Analysis of F# programs

Péter Diviánszky, Mónika Mészáros, Gábor Páli
E-mail: divip@aszt.inf.elte.hu, {bonnie,pgj}@inf.elte.hu

Dept. of Programming Languages and Compilers, Fac. of Informatics, ELTE

May 27, 2009

- Metrics for F# programs
- Lightweight resource analysis in F#
- Efficient implementation of data structures in F#
- Refactor in F# programs
- New approach of handling documentary structure in F# programs

- Metrics for F# programs
- Lightweight resource analysis in F#
- Efficient implementation of data structures in F#
- Refactor in F# programs
- New approach of handling documentary structure in F# programs

- Metrics for F# programs
- Lightweight resource analysis in F#
- Efficient implementation of data structures in F#
- Refactor in F# programs
- New approach of handling documentary structure in F# programs

- Metrics for F# programs
- Lightweight resource analysis in F#
- Efficient implementation of data structures in F#
- Refactor in F# programs
- New approach of handling documentary structure in F# programs

- Metrics for F# programs
- Lightweight resource analysis in F#
- Efficient implementation of data structures in F#
- Refactor in F# programs
- New approach of handling documentary structure in F# programs

- Motivation
- Our proposal
- Advantages
- Description of the model

- Motivation
- Our proposal
- Advantages
- Description of the model

- Motivation
- Our proposal
- Advantages
- Description of the model

- Motivation
- Our proposal
- Advantages
- Description of the model

Motivation

- Documentary structure: comments and whitespace in source code

- Source code is primarily created for humans to read, and not for machines to compile
- The documentary structure of the source code should be preserved during program transformations if the produced source is for human consumption

