Low-Level Code and High-Level Theorems

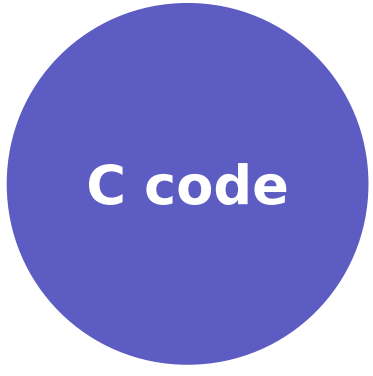
Sascha Böhme

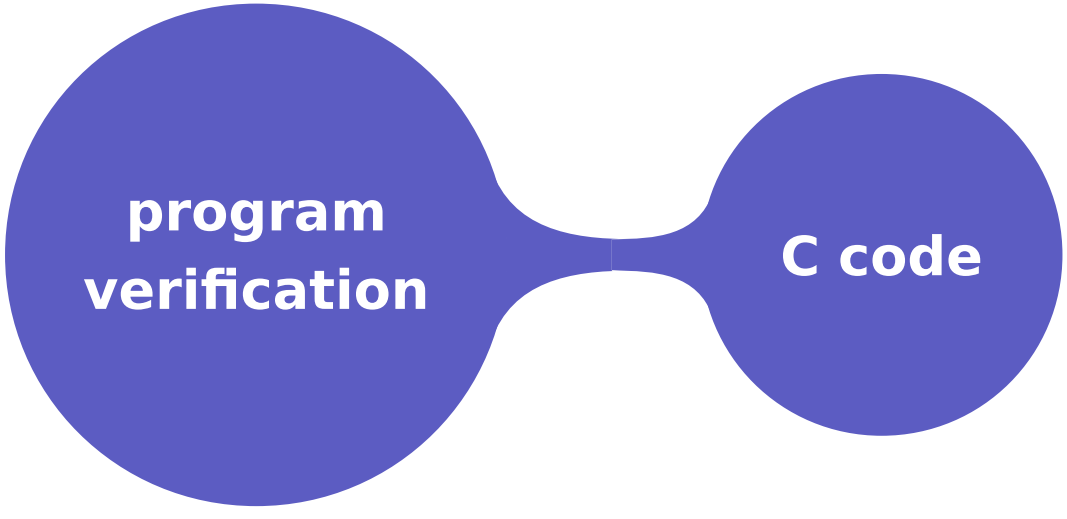
Technische Universität München, Germany

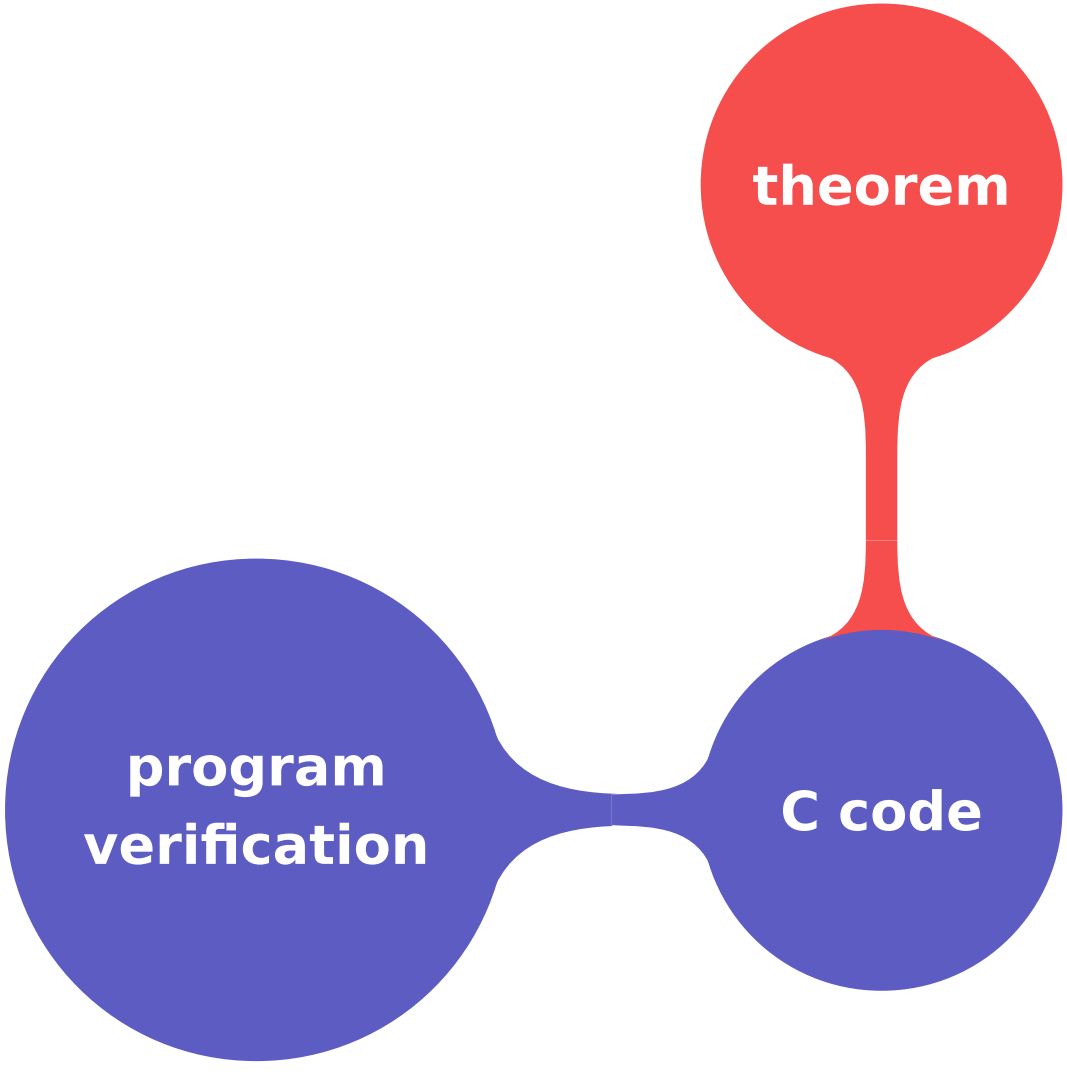
Joint work with

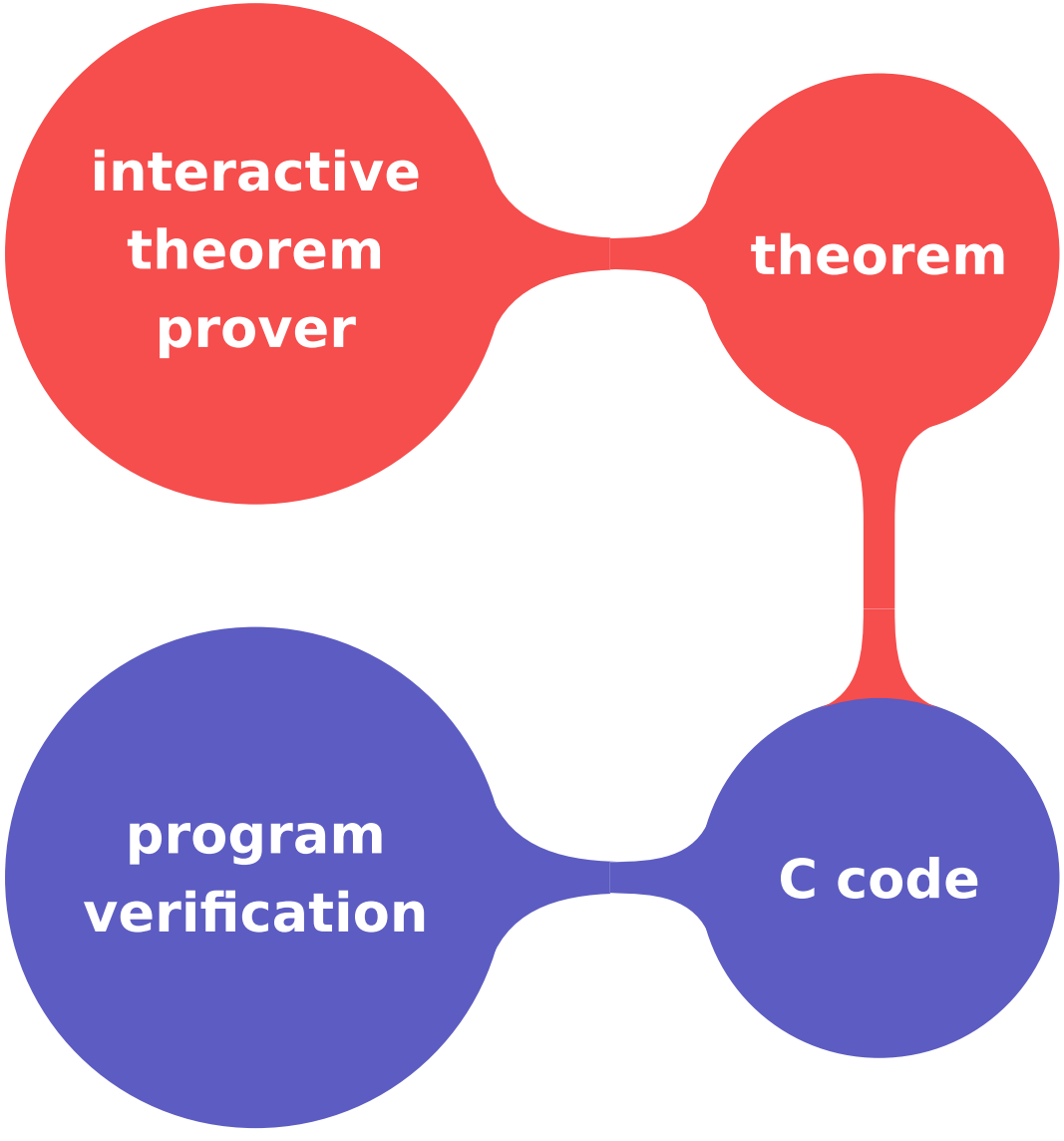
Eyad Alkassar[1], Ernie Cohen[2], Kurt Mehlhorn[3] and Christine Rizkallah[3]

