Semantics of an Intermediate Language for Program Transformation

Sigurd Schneider
Advisors: Sebastian Hack, Gert Smolka

Student Session at POPL 2013

Saarland University
Graduate School of Computer Science

January 25, 2013

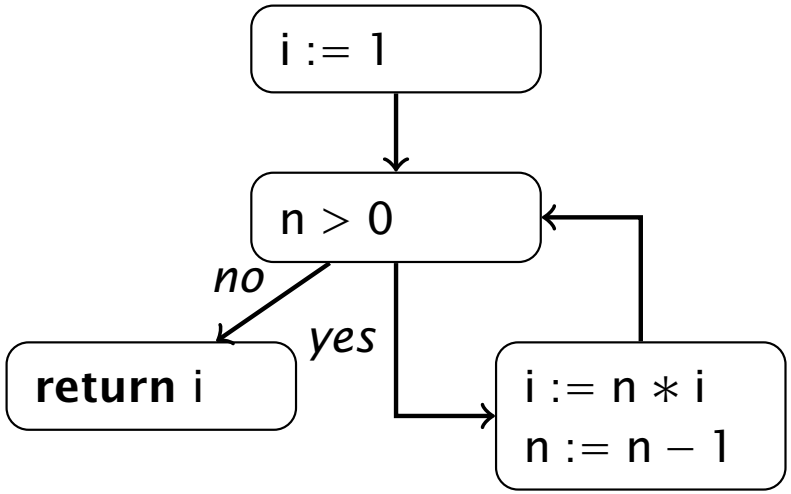
Motivation

1. Compiler technology is critical for
   - Performance
   - Reliability
   - Availability
2. State of the art
   - Unverified: LLVM, HotSpot, GCC, MSVC, ICC
   - Verified: CompCert
   - As of today, compiler correctness is hard
3. How to verify advanced optimizations in an extensible way?
   - Optimizations from research compiler
   - Correctness proofs in type theory (Coq)

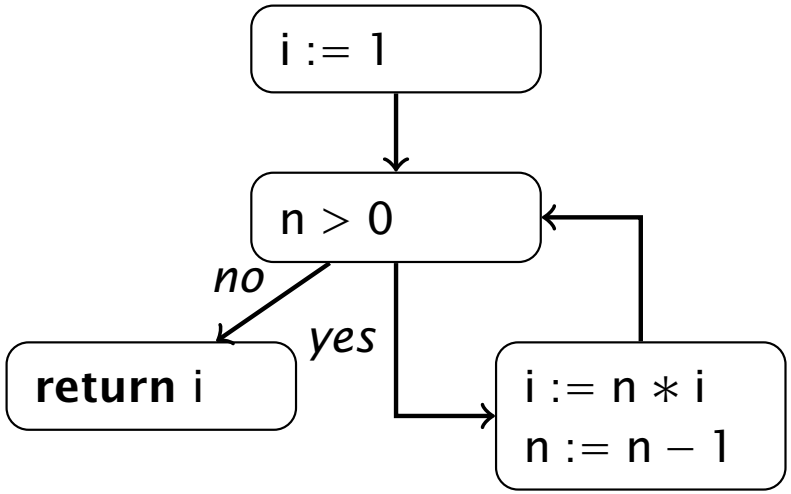
Approach

1. Functional intermediate language
   - Well-understood semantics
   - Compiler more effective and easier to implement
2. State of the art in functional IRs
   - VSDG: Higher Order [Johnson and Mycroft 2003]
   - PEG: “Stream Semantics” [Tate et al. 2009]
   - Firm-IR: “Click’s Sea of Nodes” [Braun, Buchwald, and Zwinkau 2011]
   - CompCert does not use functional intermediate language
3. Research questions
   - Are proofs simpler and more modular for functional IL?
   - Can we exploit results from the literature?
   - **Translation: Imperative $\Leftrightarrow$ Functional**

IL/I: Formalizing Control Flow Graphs



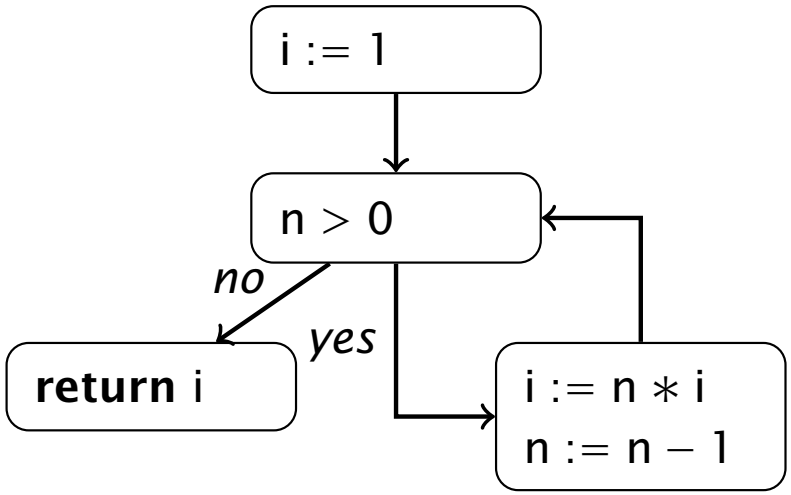
IL/I: Formalizing Control Flow Graphs




```
1  i := 1;
2  letrec f () =
3    if n > 0 then
4      i := n * i;
5      n := n - 1;
6      f ()
7    else
8      i
9  in
10   f ()
```

- CFG can be **represented** by IL/I
  1. (Irreducible) control flow as (mutual) recursion
  2. Data flow through imperative variables
- Parameter passing is parallel assignment

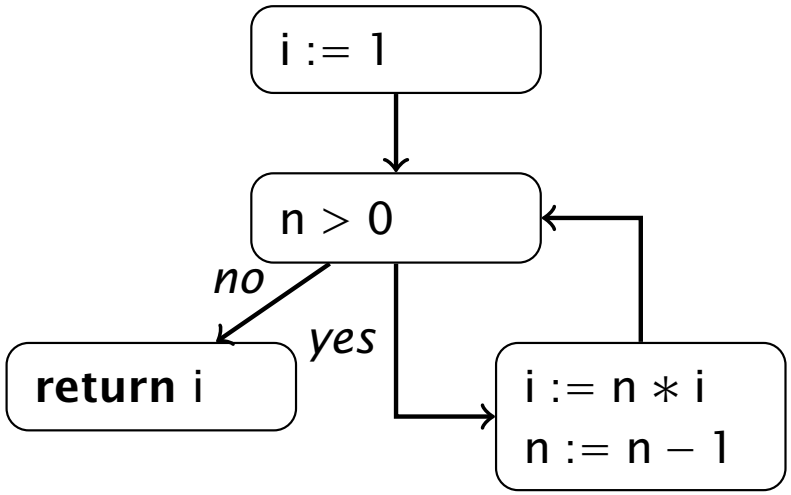
IL/F: Functional language representing CFGs




```
1  let i = 1 in
2  letrec f (n, i) =
3    if n > 0 then
4      let i = n * i in
5      let n = n - 1 in
6      f (n, i)
7    else
8      i
9  in
10   f (n, i)
```