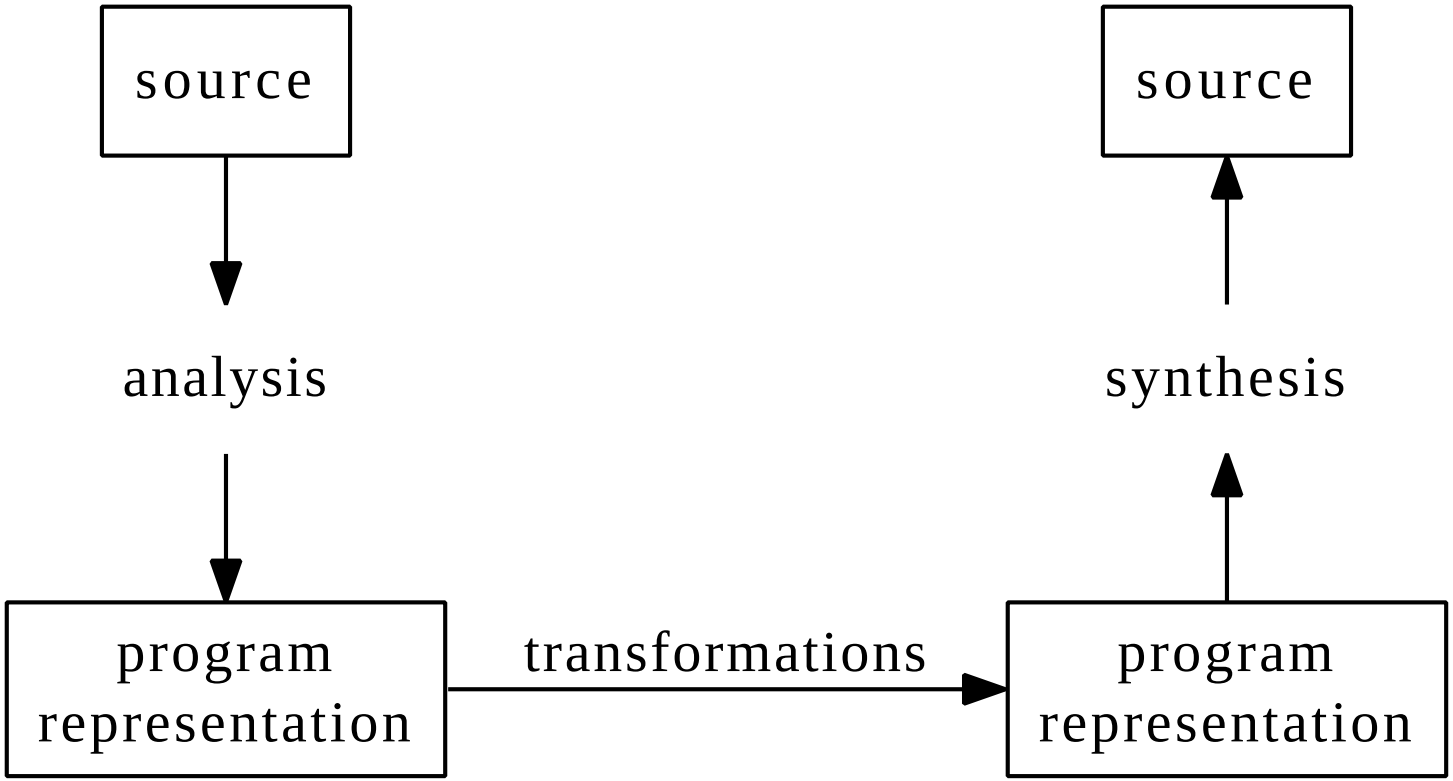
Motivation

- Documentary structure: comments and whitespace in source code
- Source code is primarily created for humans to read, and not for machines to compile
- The documentary structure of the source code should be preserved during program transformations if the produced source is for human consumption

Motivation

- Documentary structure: comments and whitespace in source code
- Source code is primarily created for humans to read, and not for machines to compile
- The documentary structure of the source code should be preserved during program transformations if the produced source is for human consumption

Usual approach



Our proposal

- The documentary structure is detached before the transformations
- and it is re-attached after the transformations
- The detached information is stored in layers
- The reconstruction is lossless

Our proposal

- The documentary structure is detached before the transformations
- and it is re-attached after the transformations
- The detached information is stored in layers
- The reconstruction is lossless

Our proposal

- The documentary structure is detached before the transformations
- and it is re-attached after the transformations
- The detached information is stored in layers