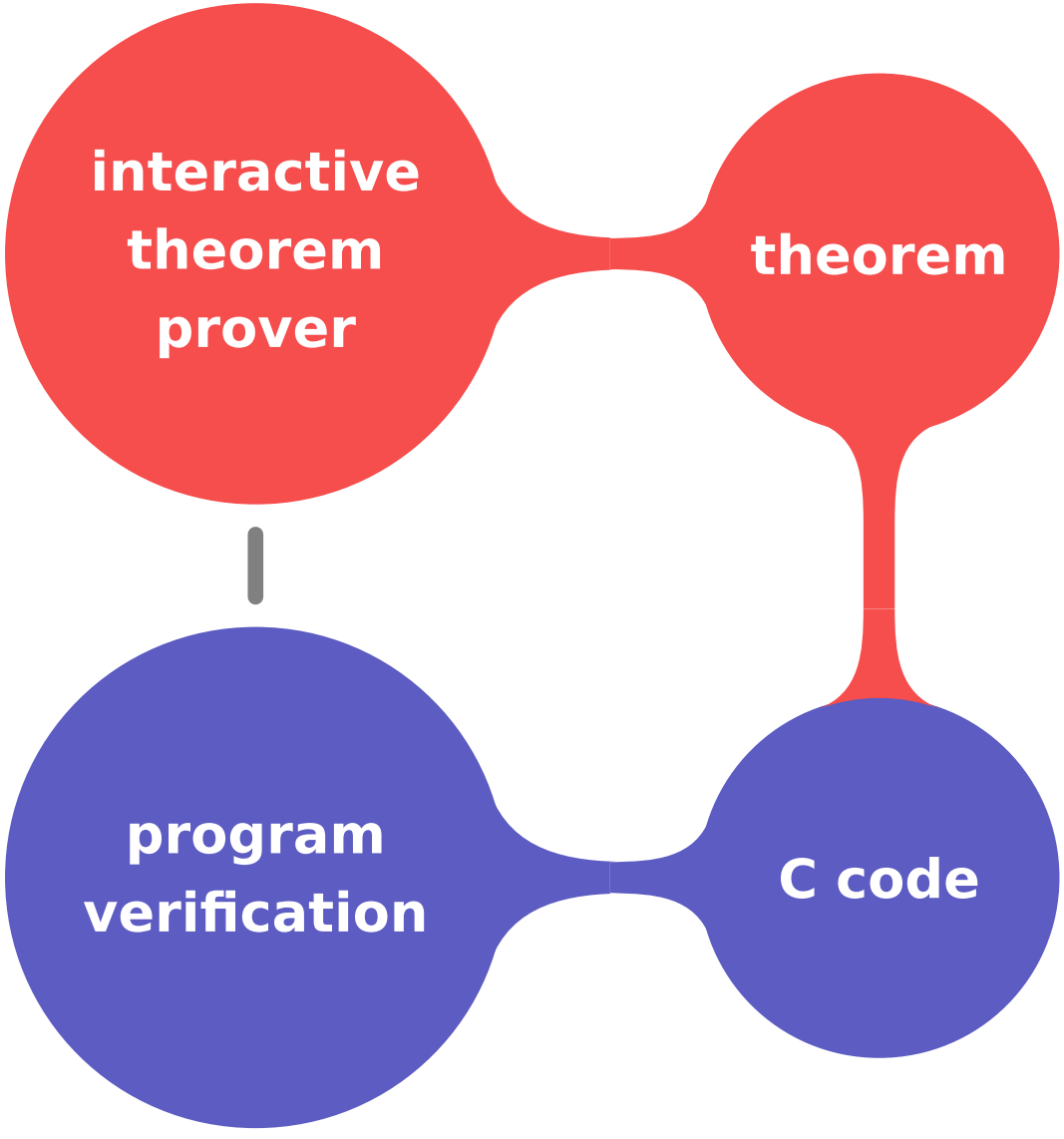
[1]Universität des Saarlandes, Germany
[2]European Microsoft Innovation Center, Aachen, Germany
[3]Max-Planck-Institut für Informatik, Saarbrücken, Germany

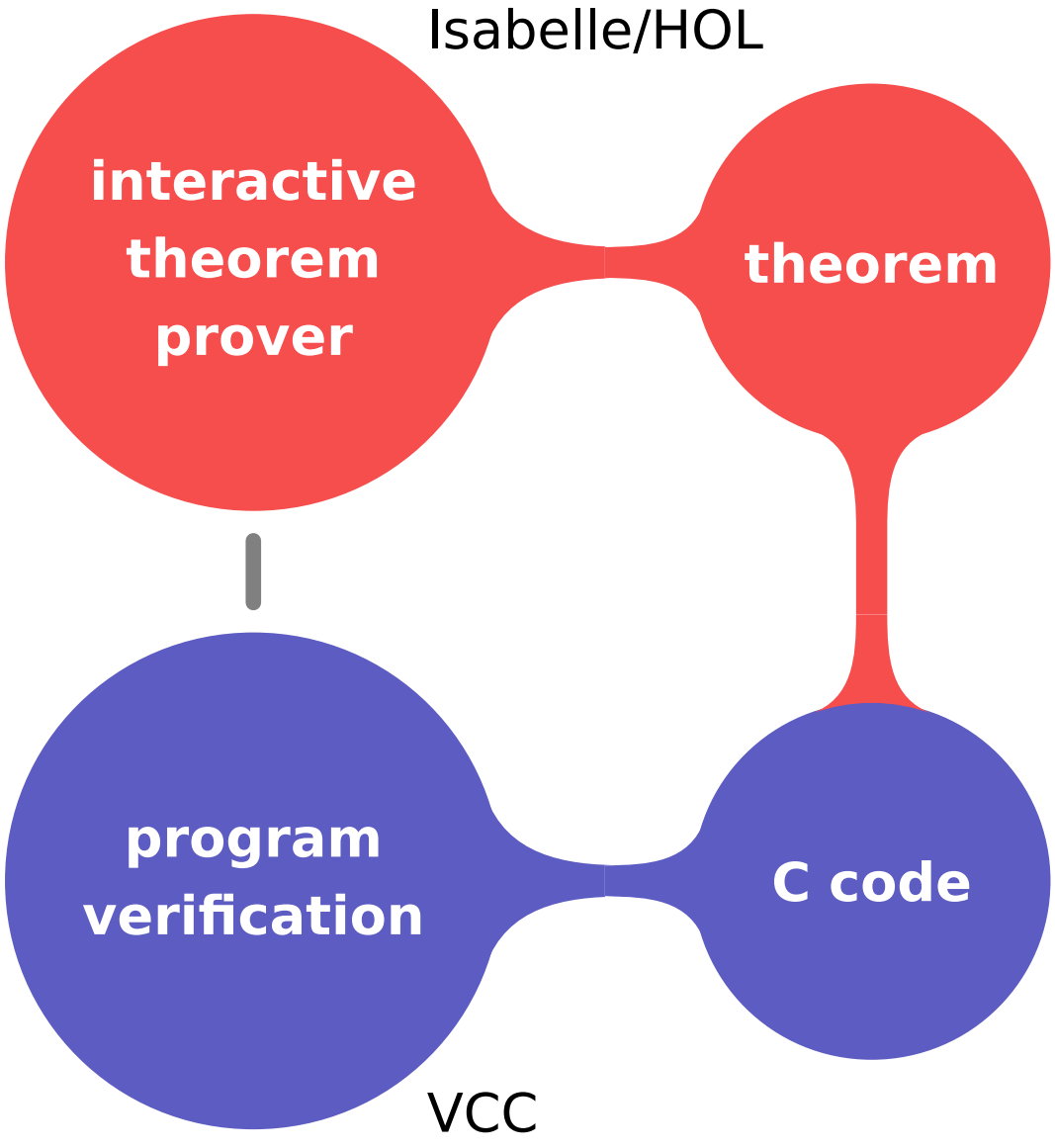

C code


program
verification
C code


theorem
program
verification
C code


interactive
theorem
prover
theorem
program
verification
C code


interactive
theorem
prover
theorem
program
verification
C code


Isabelle/HOL
interactive theorem prover
theorem
program verification
C code
VCC

VCC:

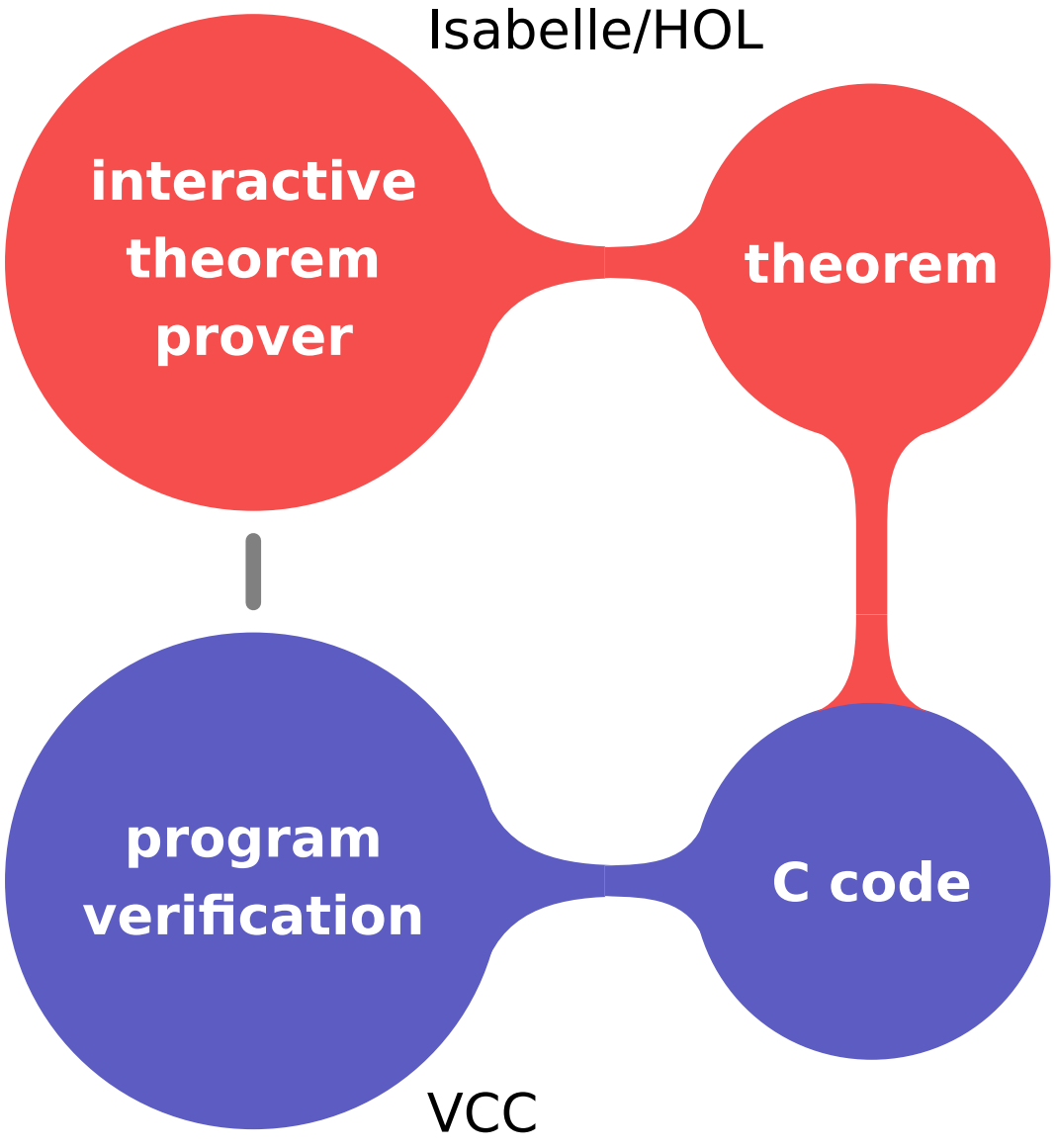
- assertional verifier for full C
- first-order logic as specification language
- fully automatic thanks to Boogie and Z3
- specification by code annotations
- function contracts, object invariants
- ghost code, ghost functions