- CFG can be **translated** to functional language IL/F
  1. (Irreducible) control flow as (mutual) recursion
  2. Data flow through binding and explicit arguments
- Static Single Assignment (SSA) [Appel 1998; Kelsey 1995] [Barthe, Demange, and Pichardie 2012; Zhao and Zdancewic 2012]

IL/F: Functional language representing CFGs

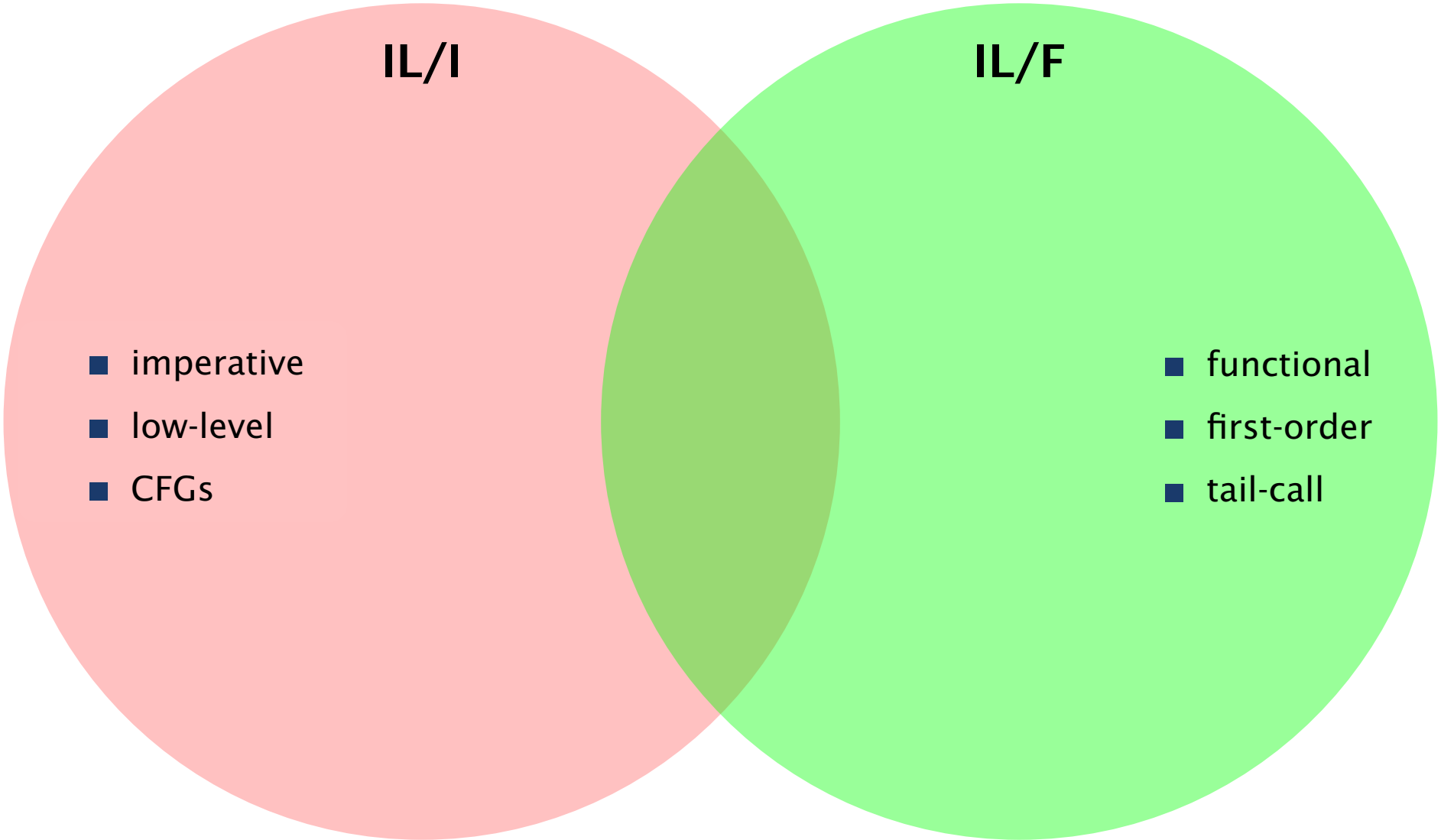



```
1  let i = 1 in
2  letrec f (n,i) =
3    if n > 0 then
4      let j = n * i in
5      let m = n - 1 in
6      f (m,j)
7    else
8      i
9  in
10   f (n,i)
```

- CFG can be **translated** to functional language IL/F
  1. (Irreducible) control flow as (mutual) recursion
  2. Data flow through binding and explicit arguments
- Static Single Assignment (SSA) [Appel 1998; Kelsey 1995] [Barthe, Demange, and Pichardie 2012; Zhao and Zdancewic 2012]

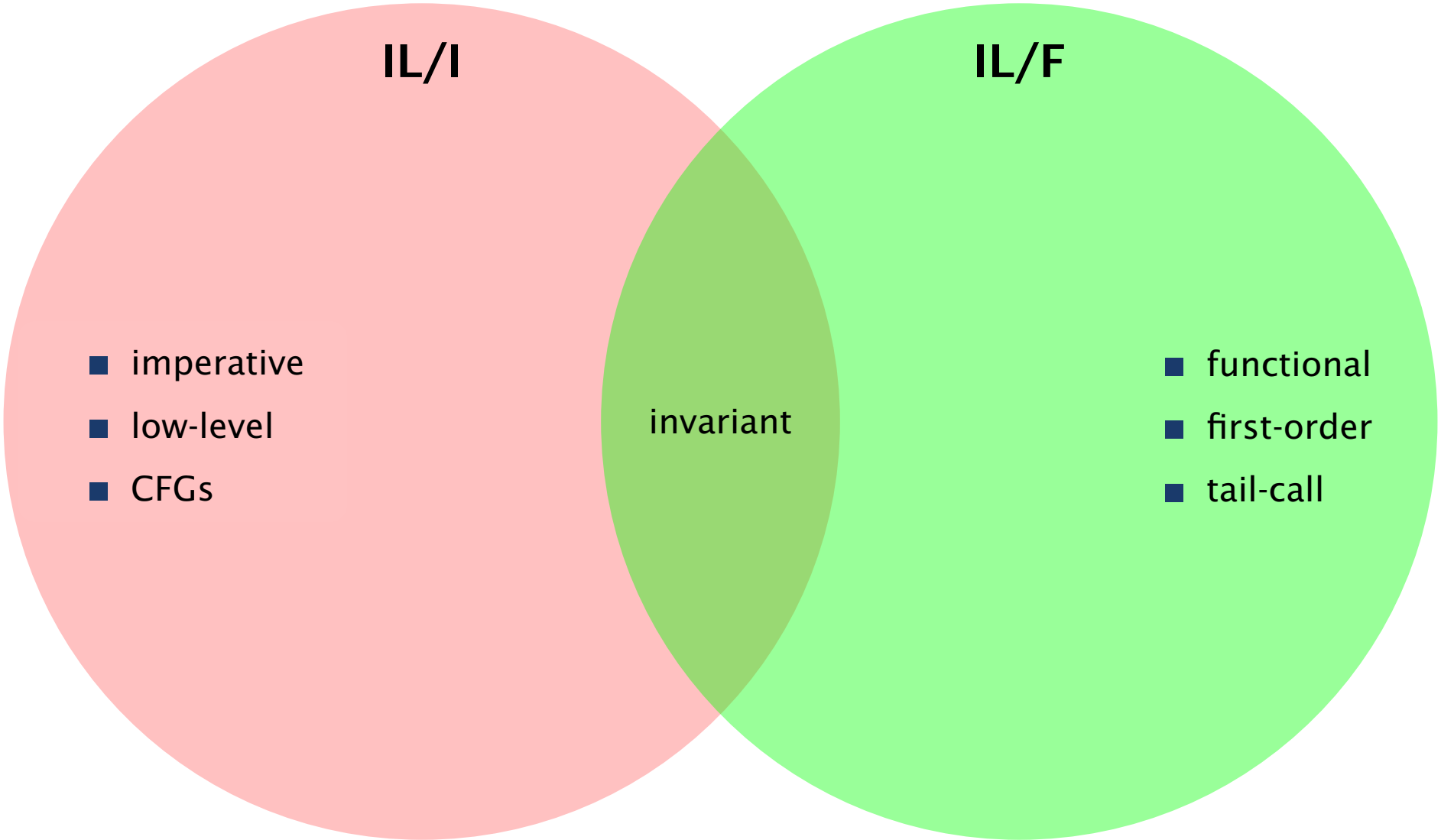
IL: One syntax, two semantic interpretations