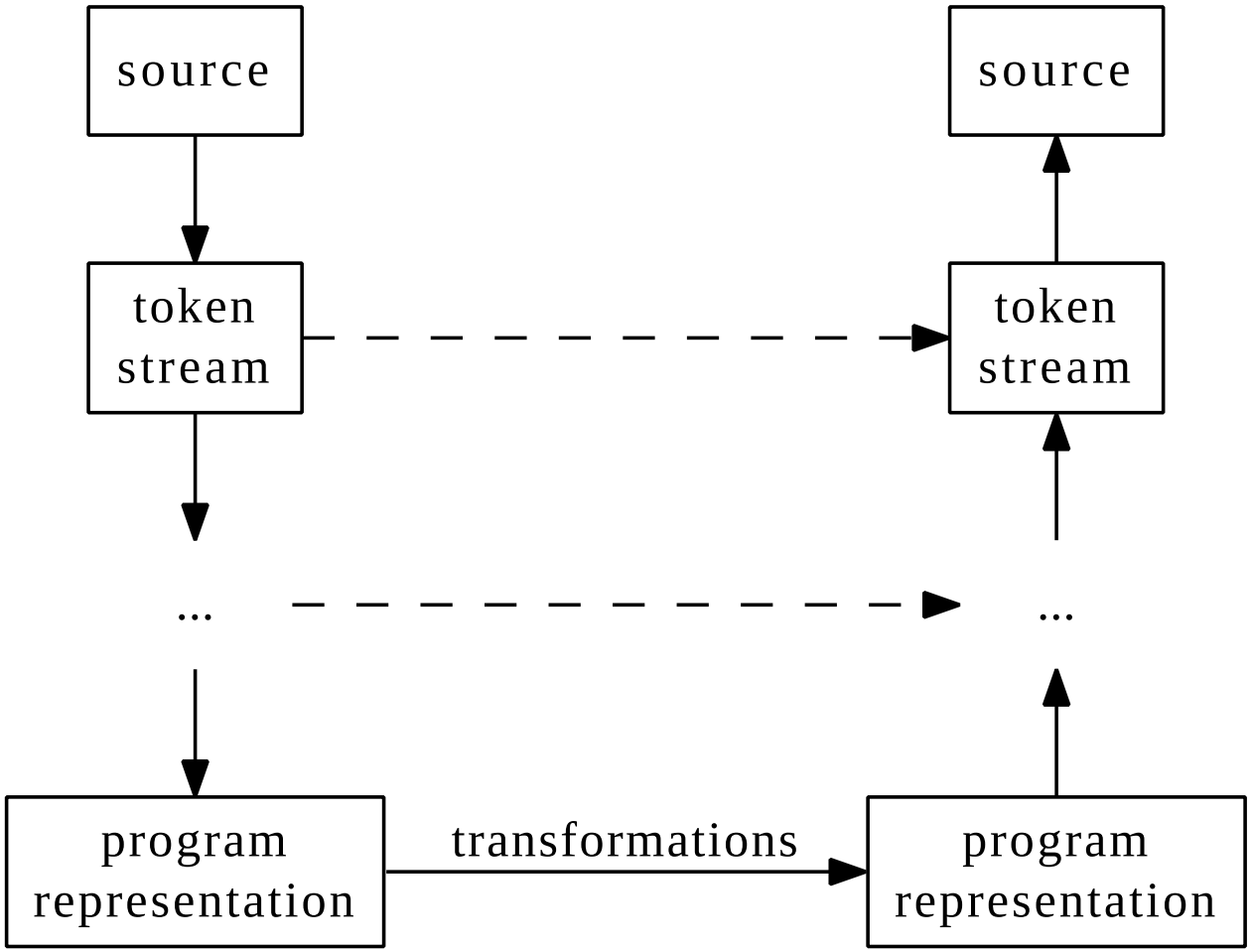
- The reconstruction is lossless

Our proposal

- The documentary structure is detached before the transformations
- and it is re-attached after the transformations
- The detached information is stored in layers

- The reconstruction is lossless

New approach



Advantages

- Program transformations are easier to be described if the documentary structure may be discarded
- Is sufficient to process only the actual layer
- This representation is also suitable for analysing and refactoring the documentary structure of the source code

Advantages

- Program transformations are easier to be described if the documentary structure may be discarded
- Is sufficient to process only the actual layer
- This representation is also suitable for analysing and refactoring the documentary structure of the source code

Advantages

- Program transformations are easier to be described if the documentary structure may be discarded
- Is sufficient to process only the actual layer
- This representation is also suitable for analysing and refactoring the documentary structure of the source code