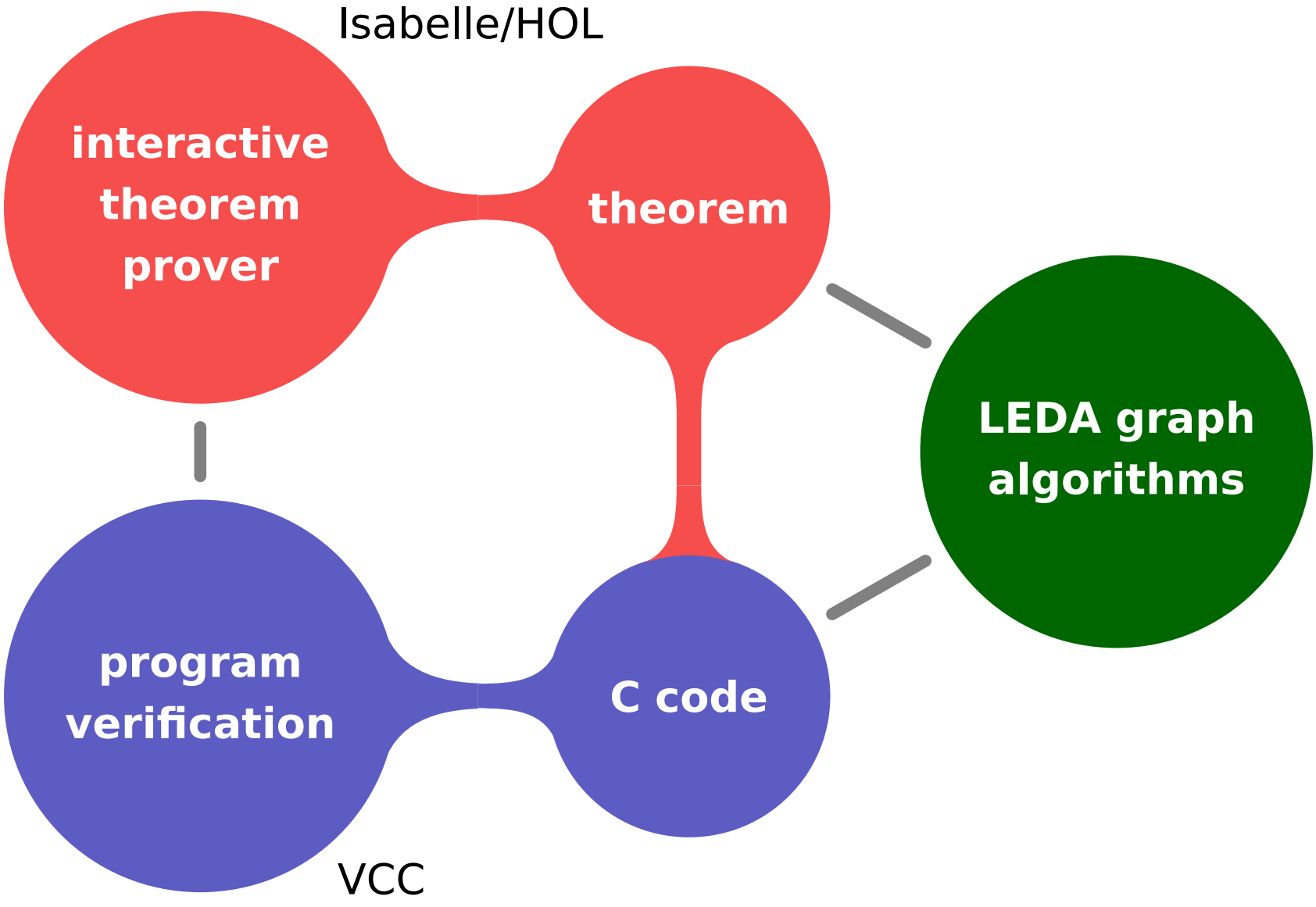
VCC:

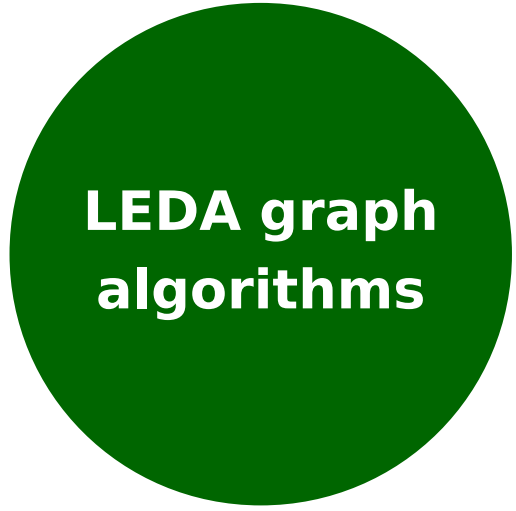
- assertional verifier for full C
- first-order logic as specification language
- fully automatic thanks to Boogie and Z3
- specification by code annotations
- function contracts, object invariants
- ghost code, ghost functions

Isabelle/HOL:

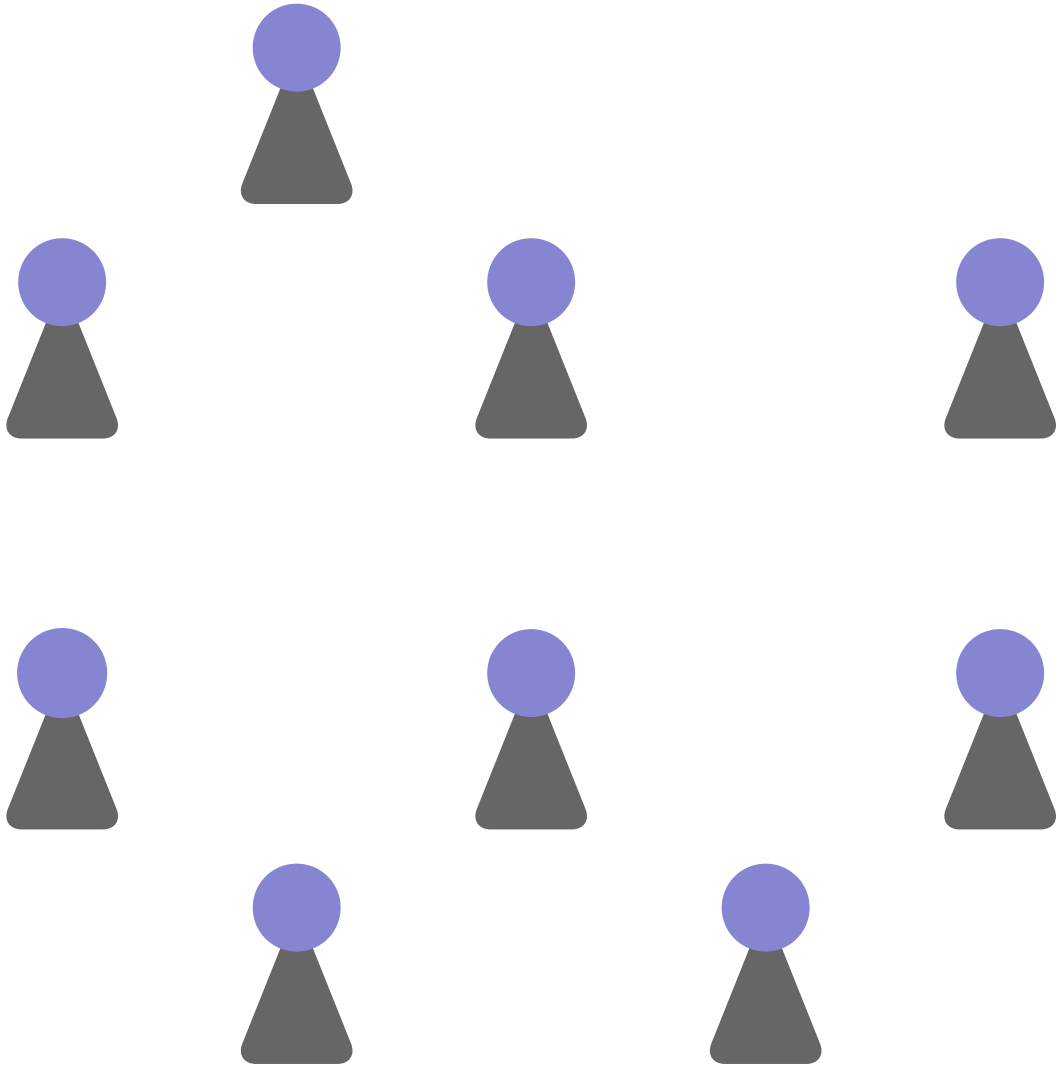
- interactive theorem prover for higher-order logic
- rich set of formalized mathematics
- various automated provers


Isabelle/HOL
interactive theorem prover
theorem
program verification
C code
VCC


Isabelle/HOL
interactive theorem prover
theorem
LEDA graph algorithms
program verification
C code
VCC

