Automated Module Interface Upgrade

László Lövei

Department of Programing Languages and Compilers
Faculty of Informatics
Eötvös Loránd University

Erlang Workshop 2009

Outline

Regexp Upgrade Example

```
find(Str) ->
    case regexp:match(Str, ?RE) of
        {match, Start, Len} ->
            strings:substr(Str, Start, Len);
        nomatch ->
            ""
    end.
```

Regexp Upgrade Example

```
find(Str) ->
    case regexp_match(Str, ?RE) of
        {match, Start, Len} ->
            strings:substr(Str, Start, Len);
        nomatch ->
            ""
    end.

regexp_match(S, R) ->
    case re:run(S, R, [{capture,first}]) of
        {match, [{St, Ln}]} -> {match, St+1, Ln};
        nomatch             -> nomatch
    end.
```

Regexp Upgrade Example

```
find(Str) ->
    case case re:run(Str, ?RE, [{capture,first}]) of
            {match, [{St, Ln}]} -> {match, St+1, Ln};
            nomatch             -> nomatch
         end of
        {match, Start, Len} ->
            strings:substr(Str, Start, Len);
        nomatch ->
            ""
    end.
```

Regexp Upgrade Example

```
find(Str) ->
    case re:run(Str, ?RE, [{capture,first}]) of
        {match, [{Start, Len}]} ->
            strings:substr(Str, Start+1, Len);
        nomatch ->
            ""
    end.
```

Regexp Upgrade Example

```
find(Str) ->
    case re:run(Str, ?RE, [{capture,first}]) of
        {match, [{Start, Len}]} ->
            strings:substr(Str, Start+1, Len);
        nomatch ->
            ""
    end.
```

Return value changes are propagated to the place of usage

Generic Interface Migration

- Same functionality
  - Arguments and return values have the same information
- Incompatible interfaces
  - Data is restructured or slightly modified
- Many simple library changes could be supported by an automated generic migration tool

```
case dict:find(K,S) of
  {ok, Val} -> Val;
  error -> throw(miss)
end
```

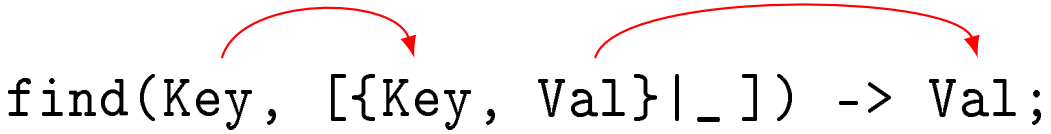
```
case gb_trees:lookup(K,S) of
  {value, Val} -> Val;
  none -> throw(miss)
end
```

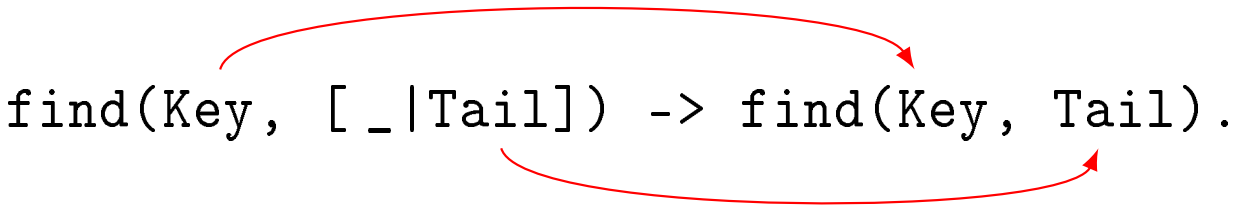
Data Flow Analysis

```
find(Key, [{Key, Val}|_]) -> Val;

find(Key, [_|Tail]) -> find(Key, Tail).

f() -> find(a, [{a,1}]).
```