SAARLAND UNIVERSITY
COMPUTER SCIENCE



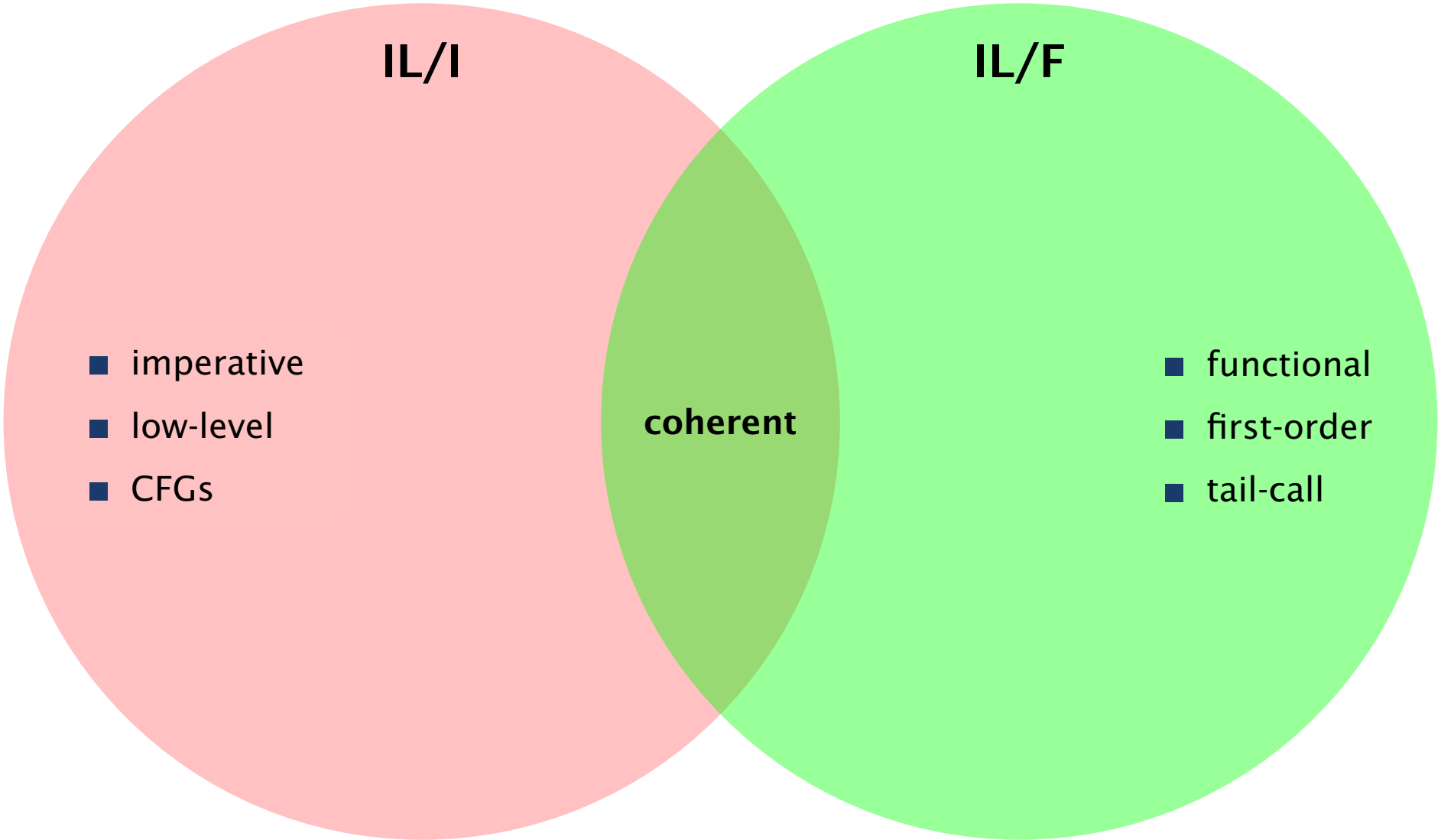
IL: One syntax, two semantic interpretations