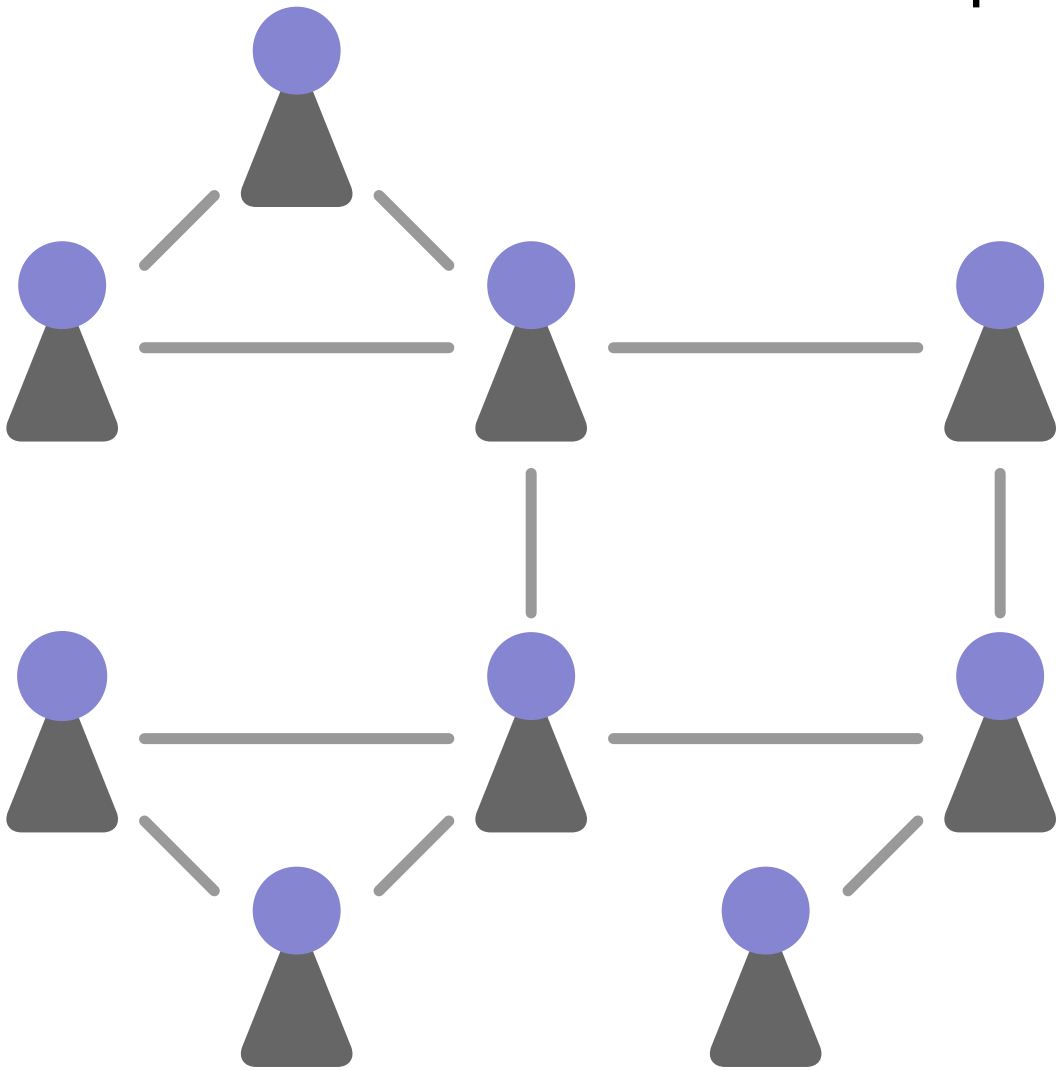

LEDA graph
algorithms

Programmers



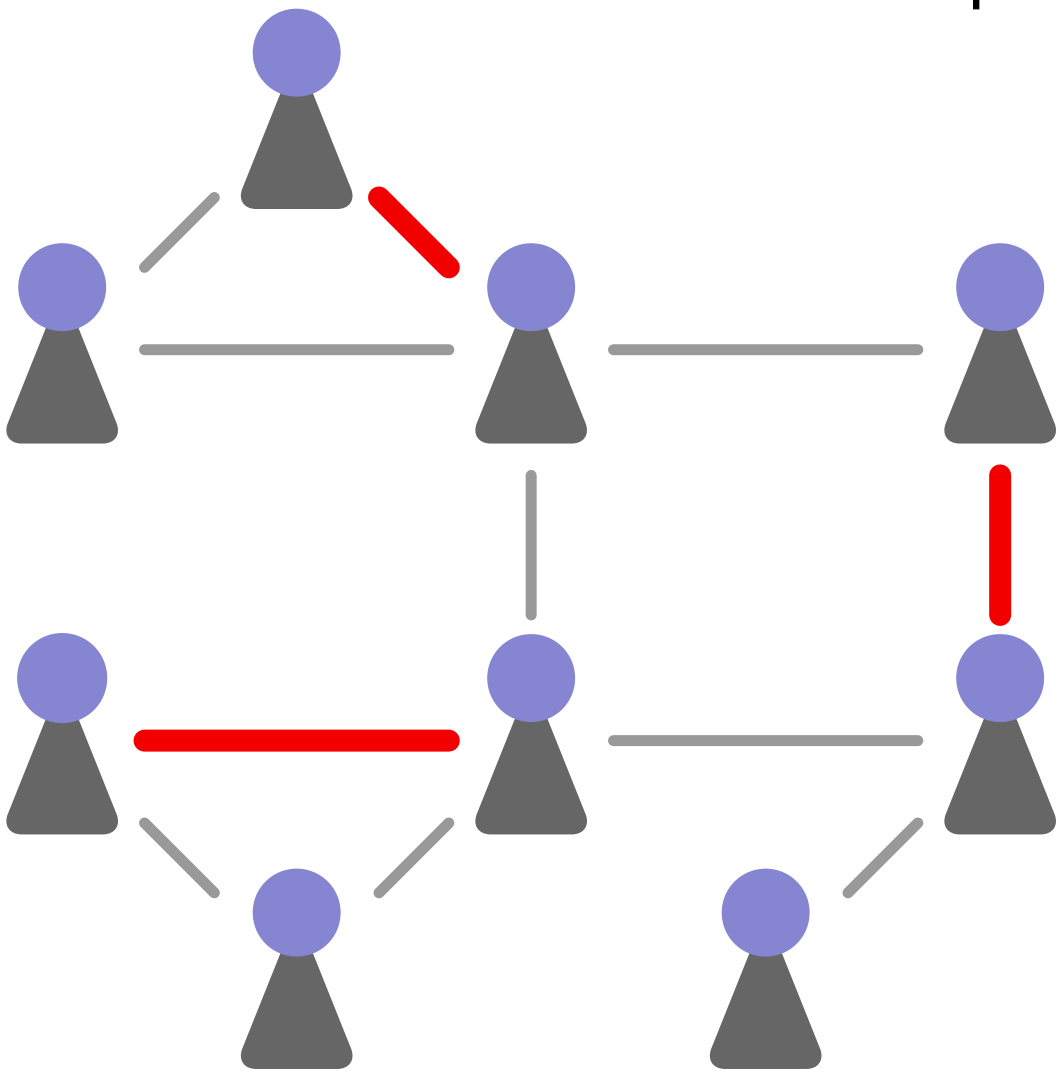
Social programmers

Graph



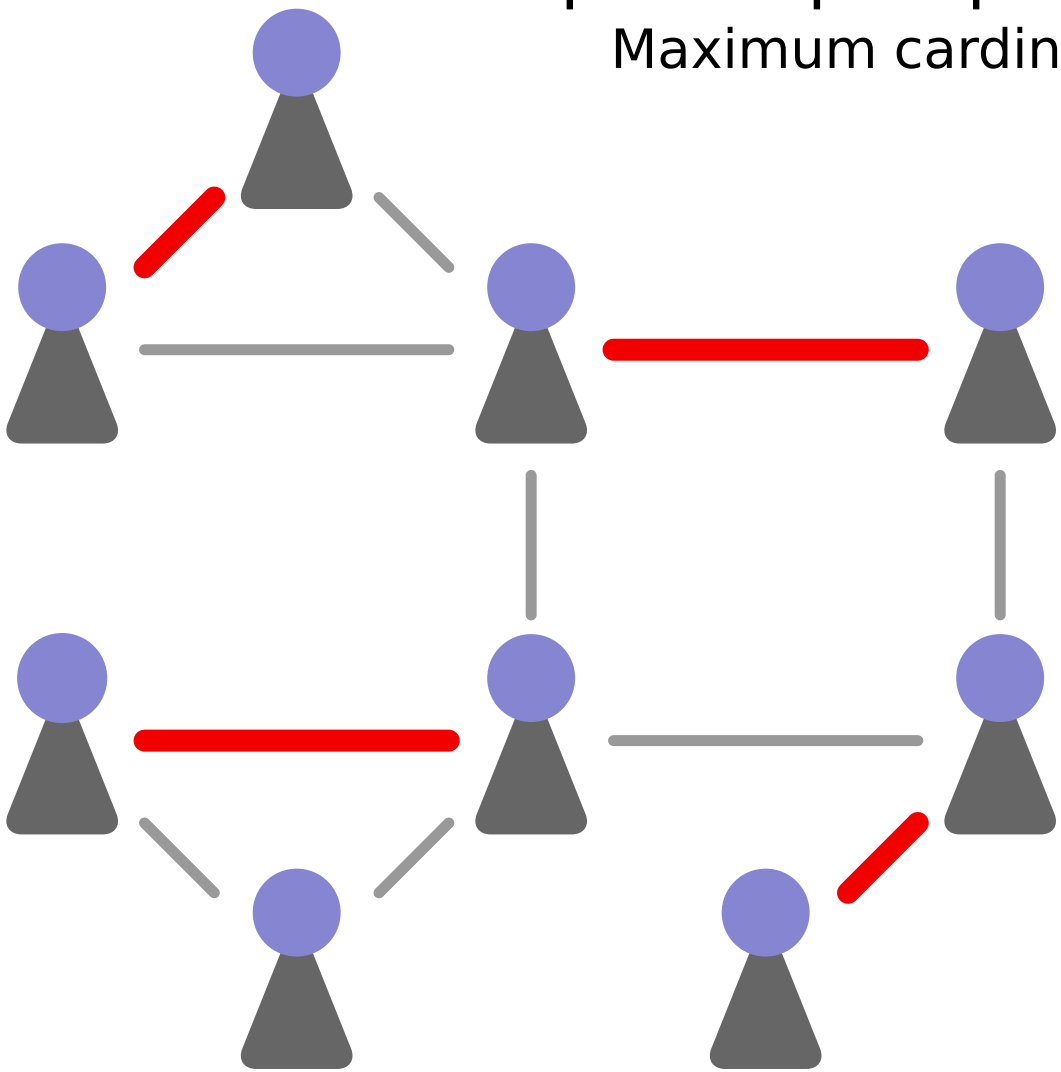
Pair programming

Matching



Optimal pair programming

Maximum cardinality matching



Definitions

Matching:

- a graph
- no edge is incident to another edge

Definitions

Matching:

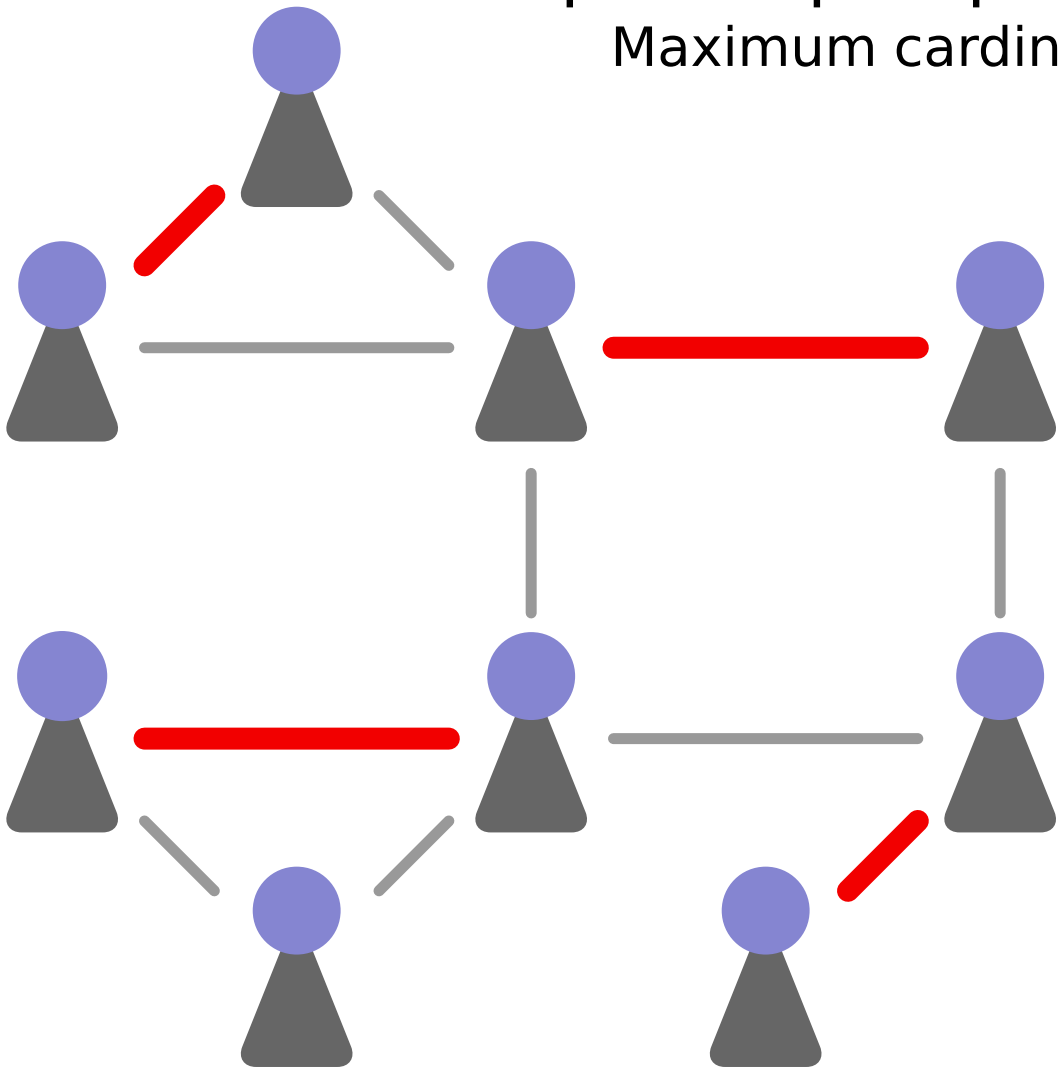
- a graph
- no edge is incident to another edge