Layers

- Token formattings
- Comments
- Layout

- Further layers can be defined

Layers

- Token formattings
- Comments
- Layout

- Further layers can be defined

Layers

- Token formattings
- Comments
- Layout

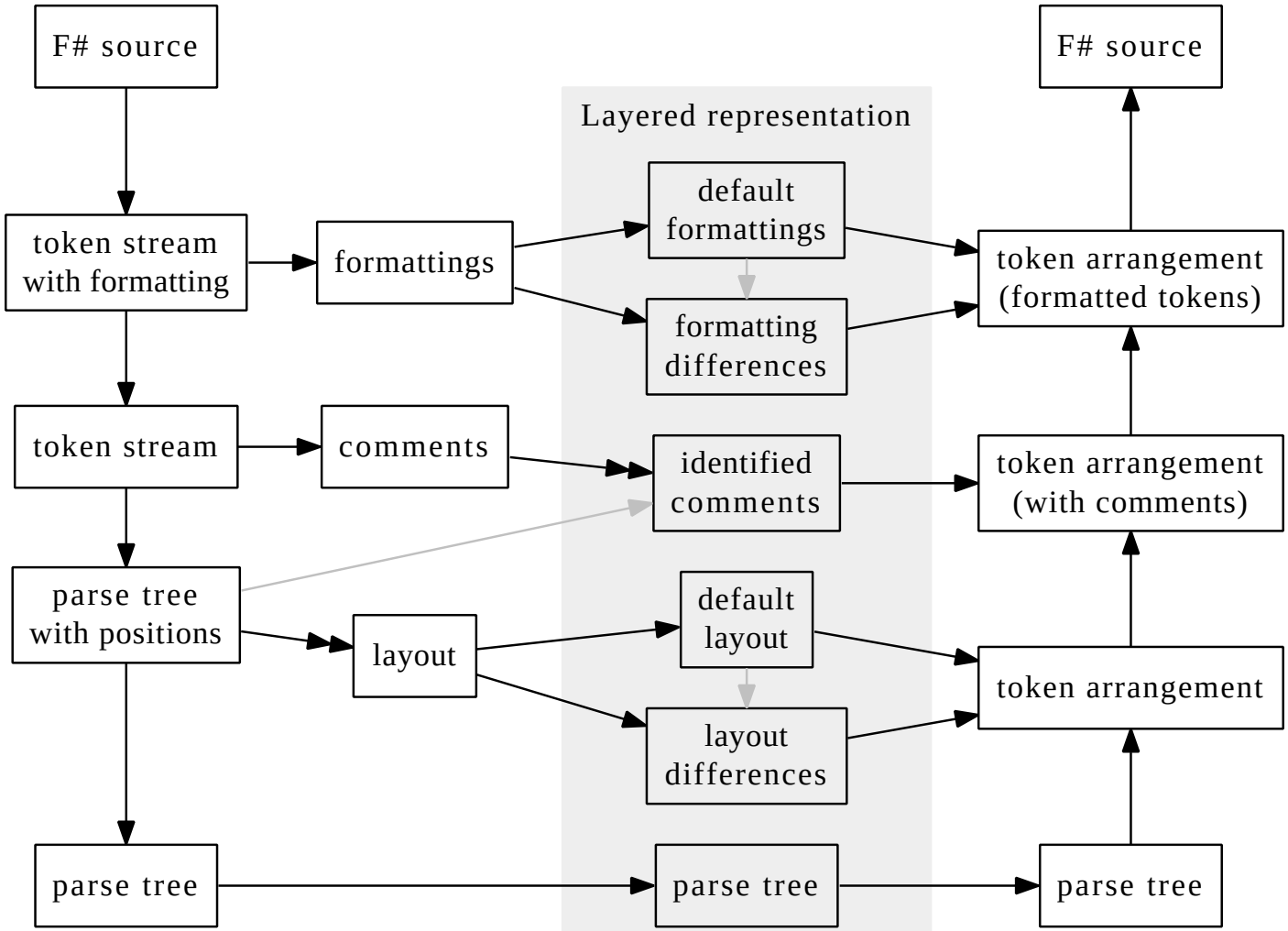
- Further layers can be defined

Layers

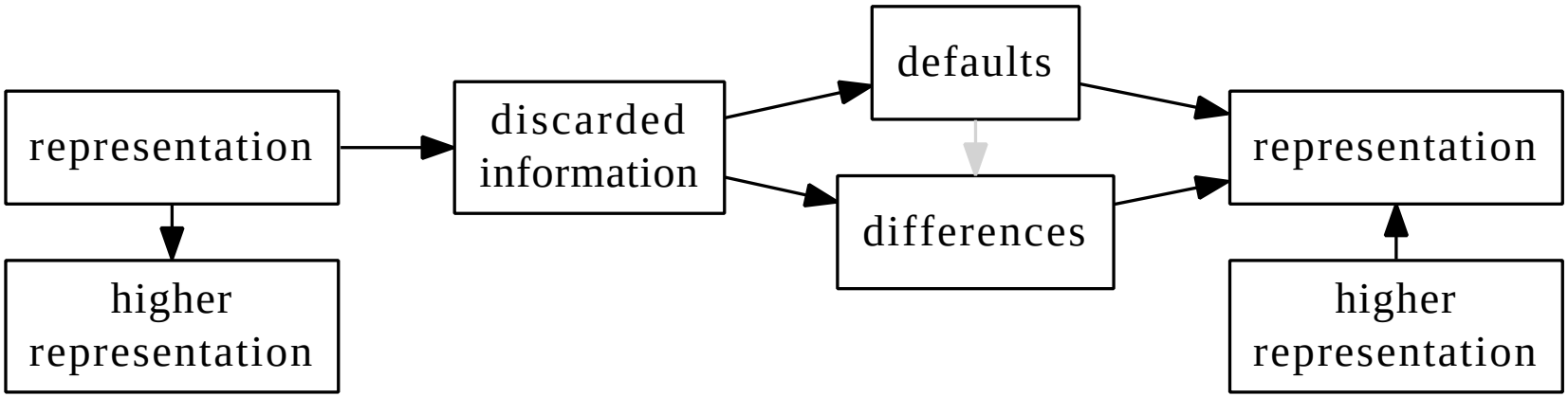
- Token formattings
- Comments
- Layout

- Further layers can be defined

The detailed model

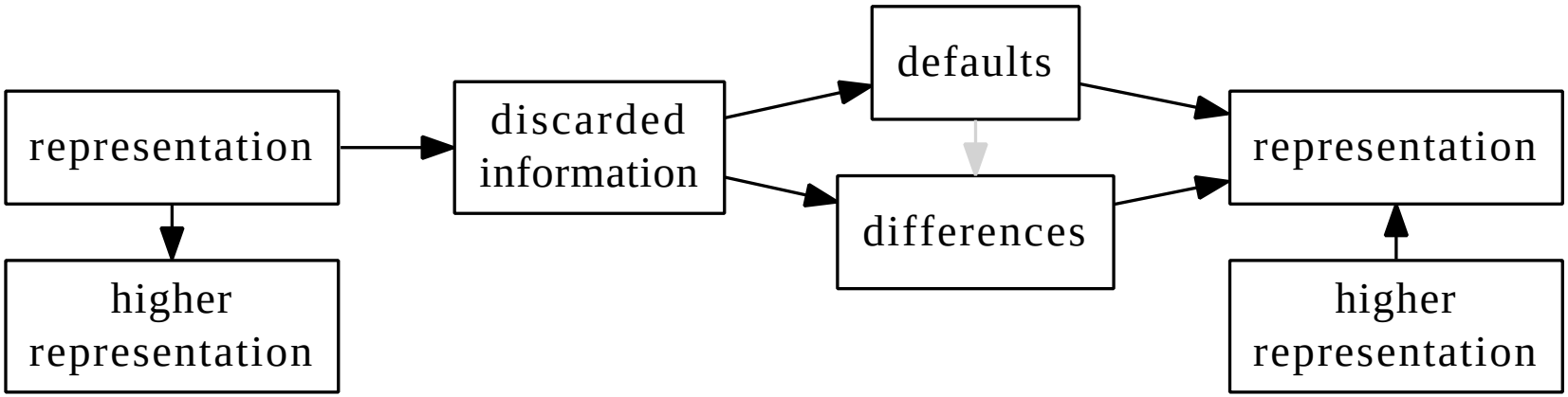


One layer



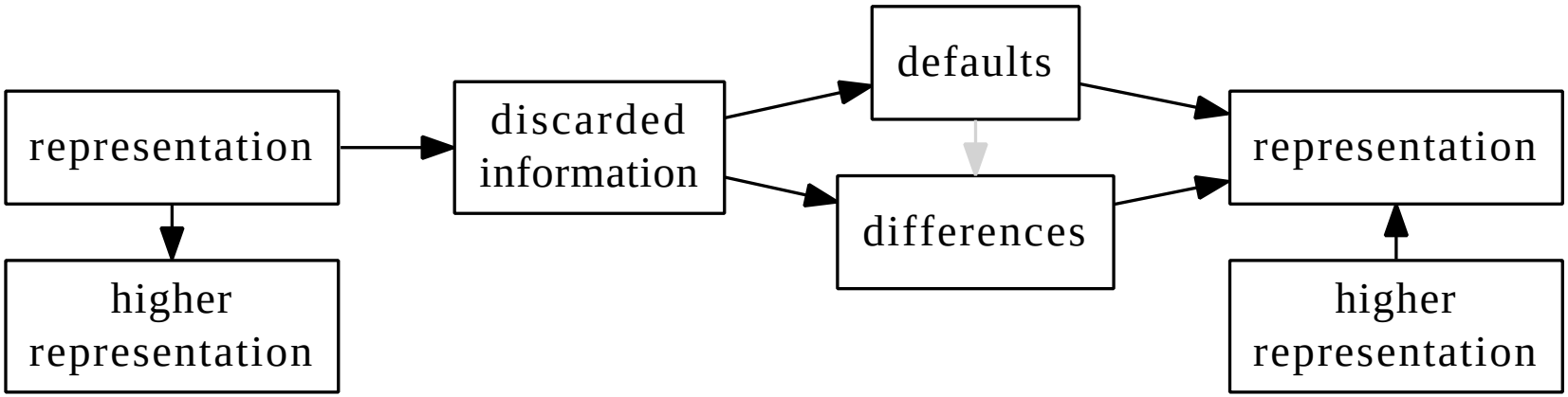
- Defaults
- Differences

One layer



- Defaults
- Differences

One layer



- Defaults
- Differences

Advantages in practice

- Standardize the layout of a source code
- The code after refactoring will be in the default style
- The implementation of program transformation steps is easier
- Refactor in the comments

Advantages in practice

- Standardize the layout of a source code
- The code after refactoring will be in the default style
- The implementation of program transformation steps is easier
- Refactor in the comments

Advantages in practice

- Standardize the layout of a source code
- The code after refactoring will be in the default style
- The implementation of program transformation steps is easier

- Refactor in the comments

Advantages in practice

- Standardize the layout of a source code
- The code after refactoring will be in the default style
- The implementation of program transformation steps is easier

- Refactor in the comments

Homepage of our project:
`http://plc.inf.elte.hu/fsharp/`