SAARLAND UNIVERSITY
COMPUTER SCIENCE

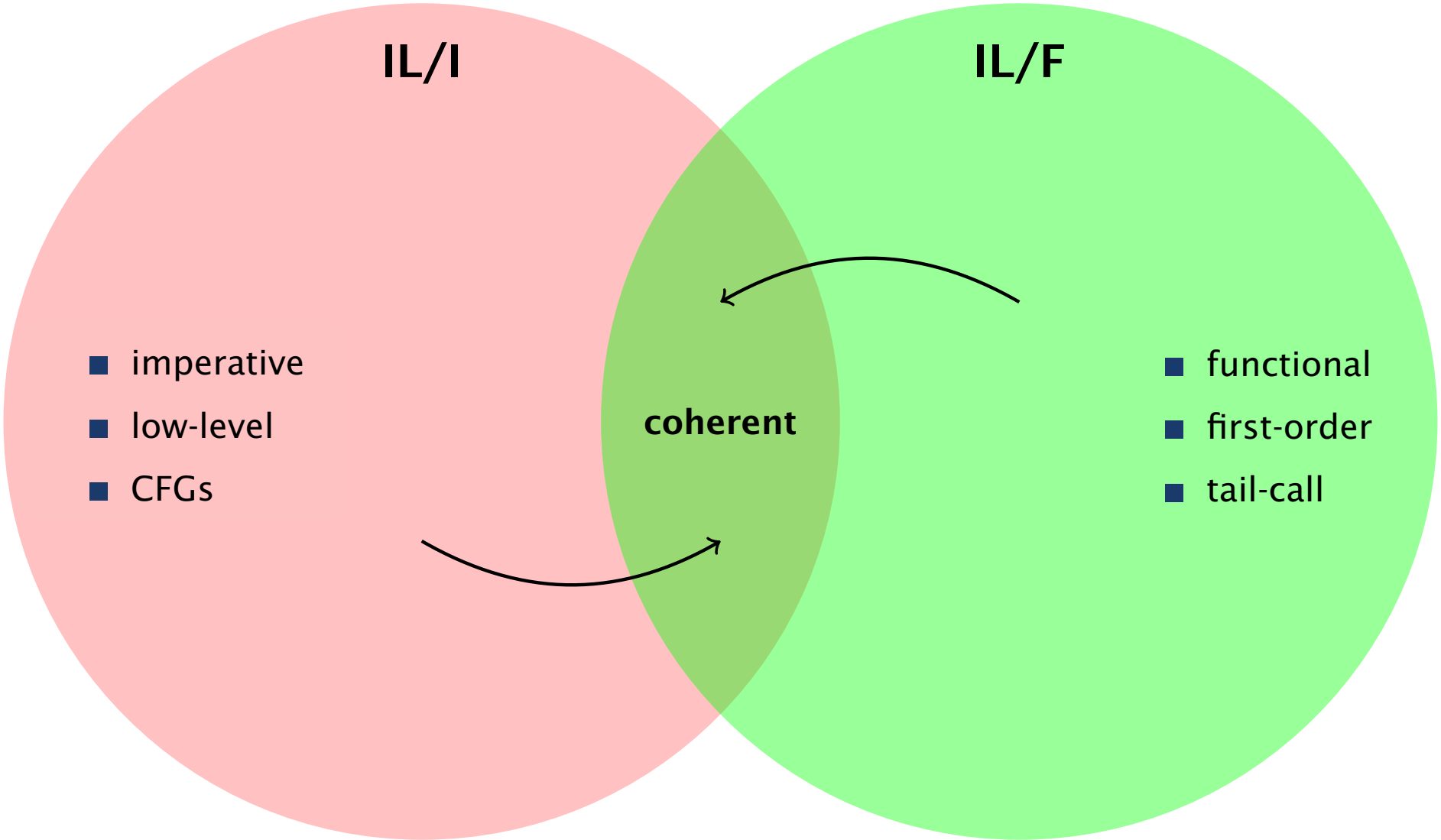


IL: One syntax, two semantic interpretations

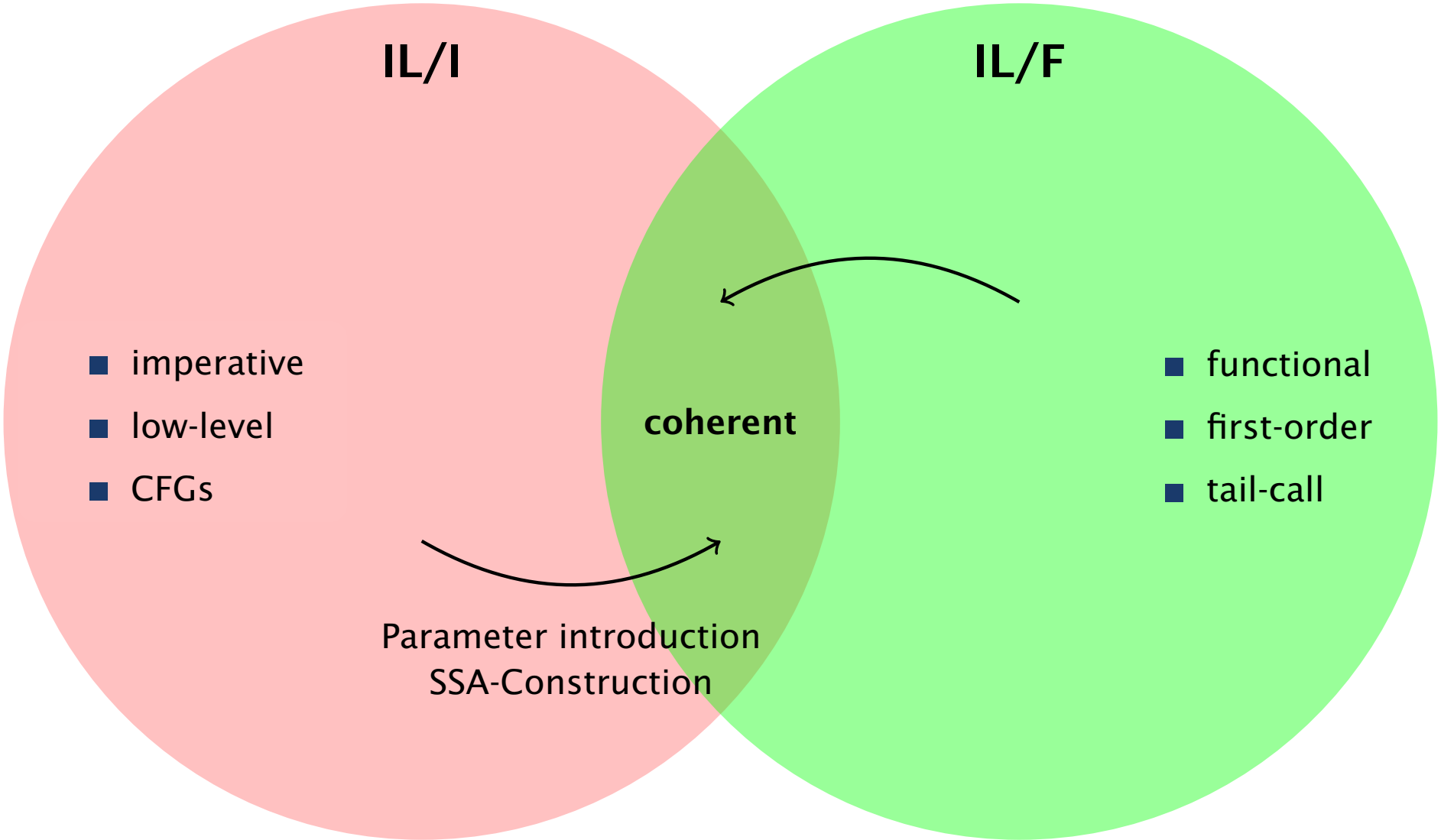
SAARLAND UNIVERSITY
COMPUTER SCIENCE



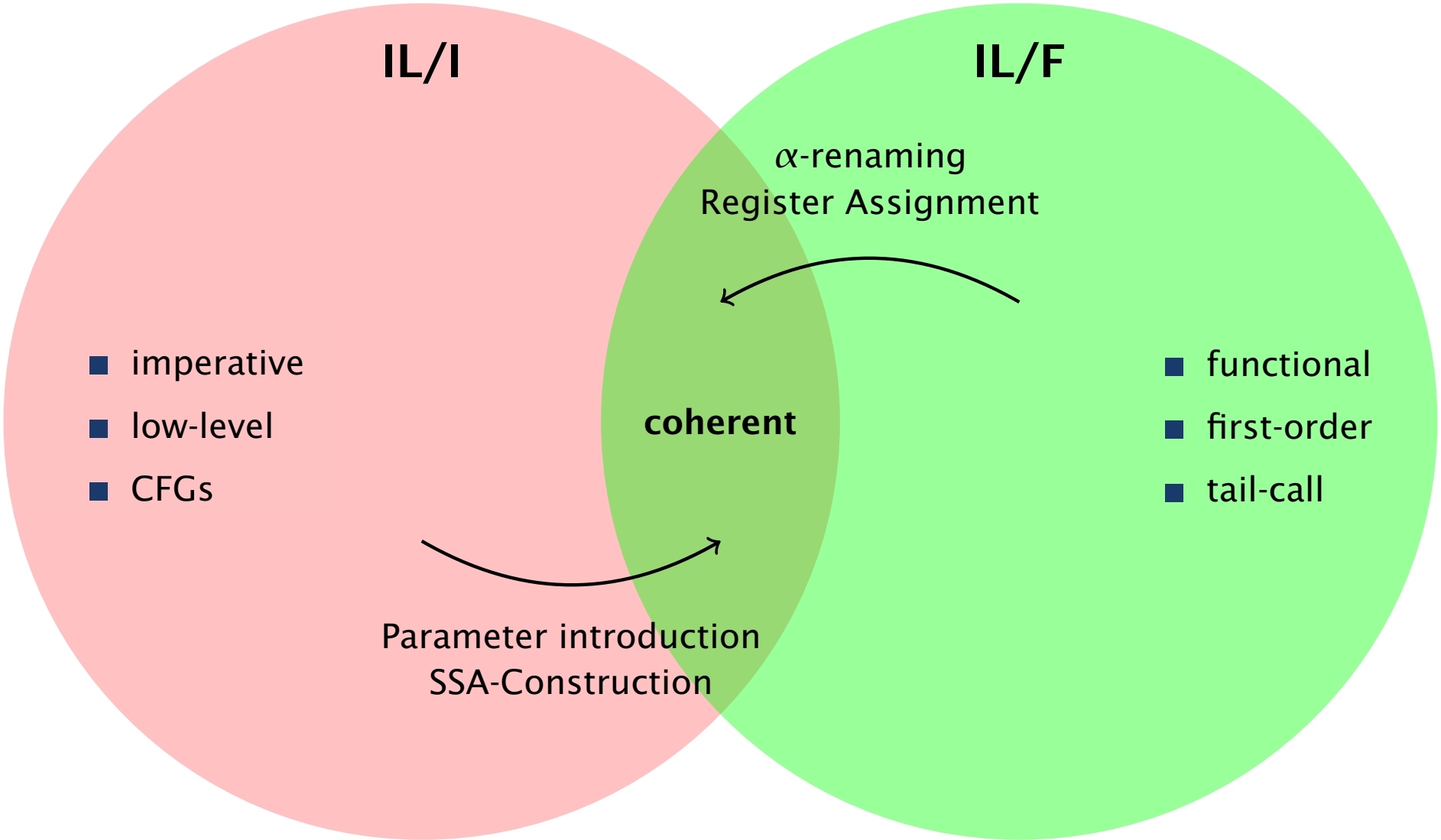
IL: One syntax, two semantic interpretations



IL: One syntax, two semantic interpretations



IL: One syntax, two semantic interpretations



Register Allocation: Example

- Register assignment
  - α-renaming to coherent program
- Parameter elimination
  - parallel assignment [Rideau, Serpette, and Leroy 2008]

4 registers in use: i,j,m,n

```
1  let i = 1 in
2  letrec f (n,i) =
3    if n > 0 then
4      let j = n * i in
5      let m = n - 1 in
6      f (m,j)
7    else
8      i
9  in
10   f (n,i)
```

Register Allocation: Example

- **Register assignment**
  - α-renaming to coherent program
- Parameter elimination
  - parallel assignment [Rideau, Serpette, and Leroy 2008]

4 registers in use: i,j,m,n

```
1  let i = 1 in
2  letrec f (n,i) =
3    if n > 0 then
4      let j = n * i in
5      let m = n - 1 in
6      f (m,j)
7    else
8      i
9  in
10   f (n,i)
```

2 registers in use: i,n

```
1  let i = 1 in
2  letrec f (n,i) =
3    if n > 0 then
4      let i = n * i in
5      let n = n - 1 in
6      f (n,i)
7    else
8      i
9  in
10   f (n,i)
```

Register Allocation: Example

- **Register assignment**
  - α-renaming to coherent program
- Parameter elimination
  - parallel assignment [Rideau, Serpette, and Leroy 2008]

4 registers in use: i,j,m,n

```
1  let i = 1 in
2  letrec f (n,i) =
3    if n > 0 then
4      let j = n * i in
5      let m = n - 1 in
6      f (m,j)
7    else
8      i
9  in
10   f (n,i)
```

2 registers in use: i,n

```
1  i := 1;
2  letrec f (n,i) =
3    if n > 0 then
4      i := n * i;
5      n := n - 1;
6      f (n,i)
7    else
8      i
9  in
10   f (n,i)
```

Register Allocation: Example

- Register assignment
  - α-renaming to coherent program
- **Parameter elimination**
  - parallel assignment [Rideau, Serpette, and Leroy 2008]