Odd-set cover:

- labeling of nodes
- every edge is incident to a node labeled 1 or connects two nodes labeled i (with $i \geq 2$)

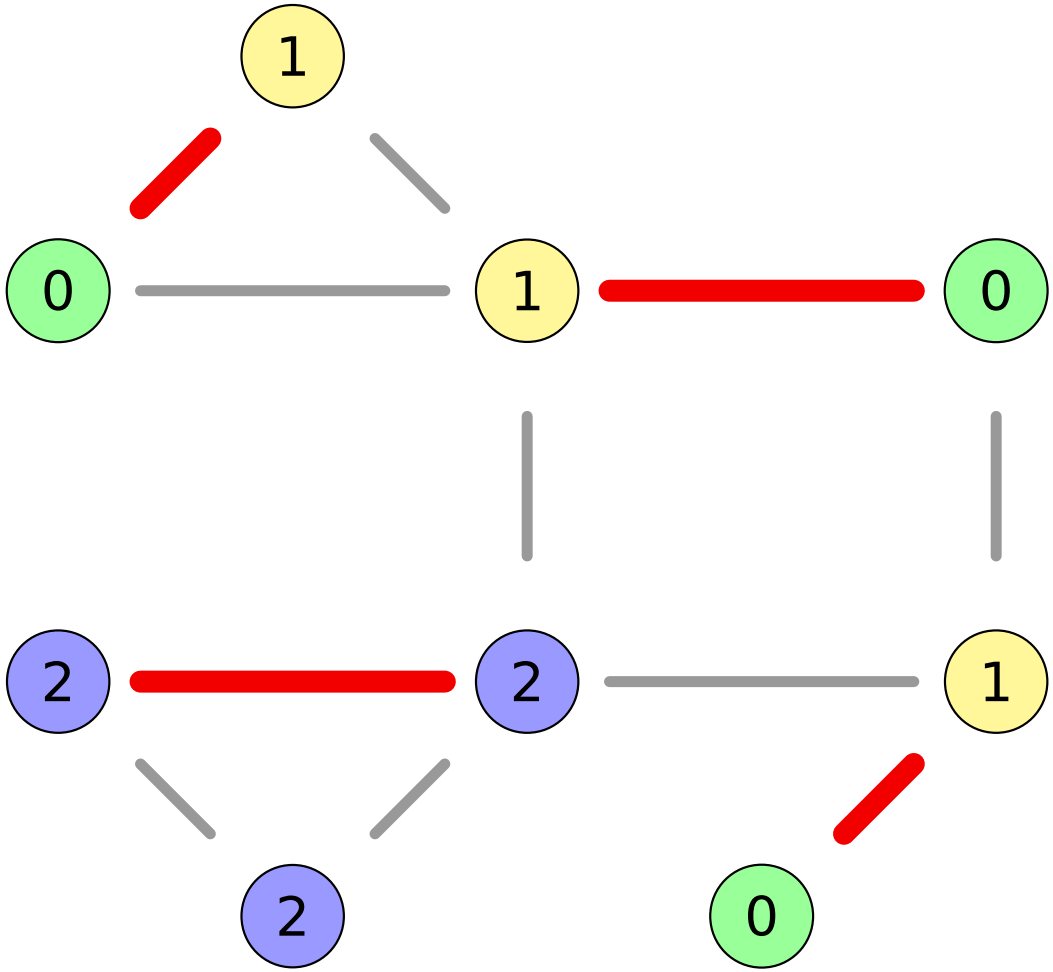
Optimal pair programming

Maximum cardinality matching



Certificate

Odd-set cover

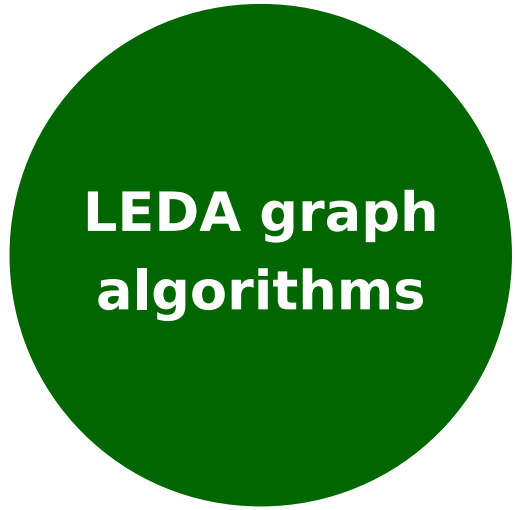


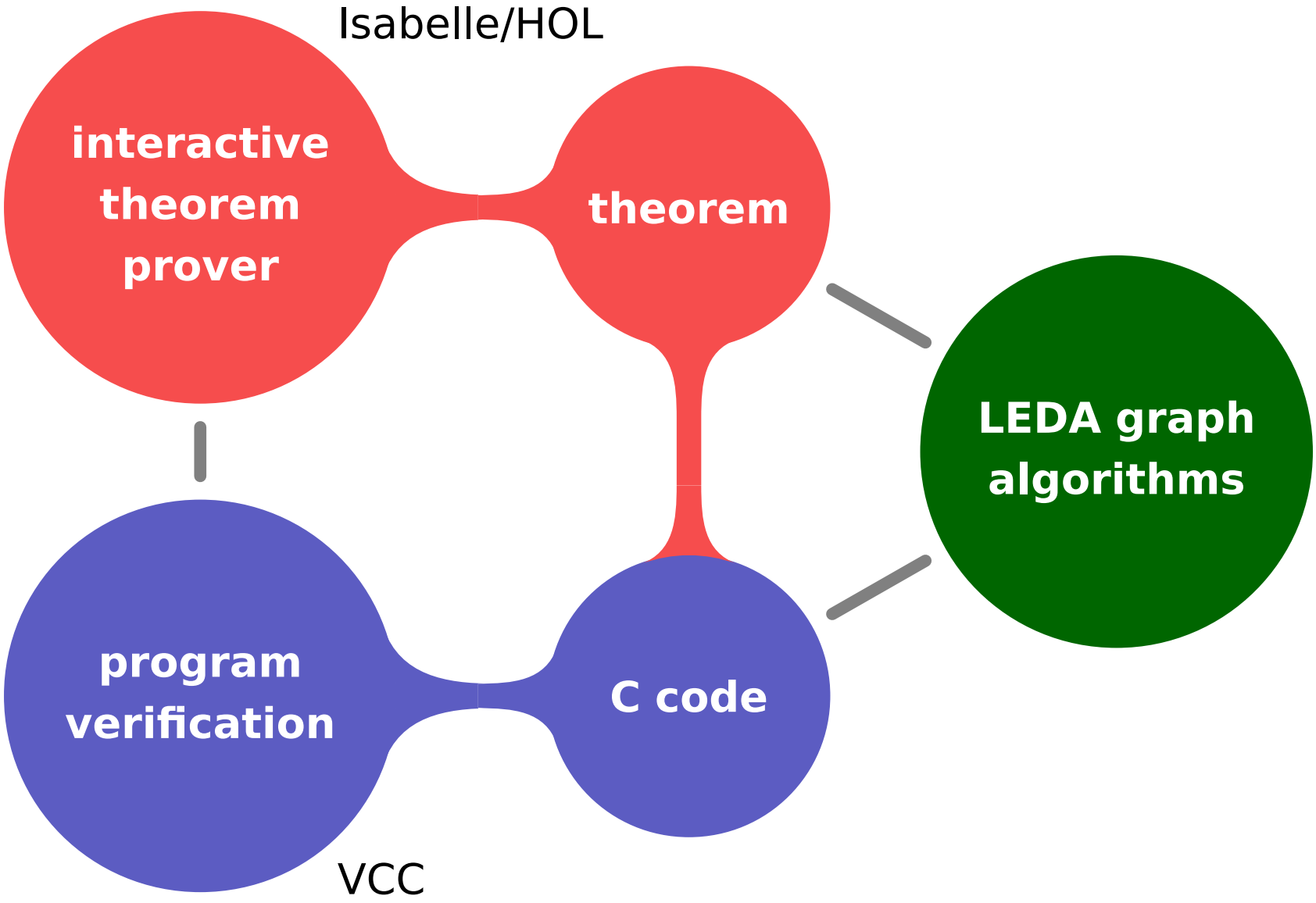
Theorem

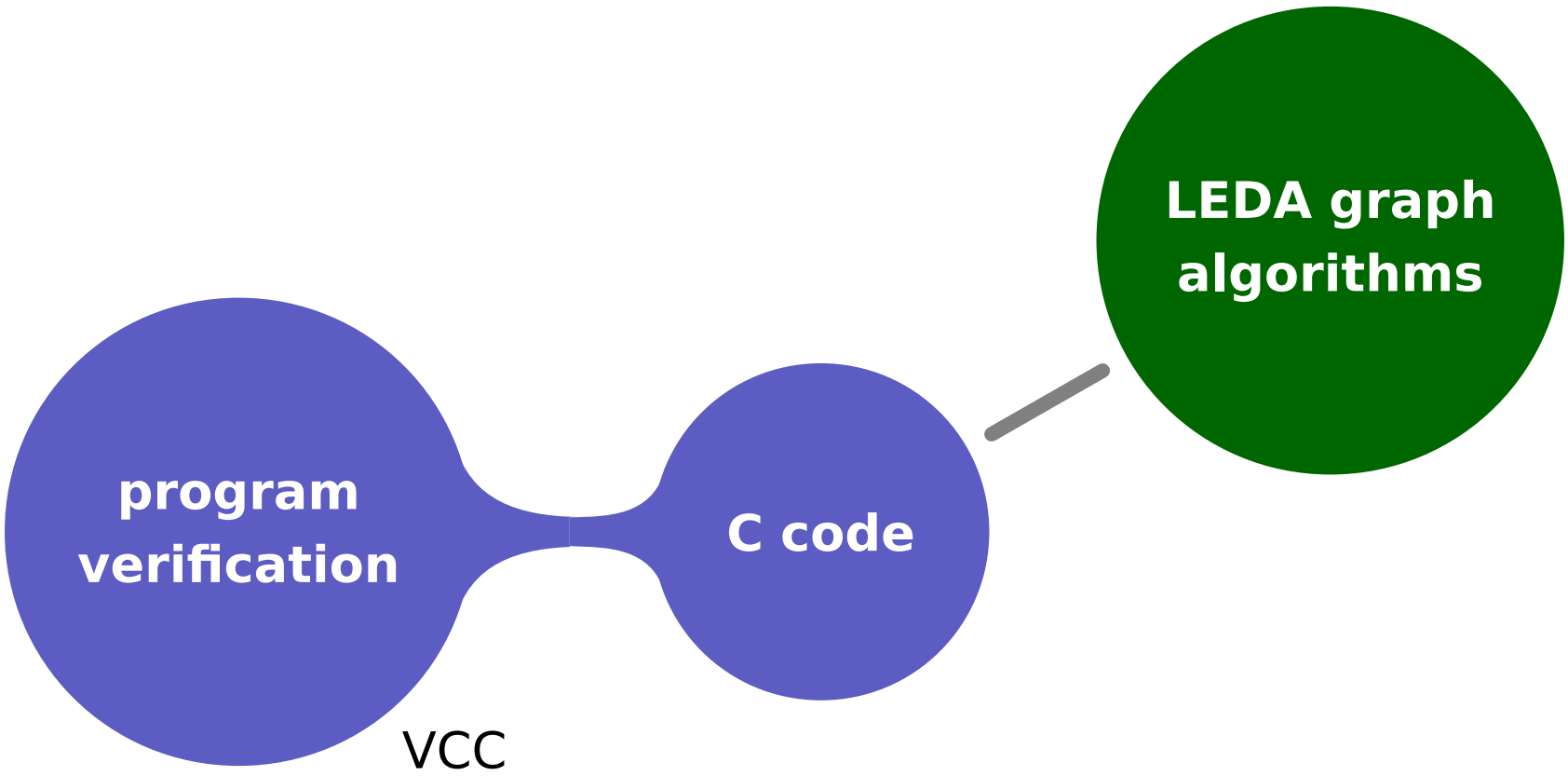
The maximum cardinality of a graph matching is

$$n_1 + \sum_{i \geq 2} \lfloor n_i / 2 \rfloor$$

where n_i is the number of nodes labeled i by an odd-set cover.