Data Flow Analysis

Direct edges: variables

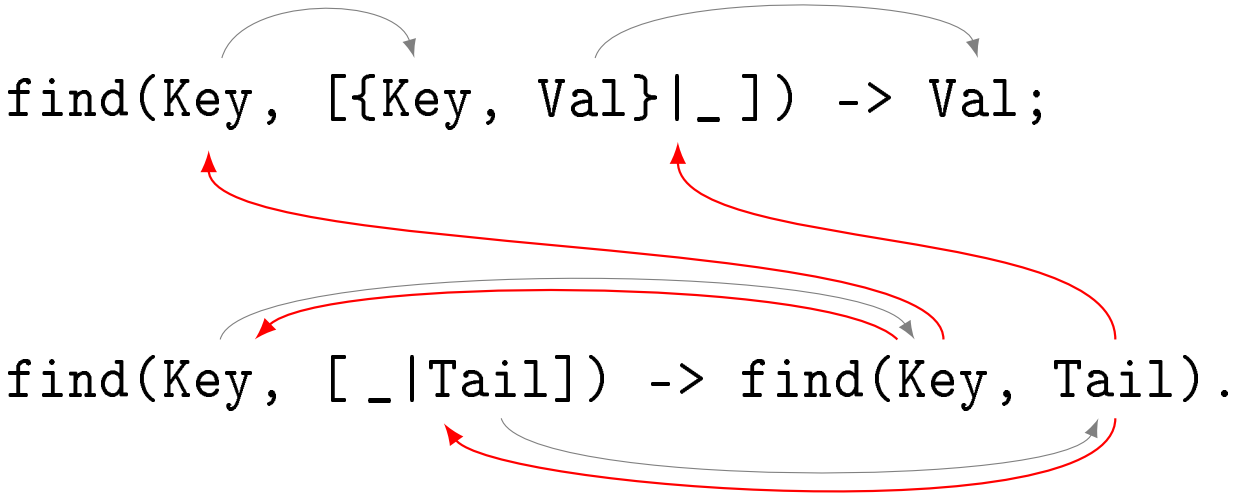





```
f() -> find(a, [{a,1}]).
```

Data Flow Analysis

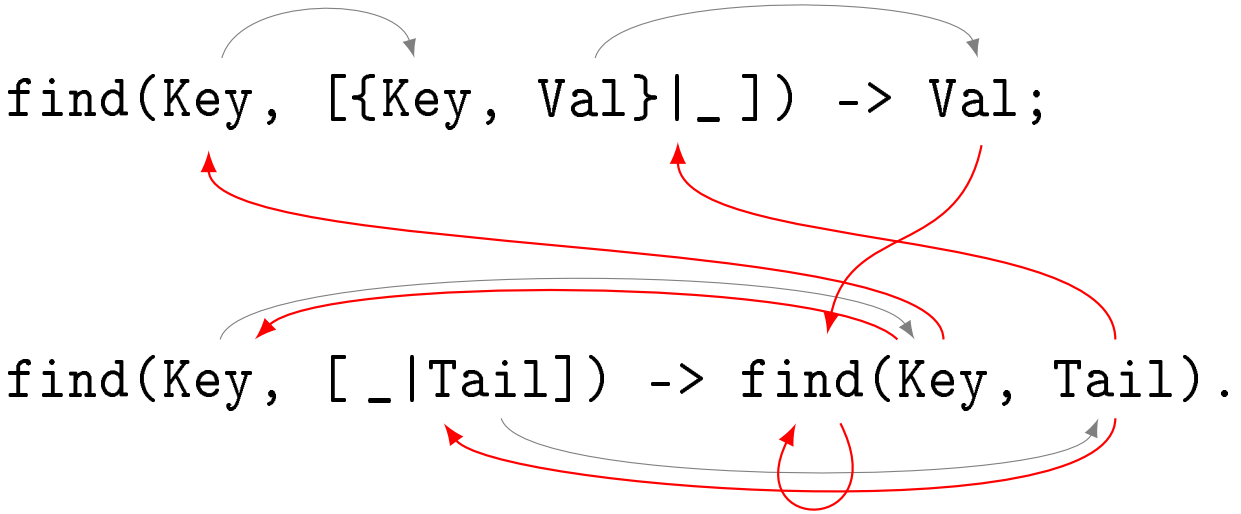
Direct edges: function calls




```
f() -> find(a, [{a,1}]).
```

Data Flow Analysis

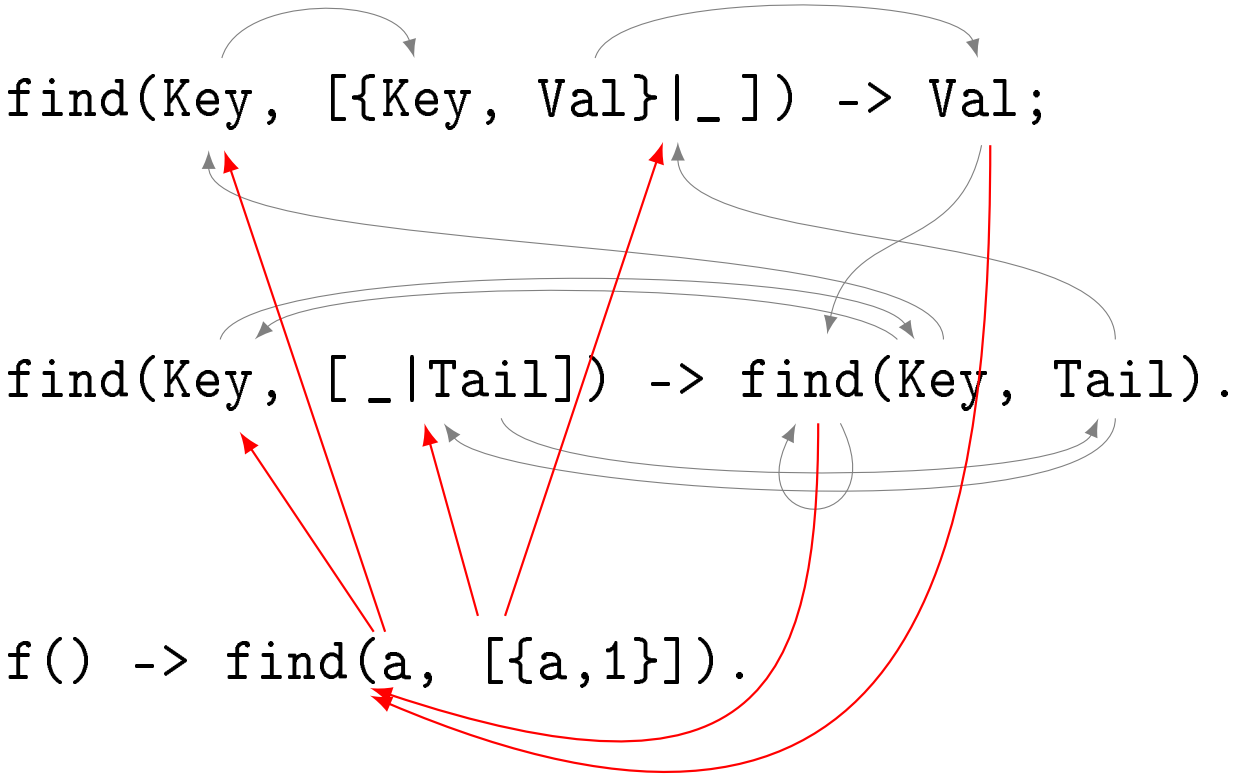
Direct edges: function calls




```
f() -> find(a, [{a,1}]).
```