4 registers in use: i,j,m,n

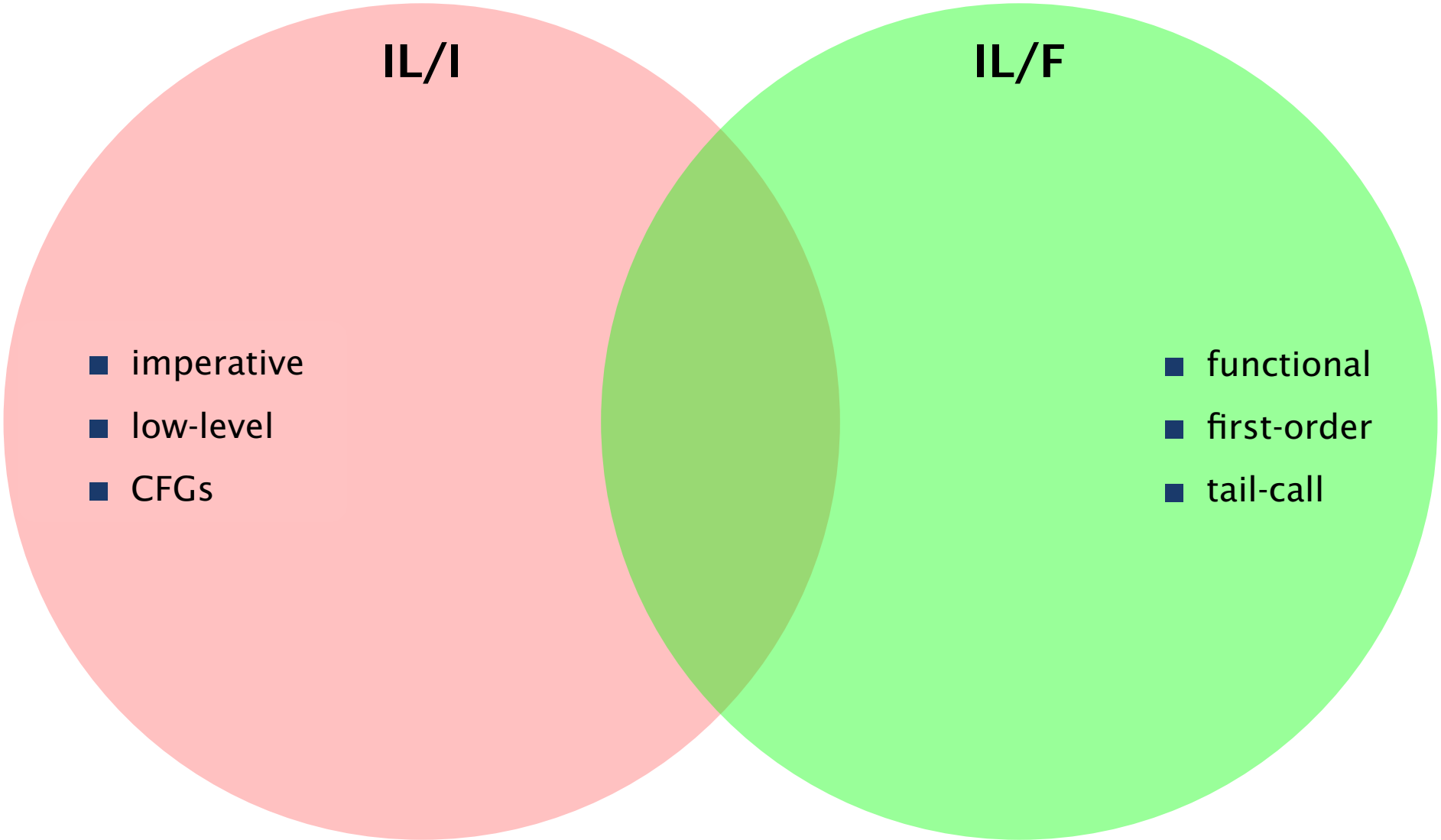
```
1  let i = 1 in
2  letrec f (n,i) =
3    if n > 0 then
4      let j = n * i in
5      let m = n - 1 in
6      f (m,j)
7    else
8      i
9  in
10   f (n,i)
```

2 registers in use: i,n

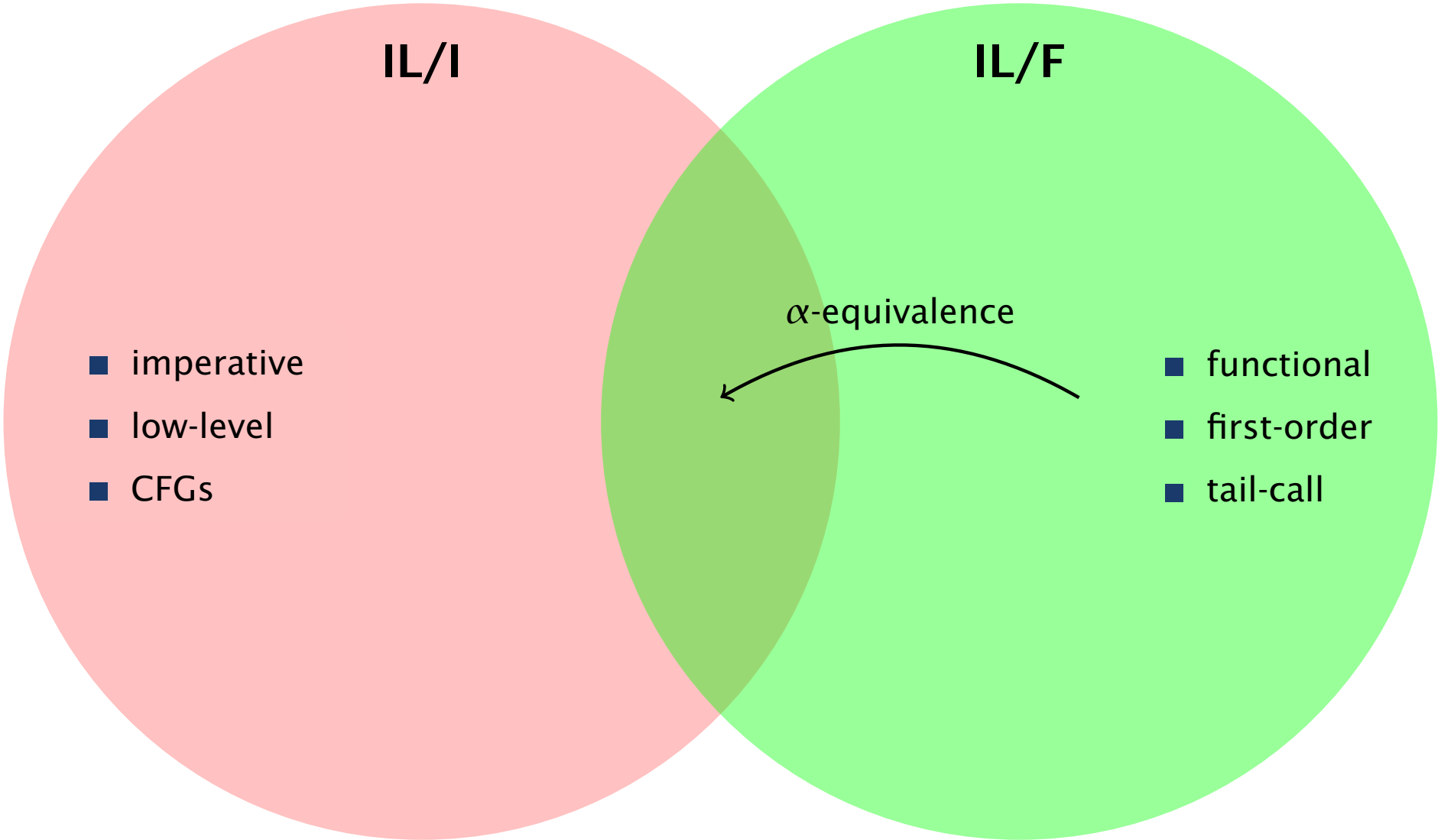
```
1  i := 1;
2  letrec f () =
3    if n > 0 then
4      i := n * i;
5      n := n - 1;
6      f ()
7    else
8      i
9  in
10   f ()
```