LEDA graph
algorithms


Isabelle/HOL
interactive theorem prover
theorem
LEDA graph algorithms
program verification
C code
VCC


LEDA graph algorithms
program verification
C code
VCC

Maximum Cardinality Matching Checker

C Code

Maximum Cardinality Matching Checker

C Code

Given:

- graph G
- graph M (maximum cardinality matching)
- labeling OSC (odd-set cover)

Maximum Cardinality Matching Checker

C Code

Given:

- graph G
- graph M (maximum cardinality matching)
- labeling OSC (odd-set cover)

Check:

- M is a matching
- M is a subset of G
- OSC is an odd-set cover of G
- M is maximal wrt. G and OSC

Maximum Cardinality Matching Checker

C Code

Given:

- graph G
- graph M (maximum cardinality matching)
- labeling OSC (odd-set cover)

Check:

- M is a matching
- M is a subset of G
- OSC is an odd-set cover of G
- M is maximal wrt. G and OSC

Implementation:

- straightforward

Maximum Cardinality Matching Checker

Specification

```
struct graph {
  edge * es; unsigned n_edges, n_nodes;




}
```

Maximum Cardinality Matching Checker

Specification

```
struct graph {
  edge * es; unsigned n_edges, n_nodes;
   invariant(∀(unsigned e; e < n_edges ⟶
    es[e].s < n_nodes ∧ es[e].t < n_nodes ∧
    es[e].s ≠ es[e].t))
}
```

Maximum Cardinality Matching Checker

Specification

```
struct graph {
  edge * es; unsigned n_edges, n_nodes;
   invariant(∀(unsigned e; e < n_edges ⟶
    es[e].s < n_nodes ∧ es[e].t < n_nodes ∧
    es[e].s ≠ es[e].t))
}


spec(bool spec_is_osc(graph * G, unsigned * osc)
  returns(... ∧
    ∀(unsigned e; e < G->n_edges ⟶
      osc[G->es[e].s] = 1 ∨ osc[G->es[e].t] = 1 ∨
      (osc[G->es[e].t] = osc[G->es[e].s] ∧
        osc[G->es[e].t] > 1)));)
```

What is proved?

$$\texttt{check(G, M, osc)} = \texttt{true} \longleftrightarrow |\mathsf{M}| = n_1 + \sum_{i \geq 2} \lfloor n_i/2 \rfloor$$

What is proved?

$$\texttt{check(G, M, osc) = true} \longleftrightarrow |\mathrm{M}| = n_1 + \sum_{i \geq 2} \lfloor n_i/2 \rfloor$$

What is missing?

$$|\mathrm{M}| = n_1 + \sum_{i \geq 2} \lfloor n_i/2 \rfloor \longrightarrow$$

$$(\forall\ \mathrm{M'}.\ \mathrm{is\_matching(M')} \wedge \mathrm{is\_subset(G,M')} \longrightarrow |\mathrm{M'}| \leq |\mathrm{M}|)$$

What is proved?

$$\texttt{check(G, M, osc) = true} \longleftrightarrow |\mathrm{M}| = n_1 + \sum_{i \geq 2} \lfloor n_i/2 \rfloor$$

What is missing?

$$|\mathrm{M}| = n_1 + \sum_{i \geq 2} \lfloor n_i/2 \rfloor \longrightarrow$$

$$(\forall\ \mathrm{M'}.\ \mathrm{is\_matching(M')} \wedge \mathrm{is\_subset(G,M')} \longrightarrow |\mathrm{M'}| \leq |\mathrm{M}|)$$

VCC:

- good at low-level code verification
- not much support for high-level proofs

What is proved?

$$\texttt{check(G, M, osc) = true} \longleftrightarrow |\mathrm{M}| = n_1 + \sum_{i \geq 2} \lfloor n_i/2 \rfloor$$

What is missing?

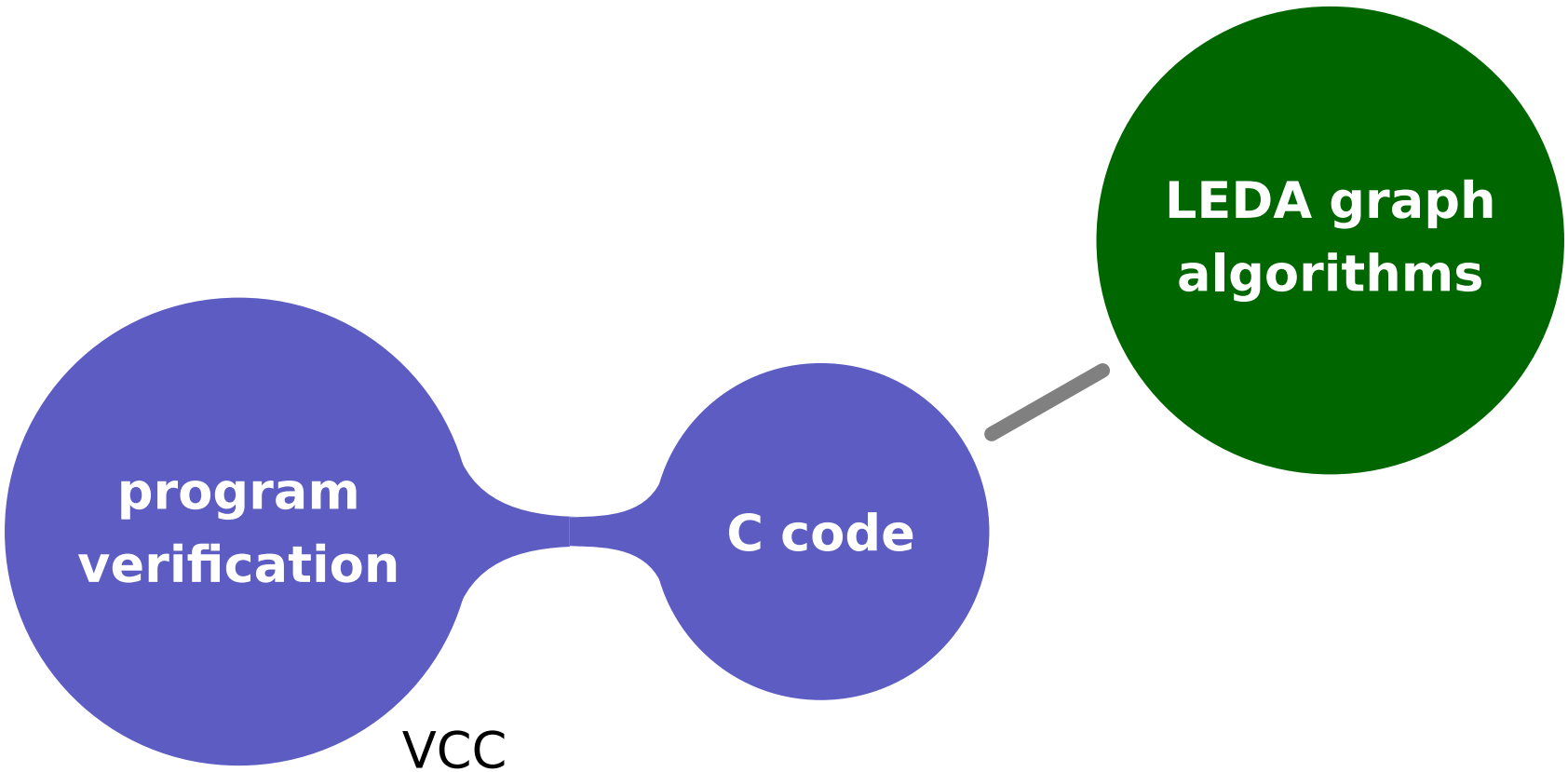
$$|\mathrm{M}| = n_1 + \sum_{i \geq 2} \lfloor n_i/2 \rfloor \longrightarrow$$

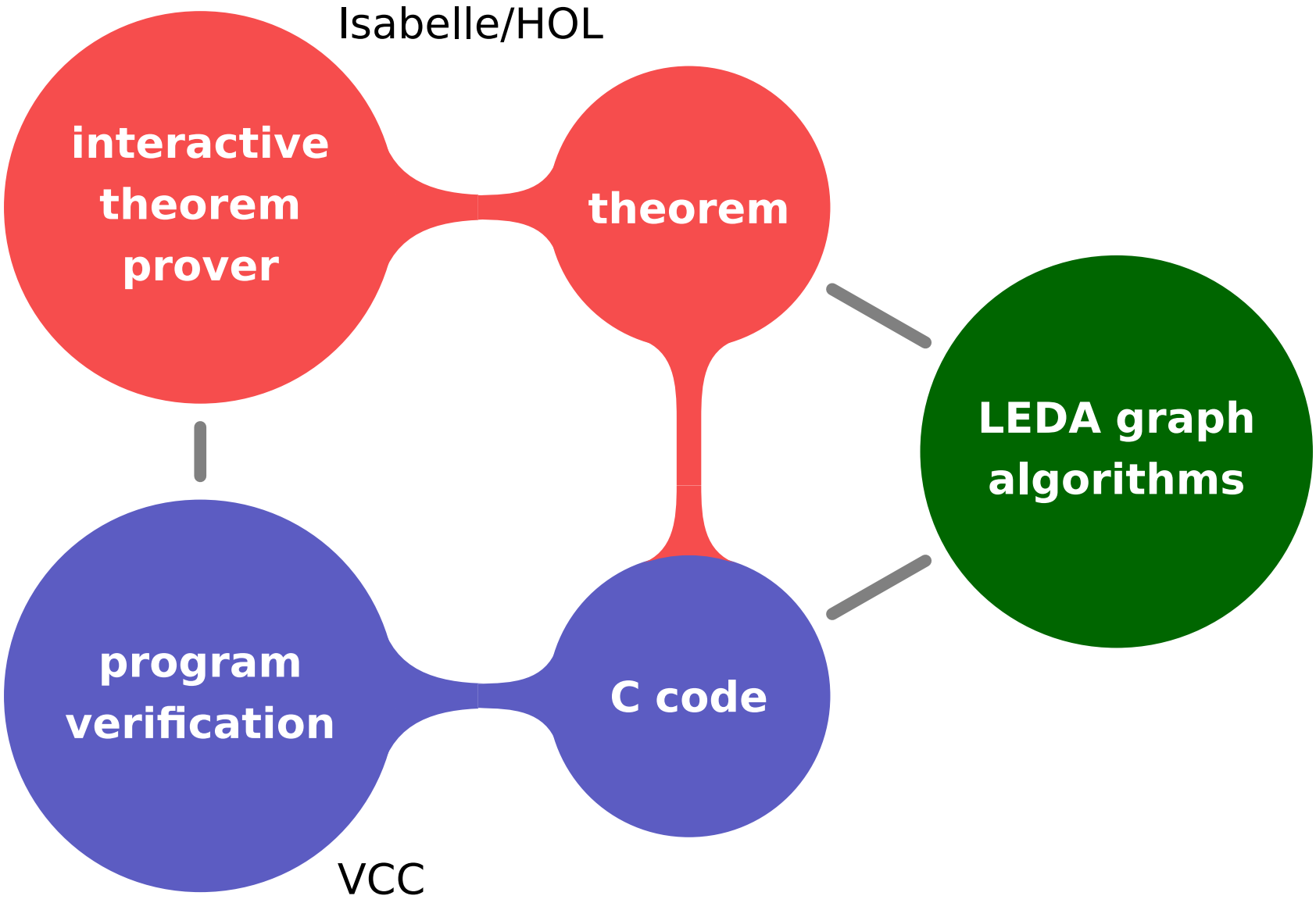
$$(\forall\ \mathrm{M'}.\ \mathrm{is\_matching(M')} \wedge \mathrm{is\_subset(G,M')} \longrightarrow |\mathrm{M'}| \leq |\mathrm{M}|)$$