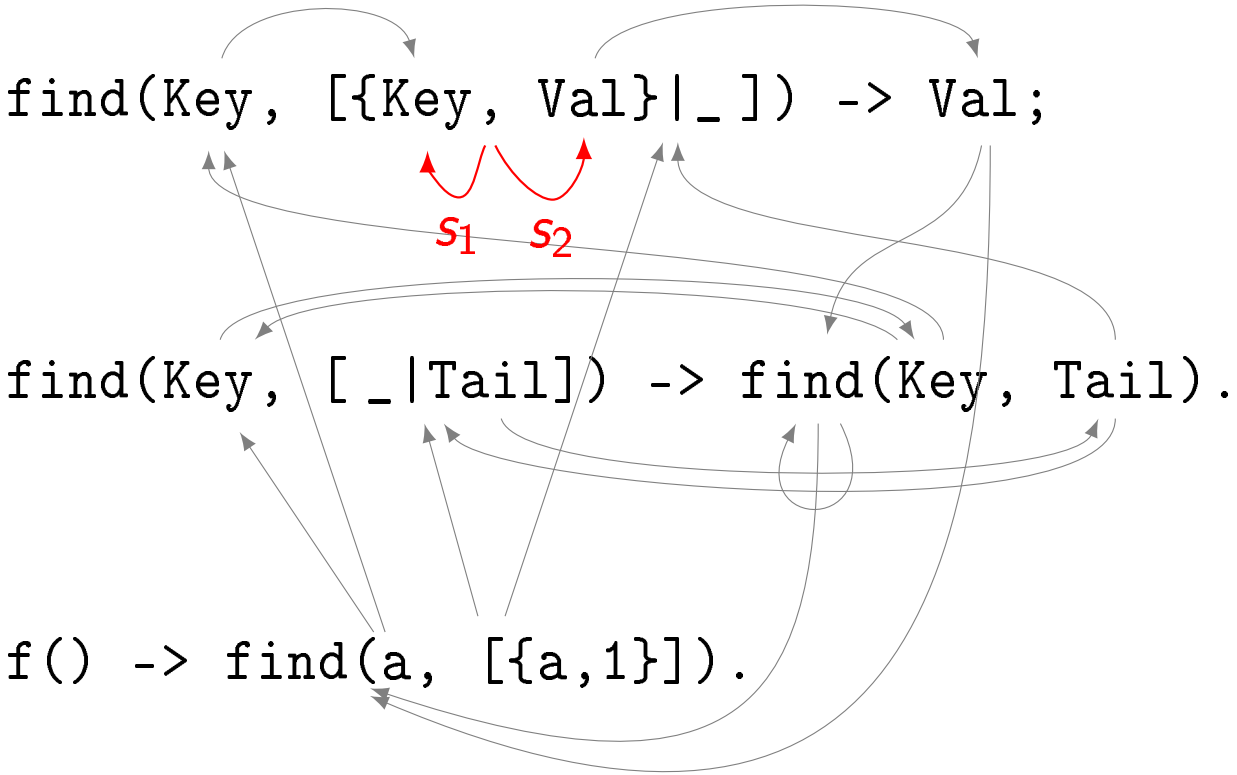
Data Flow Analysis

Direct edges: function calls



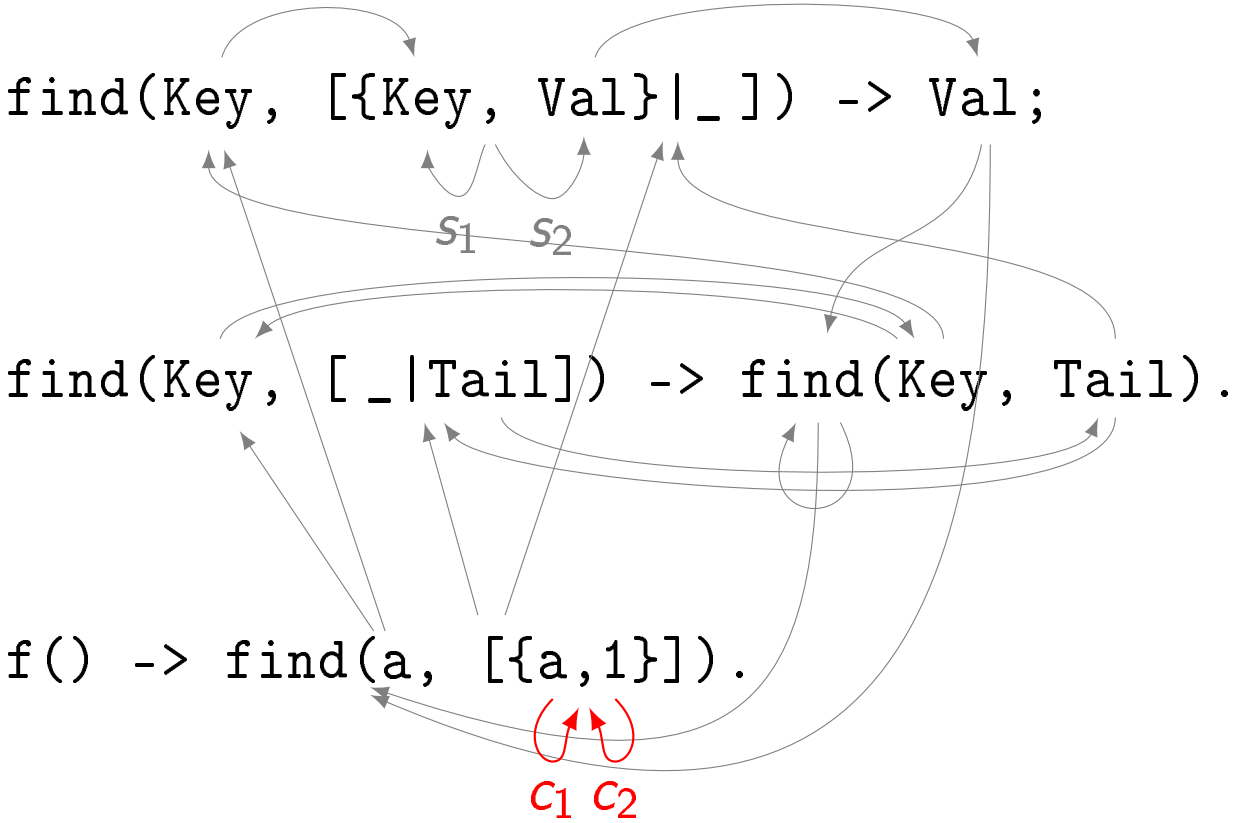
Data Flow Analysis

Direct edges: tuple selectors



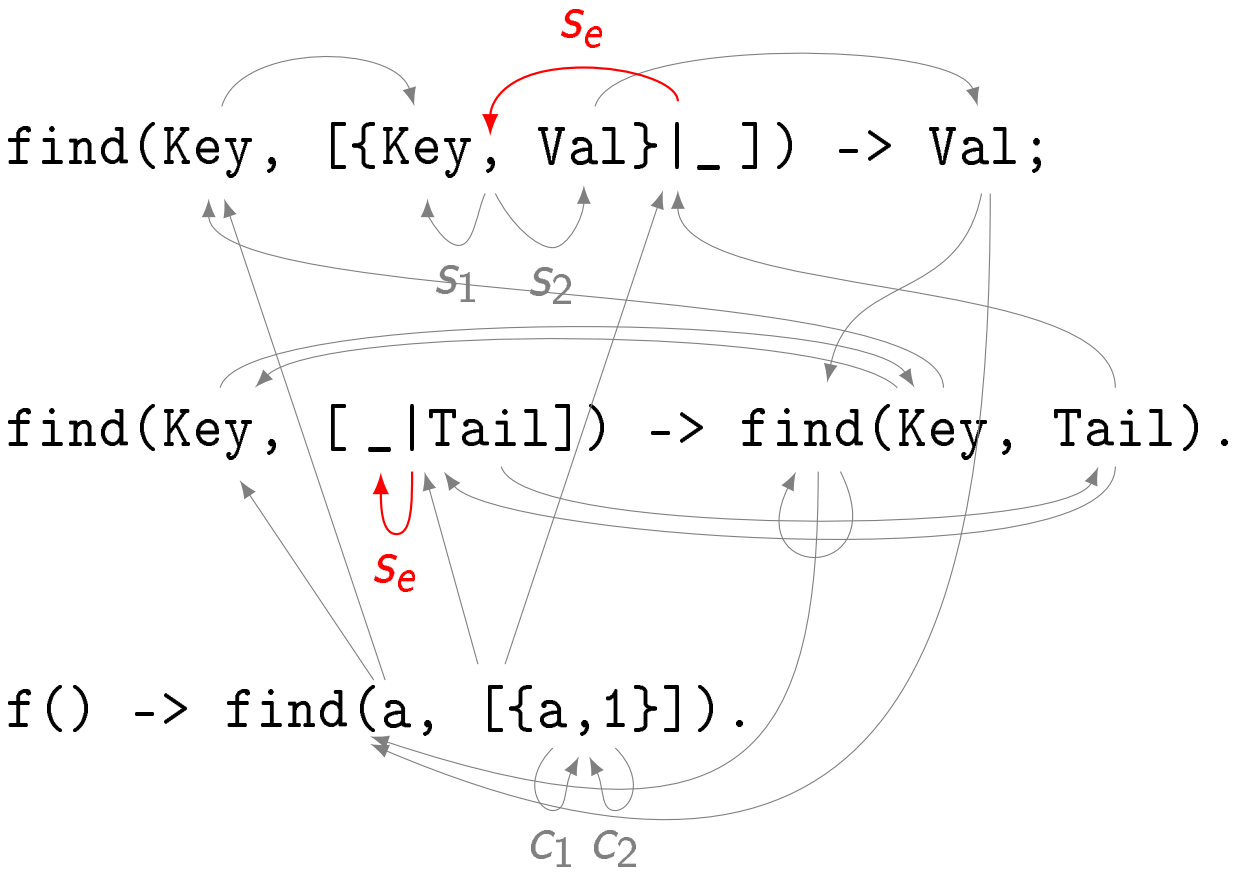
Data Flow Analysis

Direct edges: tuple constructors



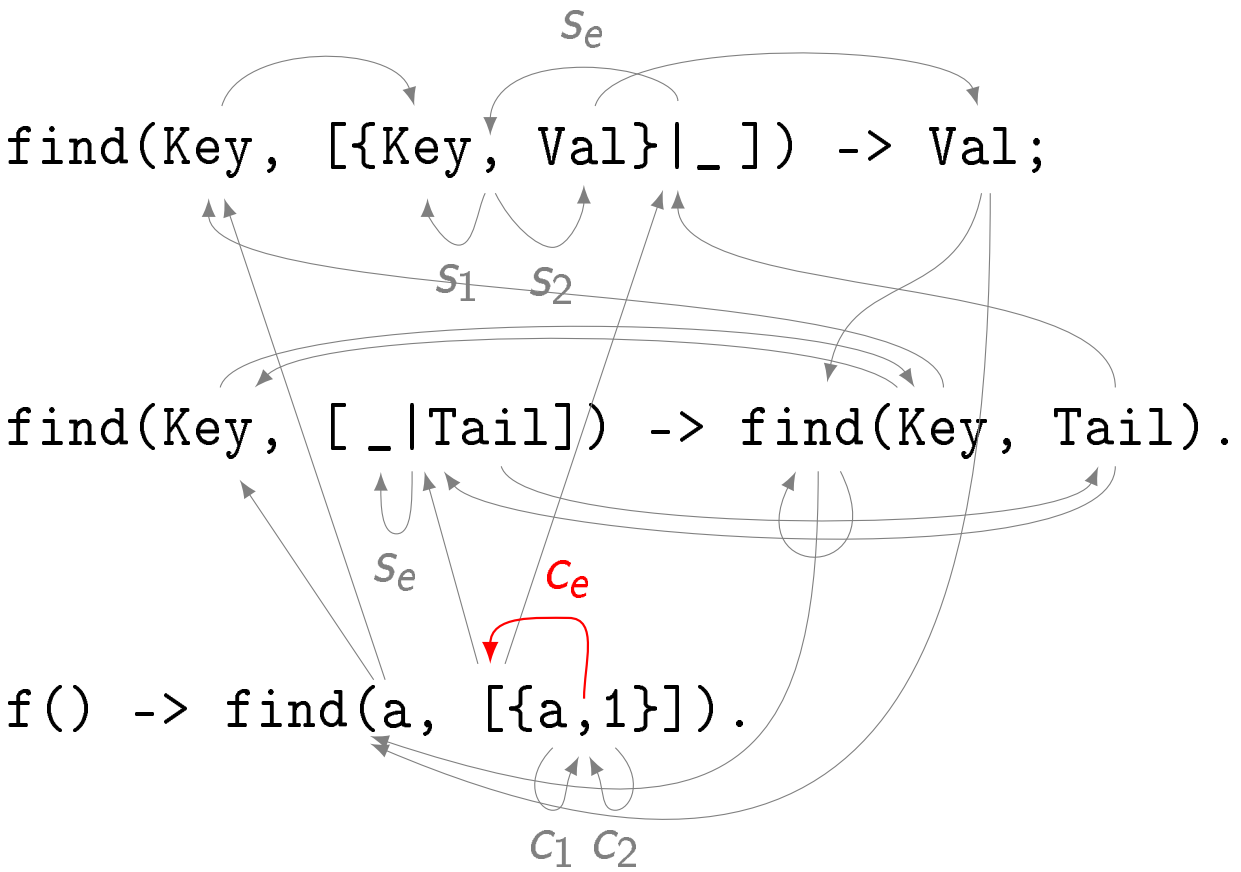
