - no front-end
 - no back-end
- Extend compiler with new intermediate levels
- Remove and add plugins during compile-time
 - based on profiling

A new perspective on DSLs

Compiler + set of plugins = (E)DSL

Challenges

Hairy areas

- dependencies between:
 - plugins
 - optimisations. How do you thread analysis info?
- API
 - What exactly do you expose?
 - What do you hide?
 - How do you achieve re-use?

Progress: prototype in GHC

Serendipity strikes

- “package ghc”
 - GHC compiled as a package = crude API
- GHCi = rudimentary plug-in infrastructure
- hs-plugins - Don Stewart
 - provides plug-in capabilities to application writers
 - leverages GHC in much the same way

Optimisation plug-in point

- has been added at point Core-to-Core passes are done
- Core = GHC’s intermediate language.
 - Lambda calculus + types + data types
- Works for simple examples. No reason for large ones to fail.

Additional progress

Meta-programming

Improves code quality of EDSLs

- Teach compiler algebraic properties of DSL via source-to-source transformation
- Used Template Haskell
 - Compile-time meta-programming extension

Example: Pan DSL

- for synthesis of interactive images/animations
- Pan had already been implemented as *embedded compiler*

Meta-programming

Successful

- 3000 LOC vs. 13000 LOC
- At least a two-fold speed-up on all examples.
- Quasi-quote syntax = elegant syntax

... but with problems

- Types needed for correct/complete optimisations
- Optimisations *before* compiler's opts
- ... and on full Haskell (not Core or other intermediate rep.)

And

- Not as good performance as original Pan implementation

Progress summary

Progress

- “Optimising Embedded DSLs using Template Haskell”.
Published and presented at GPCE’04. Primary author.
- “Plugging Haskell In” published at Haskell Workshop 2004.
Co-author
 - Added plug-in support to Pan
- Started type checking extension to TH
- Devil - an IDL for device driver programming
 - as EDSL
 - Now working on *rich data types*

Related Work

Meta-programming

- Todd Veldhuizen on Blitz++ using C++ templates
 - templates were never designed for meta-programming
- John O'Donnell on Hydra using Template Haskell
 - Syntax extension not optimisation

Source-to-source transformation

- Martin Bravenboer on Metaborg. SDF/Stratego.
 - language independent!

Plug-in compiler

- Dawson Engler on MAGIK. Extended **lcc**.
 - Front-end plugins ignored. Misses key insight.

Future work

Next six months

- Implement & benchmark Pan using core-to-core plugins
 - implementation (14 April)
 - benchmarks (14 May)
- Investigate
 - de-coupling *syntactic sugar* from compiler (1 June)
 - syntax extensions (1 July)
 - type checking extensions (31 July)
- First three chapters of thesis (1 April, 1 May, 1 June)

Following six months

- Write up rest of thesis. (3 - 6 months)

The End

Questions?