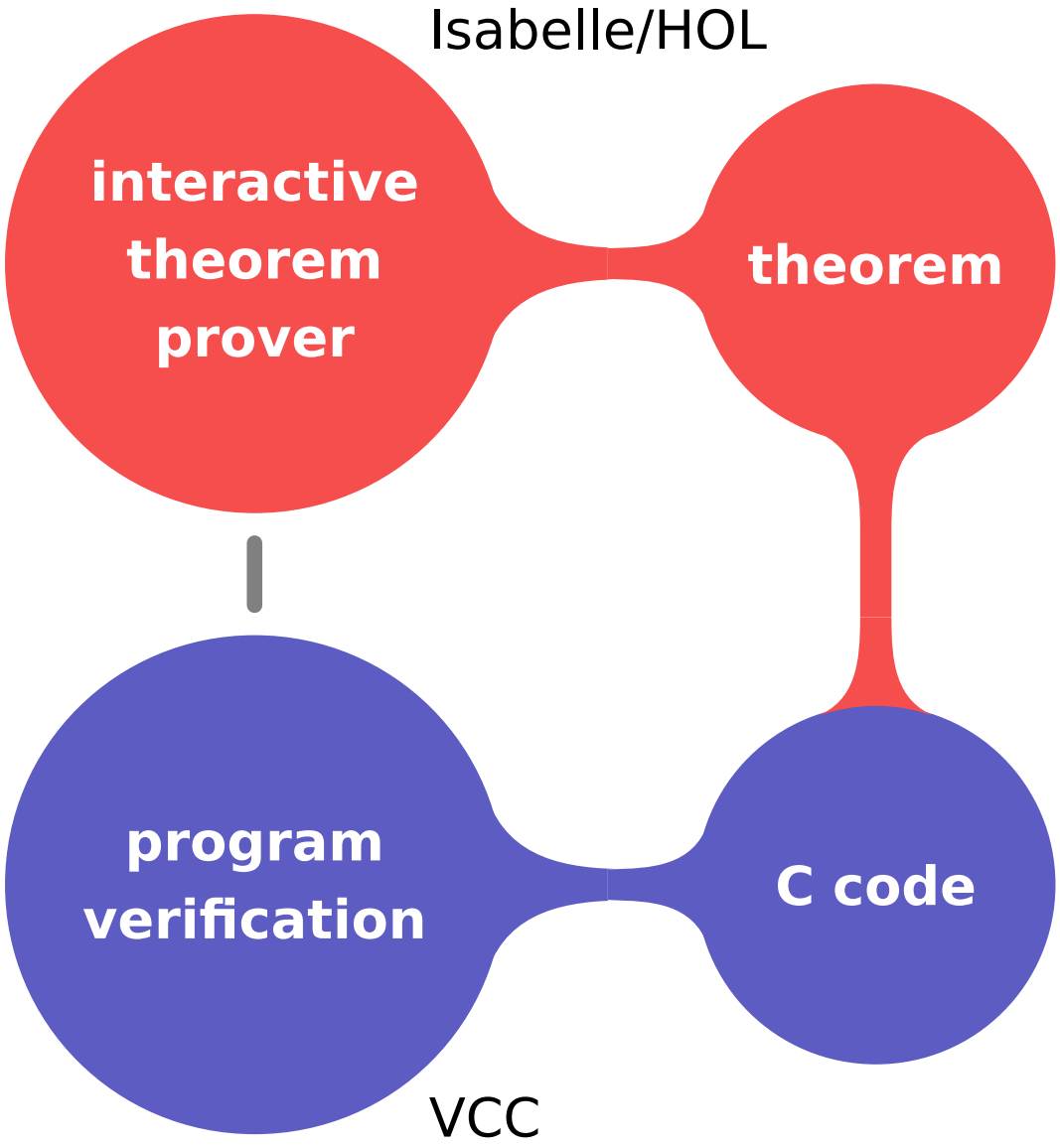
VCC:

- good at low-level code verification
- not much support for high-level proofs

Requires abstraction!


LEDA graph algorithms
program verification
C code
VCC


Isabelle/HOL
interactive theorem prover
theorem
LEDA graph algorithms
program verification
C code
VCC


Isabelle/HOL
interactive theorem prover
theorem
program verification
C code
VCC

VCC

Isabelle/HOL

Isabelle/HOL

VCC

concrete property

```
assert(p(x));
```

Isabelle/HOL

VCC

abstract property

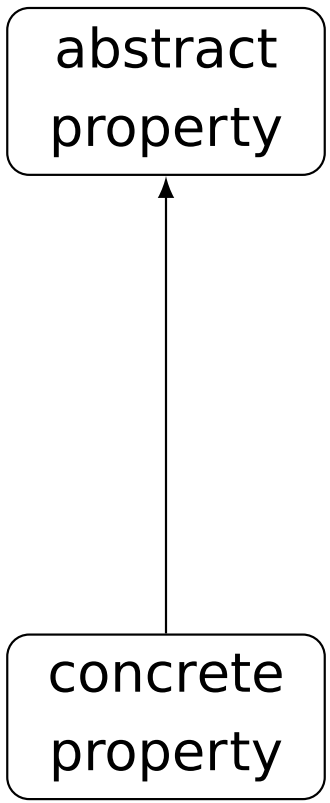
```
spec(bool P(X) returns(...);)
```

concrete property

```
assert(p(x));
```

Isabelle/HOL

VCC



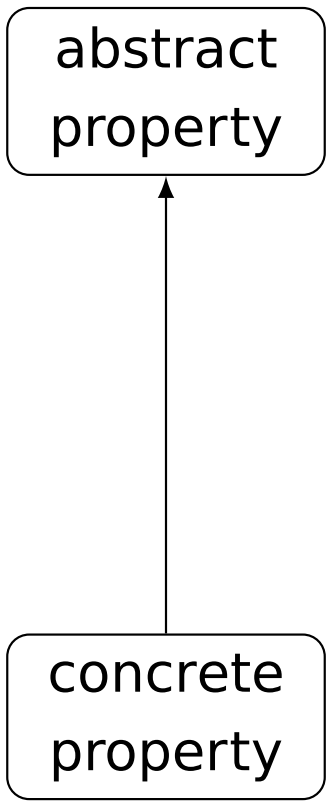

```
spec(bool P(X) returns(...);)
```

```
spec(void P_equivalence(x)
  ensures(p(x) ⟺ P(abs(x)));)
```

```
assert(p(x));
```

Isabelle/HOL

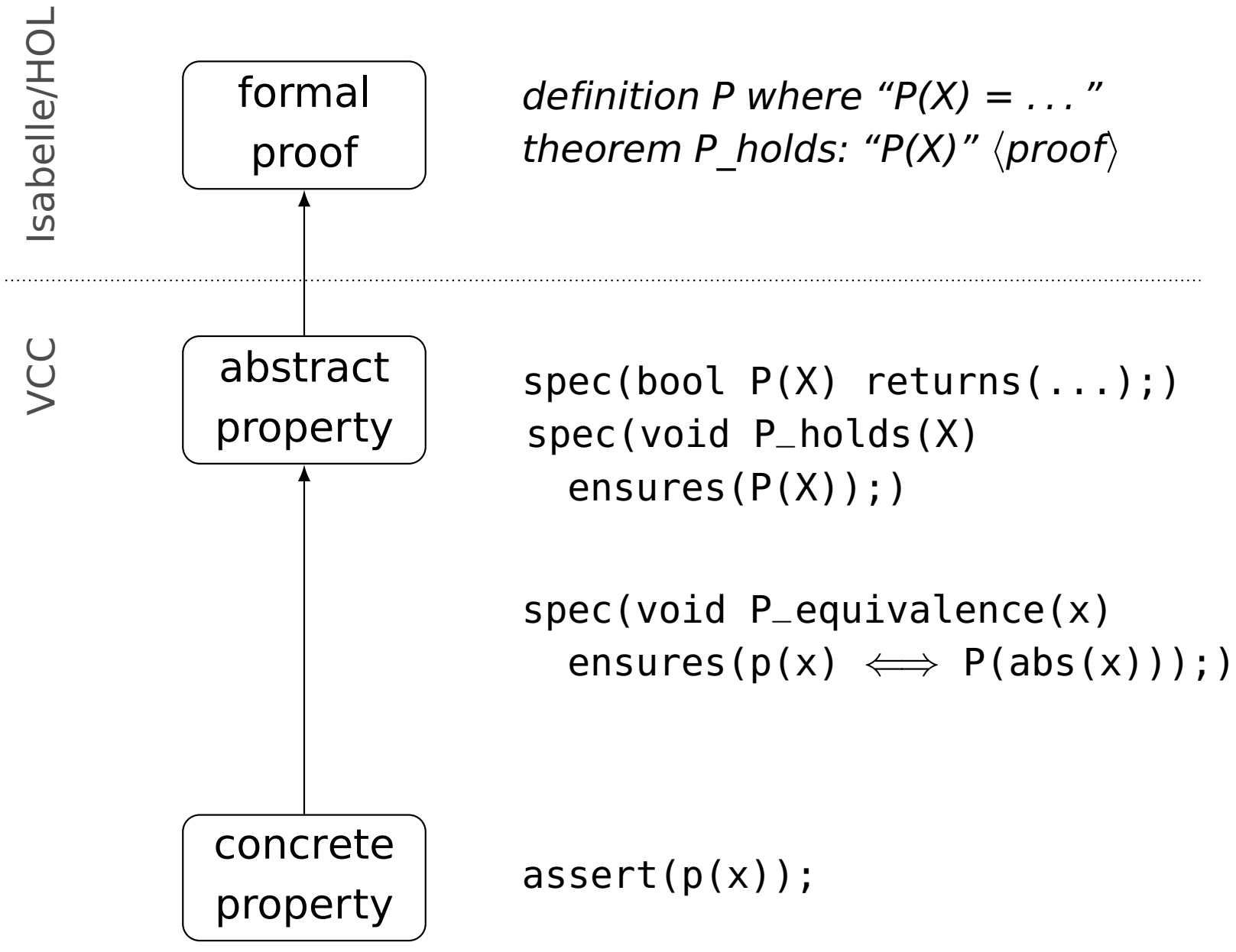
VCC




```
spec(bool P(X) returns(...);)
spec(void P_holds(X)
  ensures(P(X));)

spec(void P_equivalence(x)
  ensures(p(x) ⟺ P(abs(x)));)

assert(p(x));
```