Data Flow Analysis

Direct edges: list element selectors



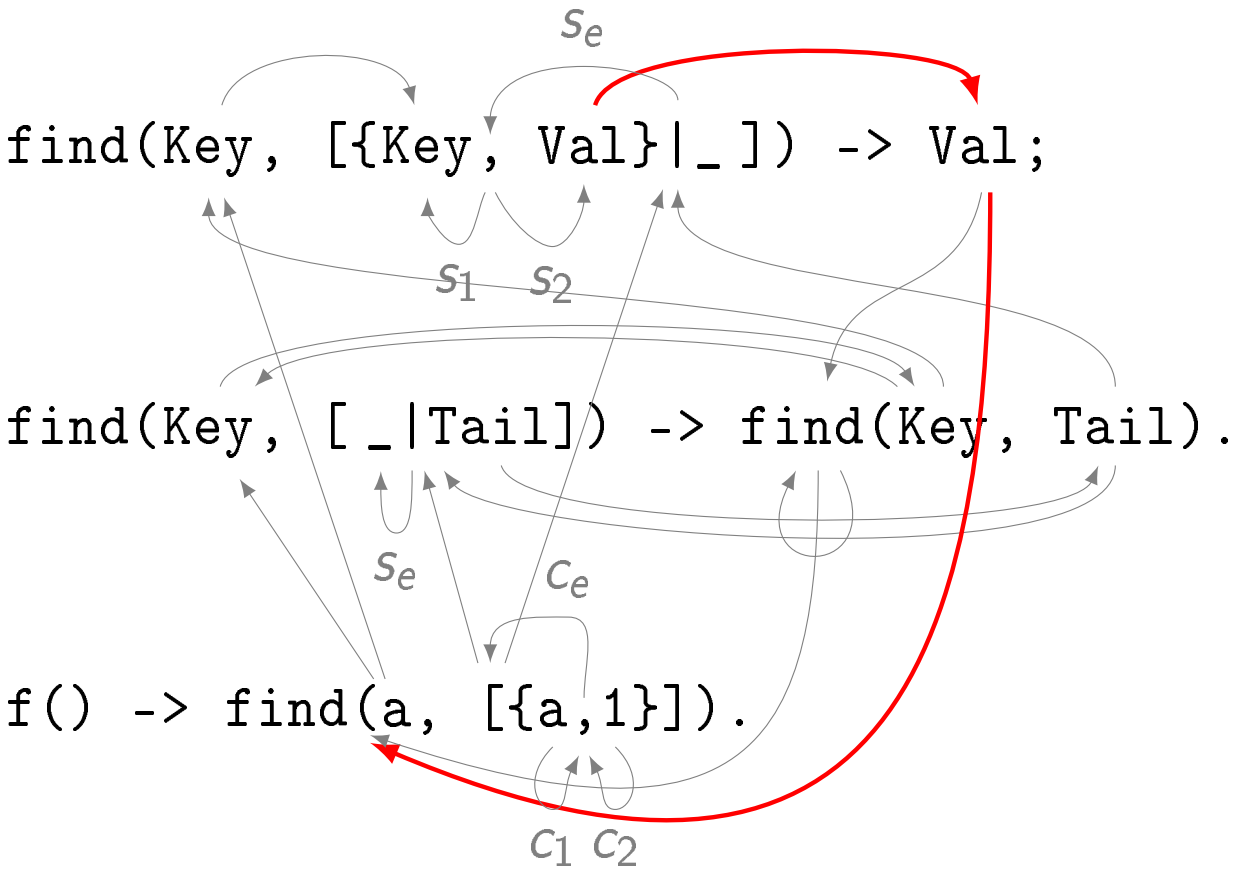
Data Flow Analysis

Direct edges: list constructors



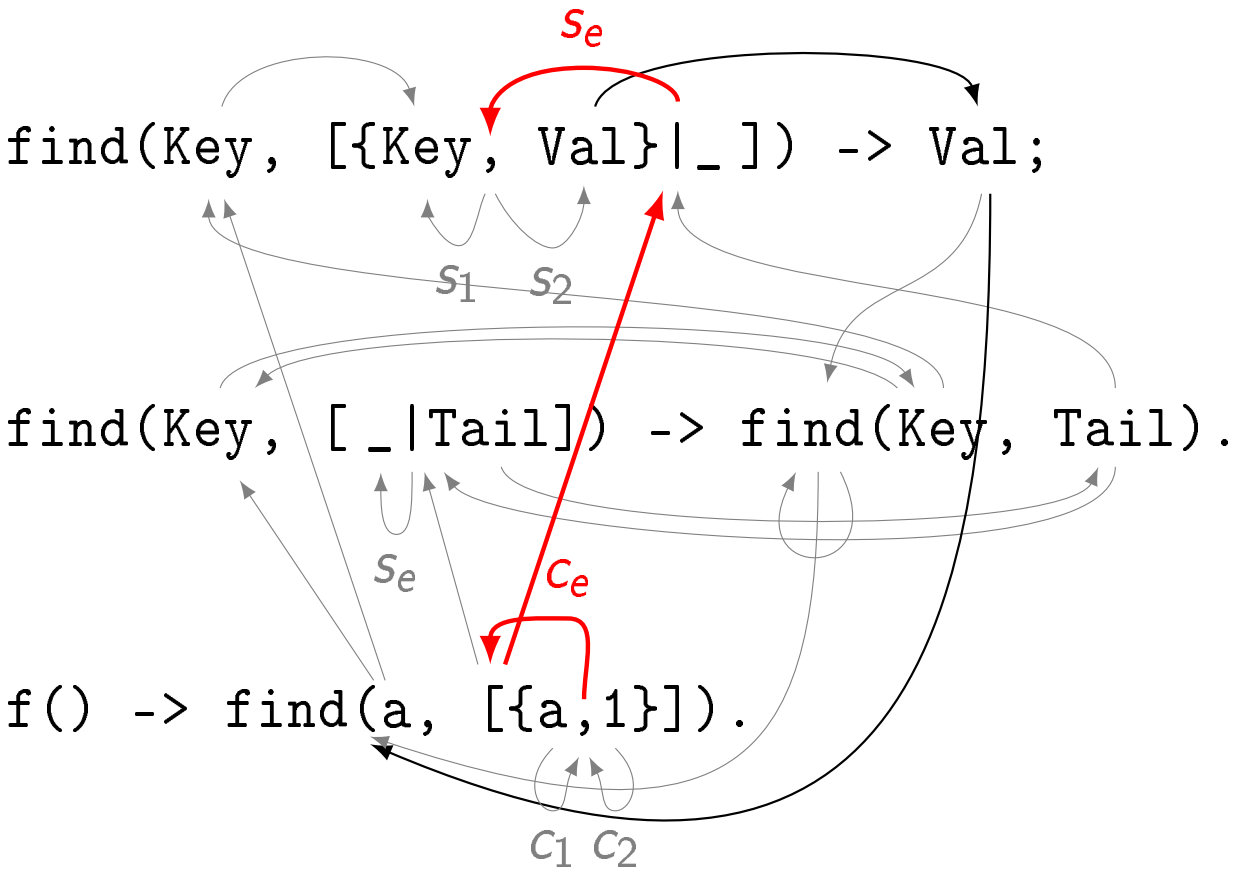
Data Flow Analysis

Reaching: transitivity



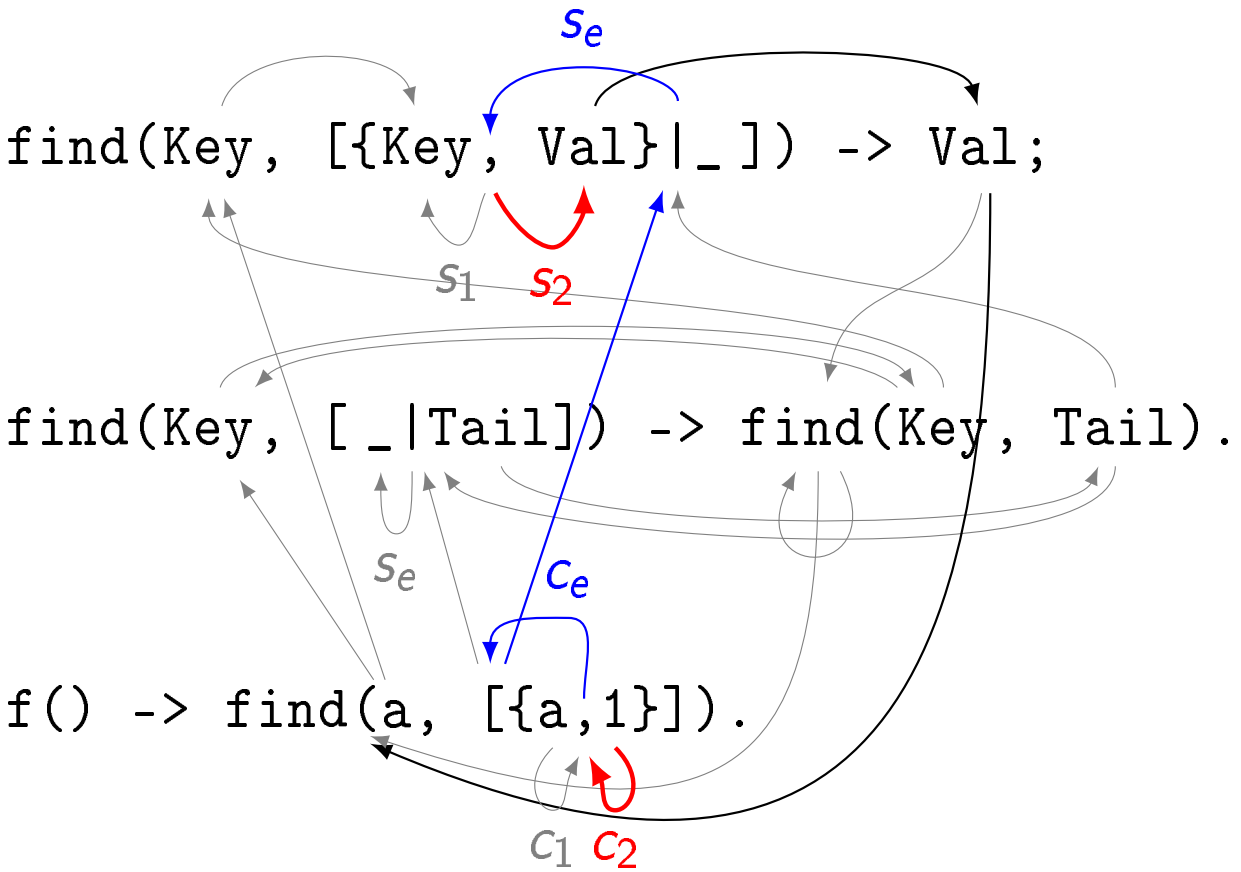