Register Allocation: Correctness Argument

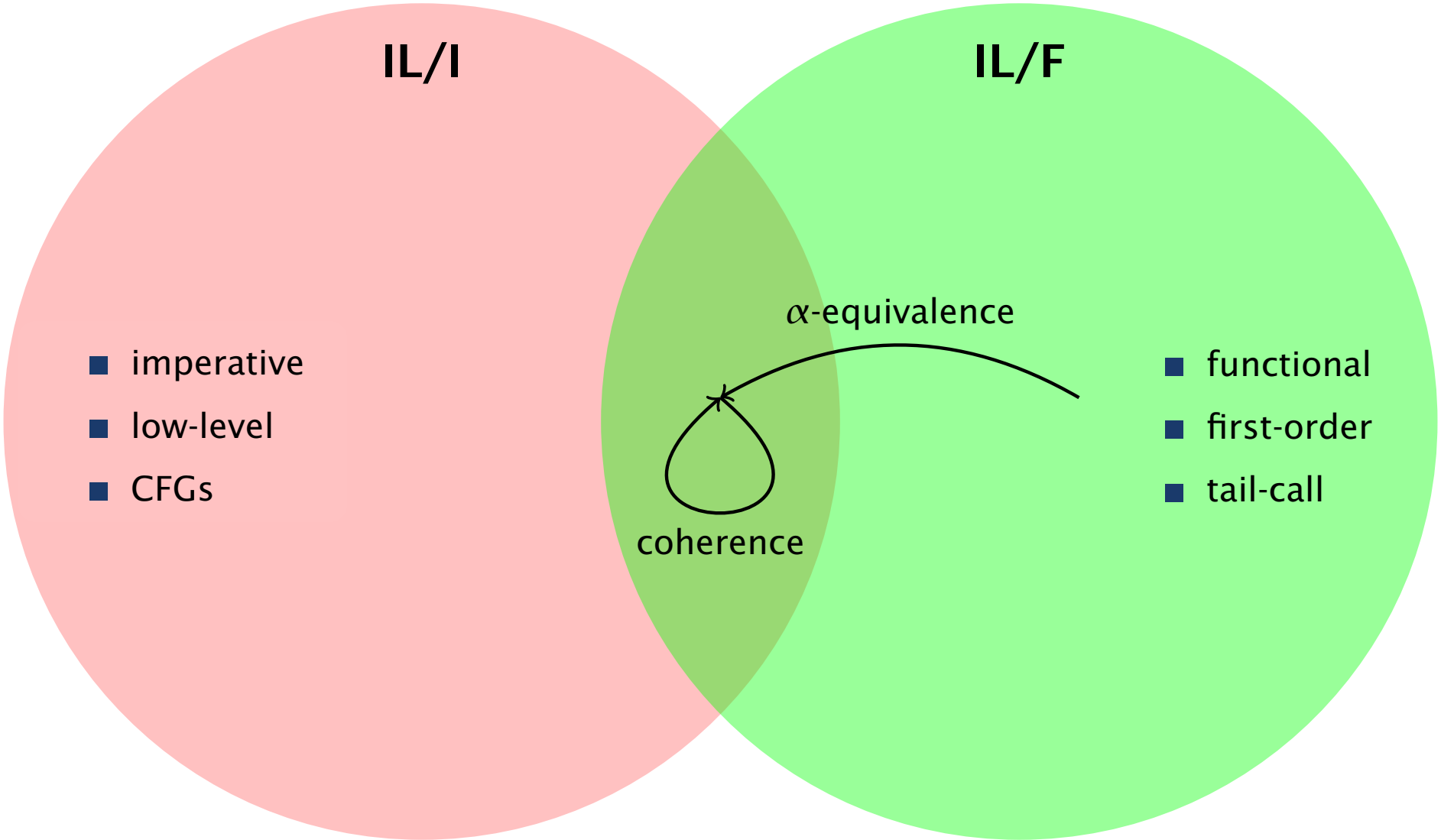
SAARLAND UNIVERSITY
COMPUTER SCIENCE



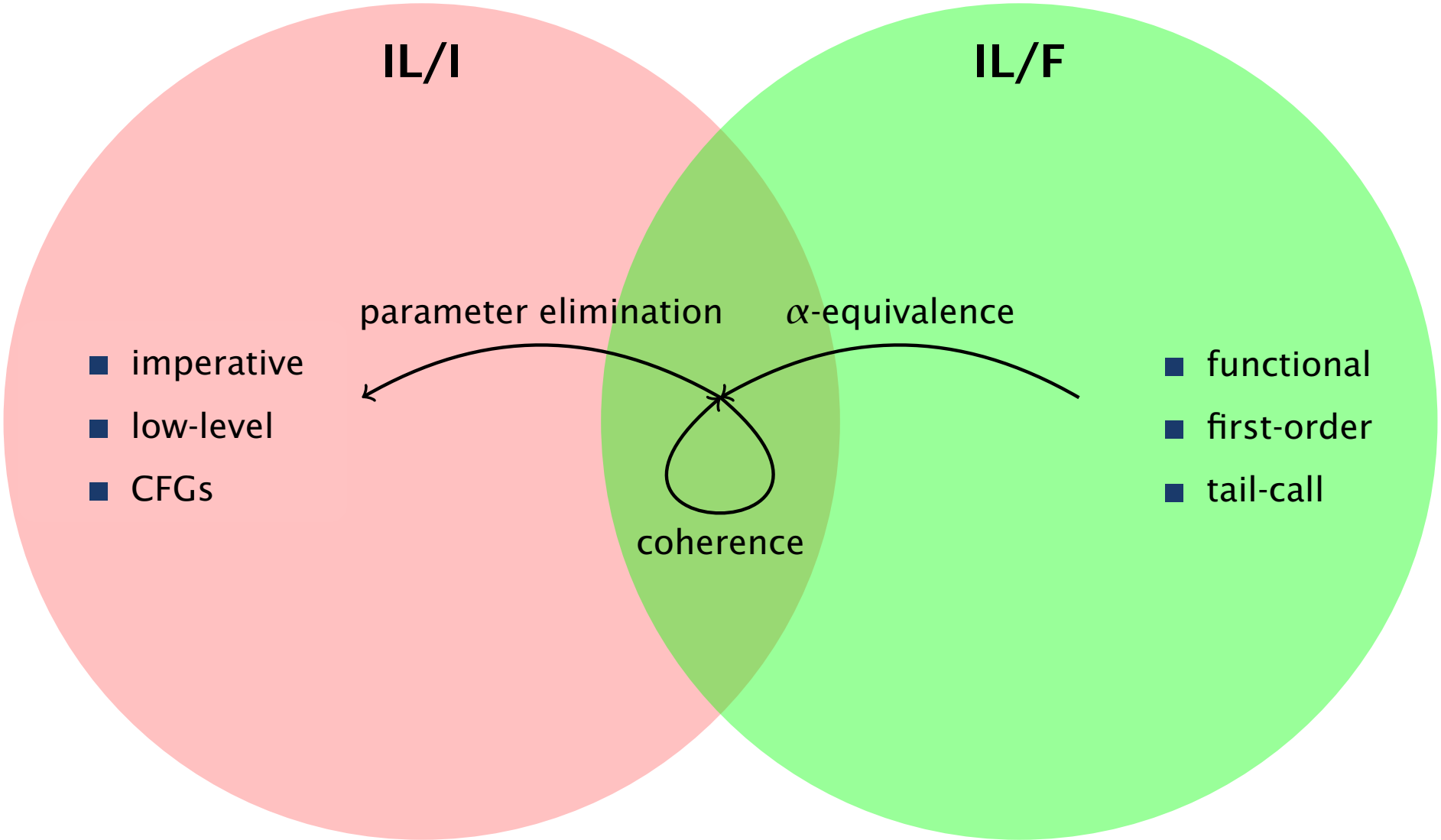
Register Allocation: Correctness Argument