Isabelle/HOL
formal proof
definition P where “P(X) = . . . ”
theorem P_holds: “P(X)” ⟨proof⟩
VCC
abstract property
spec(bool P(X) returns(...);)
spec(void P_holds(X)
ensures(P(X));)
spec(void P_equivalence(x)
ensures(p(x) ⟺ P(abs(x)));)
concrete property
assert(p(x));

Maximum Cardinality Matching Checker

vcc

Maximum Cardinality Matching Checker

VCC

```
struct abs_graph {
  abs_edge es[mathint];
  mathint n_edges, n_nodes;
};
```

Maximum Cardinality Matching Checker

VCC

```
struct abs_graph {
  abs_edge es[mathint];
  mathint n_edges, n_nodes;
};

spec(abs_graph abs_g(graph * G) ensures(...);)
```

Maximum Cardinality Matching Checker

VCC

```
struct abs_graph {
  abs_edge es[mathint];
  mathint n_edges, n_nodes;
};

spec(abs_graph abs_g(graph * G) ensures(...);)

spec(bool abs_is_osc(abs_graph G, abs_fun osc)
  ensures(...);)
```

Maximum Cardinality Matching Checker

VCC

```
struct abs_graph {
  abs_edge es[mathint];
  mathint n_edges, n_nodes;
};

spec(abs_graph abs_g(graph * G) ensures(...);)

spec(bool abs_is_osc(abs_graph G, abs_fun osc)
  ensures(...);)

void is_osc_equivalence(graph * G, unsigned * osc)
  ensures(
    spec_is_osc(G, osc) ⟺
    abs_is_osc(abs_g(G), abs_f(osc)));
```

Maximum Cardinality Matching Checker

Isabelle/HOL

Maximum Cardinality Matching Checker

Isabelle/HOL

$$
\begin{aligned}
&\textit{record abs\_graph} = \\
&\quad \textit{es} :: \textit{int} \Rightarrow \textit{abs\_edge} \\
&\quad \textit{n\_edges}, \textit{n\_nodes} :: \textit{int}
\end{aligned}
$$

Maximum Cardinality Matching Checker

Isabelle/HOL

record abs_graph =
es :: int ⇒ abs_edge
n_edges, n_nodes :: int

definition abs_is_osc where "abs_is_osc G osc =
(. . . ∧
(∀ e. 0 ≤ e ∧ e < n_edges G ⟶
osc (s (es G e)) = 1 ∨ osc (t (es G e)) = 1 ∨
(osc (t (es G e)) = osc (s (es G e)) ∧ osc (t (es G e)) > 1)))"

Maximum Cardinality Matching Checker

Isabelle/HOL

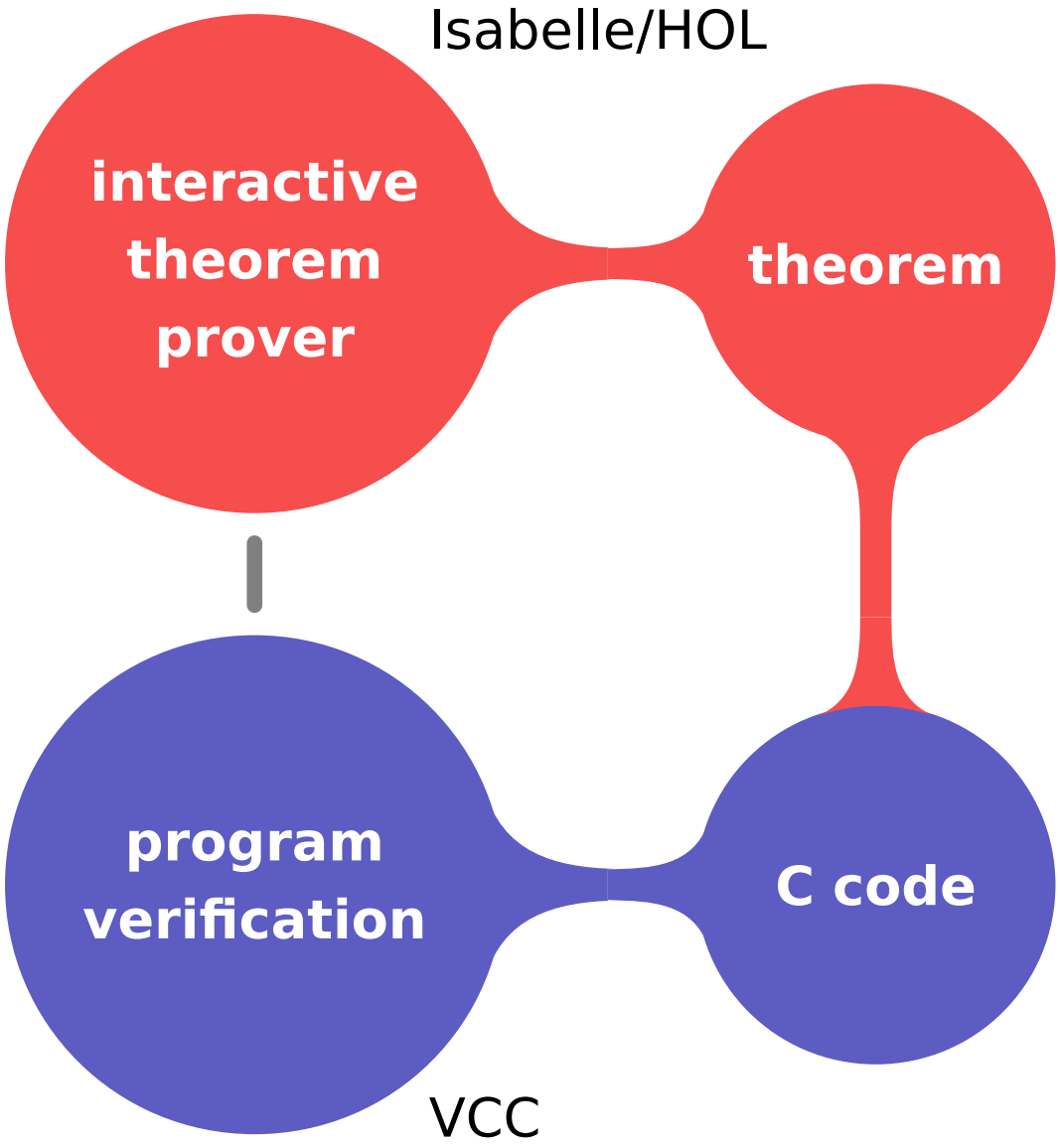
record abs_graph =
es :: int ⇒ abs_edge
n_edges, n_nodes :: int

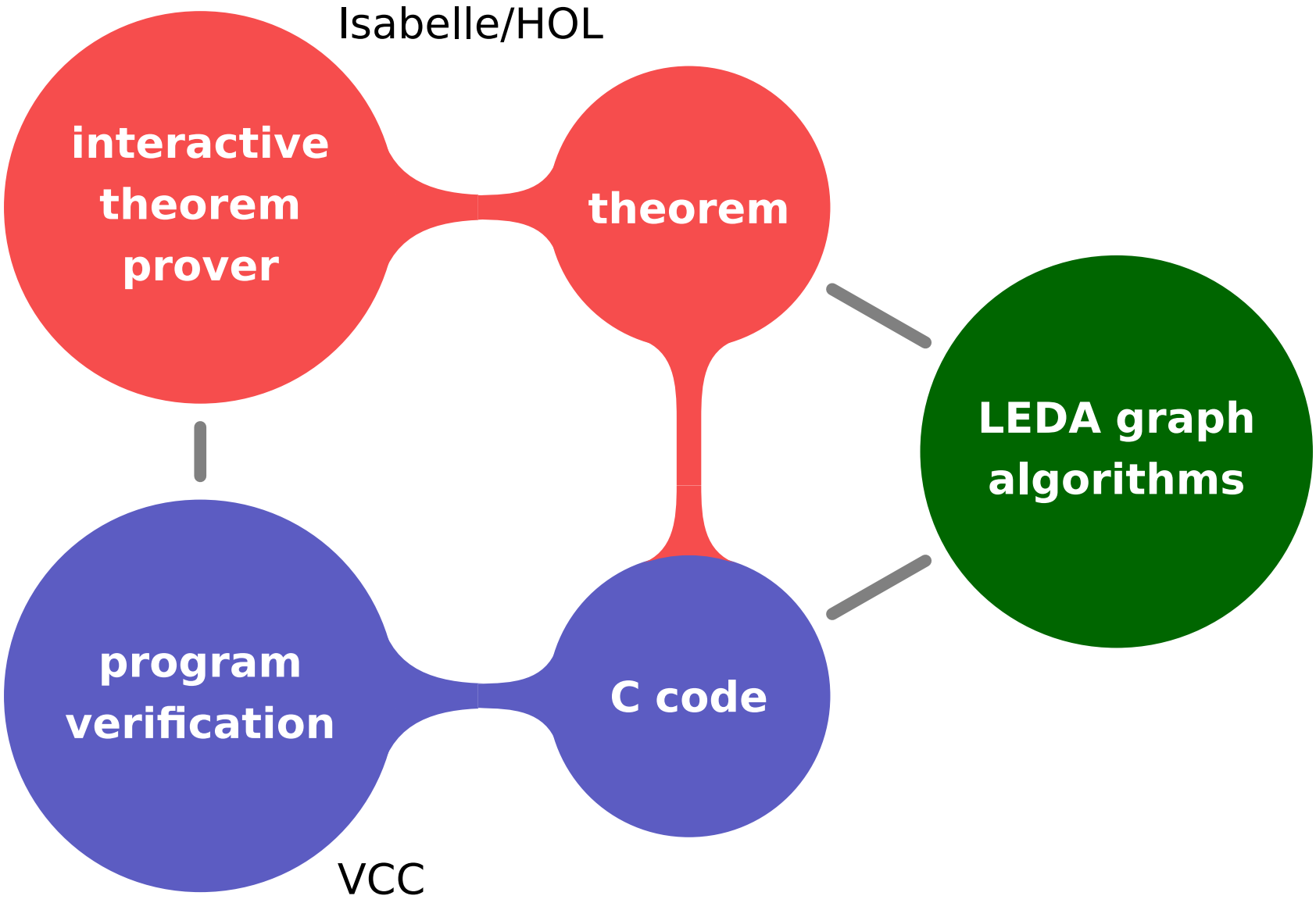
definition abs_is_osc where "abs_is_osc G osc =
(. . . ∧
(∀ e. 0 ≤ e ∧ e < n_edges G ⟶
osc (s (es G e)) = 1 ∨ osc (t (es G e)) = 1 ∨
(osc (t (es G e)) = osc (s (es G e)) ∧ osc (t (es G e)) > 1)))"

theorem
"|M| = $n_1 + \sum_{i \geq 2} \lfloor n_i/2 \rfloor$ ⟶
(∀ M'. is_matching(M') ∧ is_subset(G,M') ⟶ |M'| ≤ |M|)"
⟨proof⟩

Combination of VCC and Isabelle/HOL

- combines the best of both worlds
- low-level code verification with VCC
- high-level mathematical reasoning with Isabelle/HOL
- sound combination
- clean separation of concepts


Isabelle/HOL
interactive theorem prover
theorem
program verification
C code
VCC


Isabelle/HOL
interactive theorem prover
theorem
LEDA graph algorithms
program verification
C code
VCC