Data Flow Analysis

Reaching: list construction and selection



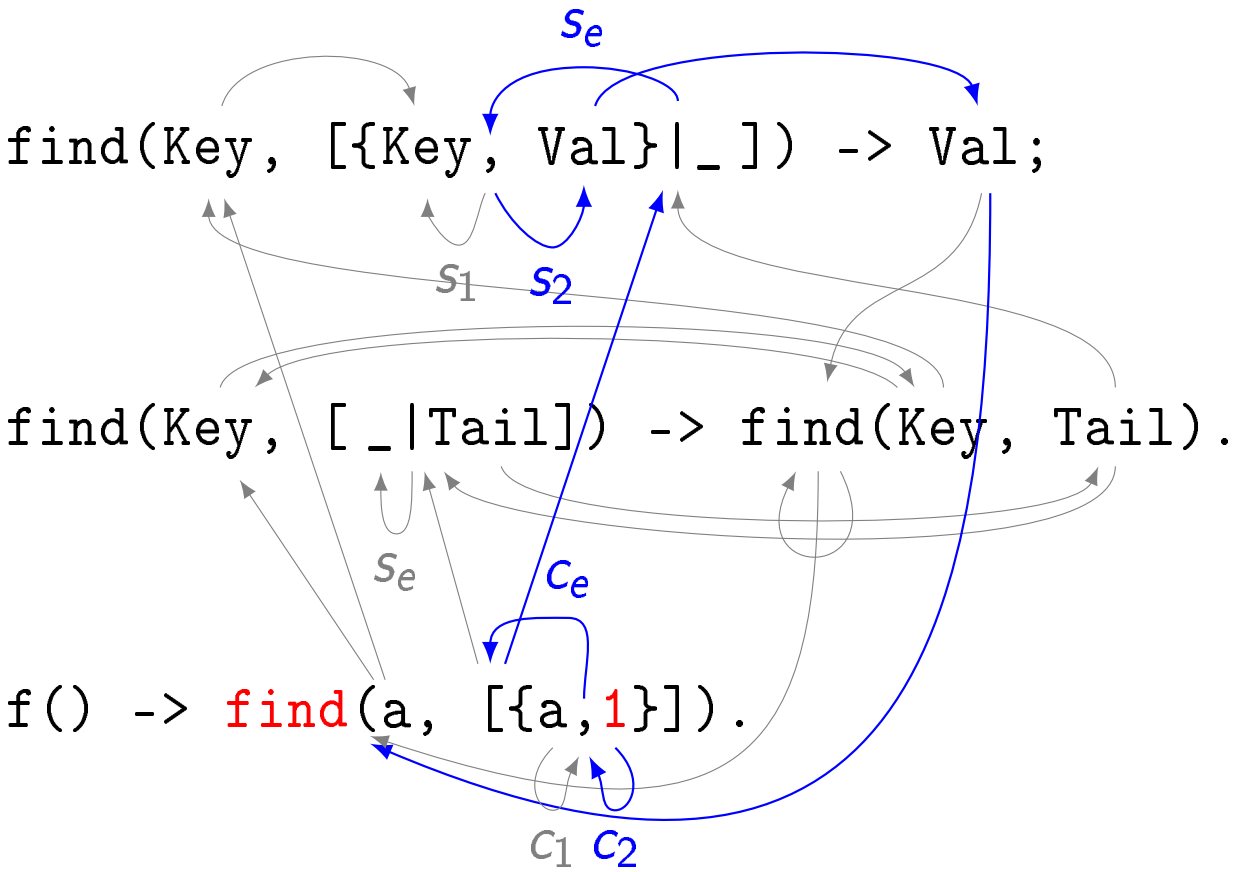
Data Flow Analysis

Reaching: tuple construction and selection



Data Flow Analysis

Propagation: finding the end of data flow paths



Describing Changes

- Only simple changes: the same data is available...
- ...in a different structure...
  - Data patterns describe the new and the old structure:

    `{match, St, Len}` $\mapsto$ `{match, [{decr(St), Len}]}`
    `nomatch` $\mapsto$ `nomatch`
- ...or in a slightly modified form
  - Compensations are provided by simple expressions:
    `decr(Old` $\mapsto$ `New): Old-1`
    `decr(New` $\mapsto$ `Old): New+1`
- These are sufficient to upgrade the `regexp` module calls

Prototype Implementation

- RefactorErl infrastructure is used
  - Semantic analysis
  - Syntax tree-based transformations
- Linear time and space complexity
  - Direct graph: about same size as the syntax tree
  - Reaching computation: breadth-first walk limited to the affected graph components
  - Usually a module is transformed in one step

Summary

Data structure refactoring for module interface migration

- Simple but powerful change descriptions
- Change propagation by data flow analysis