Register Allocation: Correctness Argument



Register Allocation: Correctness Argument



Conclusion

1. IL provides bridge between imperative and functional world
2. Correctness arguments on functional side
   - register allocation is α-renaming to coherent program
3. Formalization in Coq

4. Future work
   - Application of aggressive optimizations on the functional side [Chakravarty, Keller, and Zadarnowski 2003]
   - Contextual program equivalence in IL

Thank you for listening! Questions? I

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Thanks for the great questions and discussion after the Talk, pointing out most closely related work by Beringer, MacKenzie, and Stark (2003), who also present a language with a dual semantic interpretation.

Appel, Andrew W. (1998). "SSA is Functional Programming". In: *SIGPLAN Notices* 4.

Barthe, Gilles, Delphine Demange, and David Pichardie (2012). "A Formally Verified SSA-Based Middle-End - Static Single Assignment Meets CompCert". In: *ESOP*. Ed. by Helmut Seidl. Lecture Notes in Computer Science.

Beringer, Lennart, Kenneth MacKenzie, and Ian Stark (2003). "Grail: a functional form for imperative mobile code". In: *Electr. Notes Theor. Comput. Sci.* 1.

Braun, Matthias, Sebastian Buchwald, and Andreas Zwinkau (2011). *Firm—A Graph-Based Intermediate Representation*. Tech. rep. 35. Karlsruhe Institute of Technology.

Chakravarty, Manuel M. T., Gabriele Keller, and Patryk Zadarnowski (2003). "A Functional Perspective on SSA Optimisation Algorithms". In: *Electr. Notes Theor. Comput. Sci.* 2.

Click, Cliff and Michael Paleczny (1995). "A Simple Graph-Based Intermediate Representation". In: *Intermediate Representations Workshop*. Ed. by Michael D. Ernst.

Johnson, Neil and Alan Mycroft (2003). "Combined Code Motion and Register Allocation Using the Value State Dependence Graph". In: *CC*. Ed. by Zhong Shao and Benjamin C. Pierce.

Kelsey, Richard A. (Mar. 1995). "A correspondence between continuation passing style and static single assignment form". In: *SIGPLAN Not.* (3). ISSN: 0362-1340.

Thank you for listening! Questions? II

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Leroy, Xavier (2009). “Formal verification of a realistic compiler”. In: *Communications of the ACM* 7.

Rideau, Laurence, Bernard Paul Serpette, and Xavier Leroy (2008). “Tilting at windmills with Coq: Formal verification of a compilation algorithm for parallel moves”. In: *Journal of Automated Reasoning* 4.

Tate, Ross et al. (2009). “Equality saturation: a new approach to optimization”. In: *POPL*. Ed. by Zhong Shao and Benjamin C. Pierce.

Zhao, Jianzhou and Steve Zdancewic (2012). “Mechanized Verification of Computing Dominators for Formalizing Compilers”. In: *CPP*. Ed. by Chris Hawblitzel and Dale Miller. Lecture Notes in Computer Science.

Example: Parameter Elimination


```
1  let i = 1 in
2  letrec f (n,i) =
3    if n > 0 then
4      let m = n - 1 in
5      let j = n * i in
6      f (m,j)
7    else
8      i
9  in
10   f (n,i)
```

```
1  let i = 1 in
2  letrec f (n,i) =
3    if n > 0 then
4      let m = n - 1 in
5      let i = n * i in
6      f (m,i)
7    else
8      i
9  in
10   f (n,i)
```

Example: Parameter Elimination


```
1  let i = 1 in
2  letrec f (n,i) =
3    if n > 0 then
4      let m = n - 1 in
5      let j = n * i in
6      f (m,j)
7    else
8      i
9  in
10   f (n,i)
```

```
1  let i = 1 in
2  letrec f (n,i) =
3    if n > 0 then
4      let m = n - 1 in
5      let i = n * i in
6      f (m,i)
7    else
8      i
9  in
10   f (n,i)
```

Example: Parameter Elimination

SAARLAND UNIVERSITY
COMPUTER SCIENCE

```
1  let i = 1 in
2  letrec f (n,i) =
3    if n > 0 then
4      let m = n - 1 in
5      let j = n * i in
6      f (m,j)
7    else
8      i
9  in
10   f (n,i)
```

```
1  i := 1;
2  letrec f (n,i) =
3    if n > 0 then
4      m := n - 1;
5      i := n * i;
6      f (m,i)
7    else
8      i
9  in
10   f (n,i)
```

Example: Parameter Elimination

SAARLAND UNIVERSITY
COMPUTER SCIENCE

```
1  let i = 1 in
2  letrec f (n,i) =
3    if n > 0 then
4      let m = n - 1 in
5      let j = n * i in
6      f (m,j)
7    else
8      i
9  in
10   f (n,i)
```

```
1  i := 1;
2  letrec f () =
3    if n > 0 then
4      m := n - 1;
5      i := n * i;
6      n := m;
7      f ()
8    else
9      i
10 in
11   f ()
```

Figure : Semantics of IL/F, standard presentation

$$\text{Std-Op}\ \frac{V \vdash e \Downarrow v}{L, V, \texttt{let}\ x = e\ \texttt{in}\ s \longrightarrow L, V_v^x, s}$$

$$\text{Std-If}\ \frac{val2bool(Vx) = i}{L, V, \texttt{if}\ x\ \texttt{then}\ s_0\ \texttt{else}\ s_1 \longrightarrow L, V, s_i}$$

$$\text{Std-Let}\ \frac{c = (L, V, s, \overline{x})}{L, V, \texttt{fun}\ f\ \overline{x} = s\ \texttt{in}\ t \longrightarrow L_c^f, V, t}$$

$$\text{Std-App}\ \frac{Lf = (L', V', \overline{x}, s)}{L, V, f\ \overline{y} \longrightarrow {L'}_{Lf}^{f}, {V'}_{V\overline{y}}^{\overline{x}}, s}$$