Week 9 - README

We extend the synthesis tool created in Week 6 so that it generates boolean combinations of linear invariants. We also test it against benchmark problems that require boolean combinations. This tool only supports `while` loops.

Implementation (for the auto_verify_new.py)

This tool locates the correct Dafny `while` loop based on the week 5 parser (refer to the section on Week 5).

Invariant Synthesis Strategy

The tool attempts synthesis in priority order:

```
def load_invariants(src, method_summary, use_boolean=True):
    # 1. Check for pre-synthesized invariants in summary
    if method_summary.get("synthesized_invariants"):
        return those

    # 2. Try Boolean DNF synthesis (if enabled)
    if use_boolean and has_boolean_synthesizer:
        invs = synthesize_boolean_dnf_invariant_for_loop(...)
        if invs: return invs

    # 3. Fall back to linear synthesis
    if has_linear_synthesizer:
        invs = synthesize_linear_invariants_for_loop(...)
        if invs: return invs

    # 4. No invariants found
    return []
```

Invariant Insertion with DNF Handling

Special formatting for disjunctive invariants:

```
def insert_invariants_into_loop(src, method_summary, invariants):
    for inv in invariants:
        # Wrap disjunctions in parentheses if not already wrapped
        if "||" in inv and not (inv.startswith("(") and inv.endswith(")")):
            inv = f"({inv})"
        inv_block += f"\n{inv_indent}invariant {inv}"
```

Example transformation:

- Input: `x < 5 || y > 10`
- Output: `invariant (x < 5 || y > 10)`

Implementation (for bool_linear_invariants.py)

The start is the same as the week 6 tool (Parsing and loop extraction, building the transition relation, Guard Parsing). After guard parsing, we come to invariant generation.

Invariant generation

- **Linear invariants:**

  Templates of the form $a_1x_1 + \ldots + a_nx_n \leq c$ are generated for small integer ranges.

- **Boolean DNF invariants:**

  Disjunctions of conjunctions of linear constraints.

How it generates DNF invariants

- **Choose a small number kkk** of disjuncts (OR-components).
- For each disjunct:
  - Generate a template:a1×1+…+anxn≤cand/ora1×1+…+anxn=c

    a1×1+…+anxn≤cand/ora1×1+…+anxn=ca_1 x_1 + \dots + a_n x_n \le c \quad\text{and/or}\quad
    a_1 x_1 + \dots + a_n x_n = c

    (multiple such atoms per disjunct)

- Combine them into full DNF templates:(A1∧A2)∨(B1∧B2)...

  (A1∧A2) ∨ (B1∧B2)...(A_{1} \land A_{2}) \;\lor\; (B_{1} \land B_{2}) \ldots
- Ask Z3 to solve for the coefficients so that the *whole DNF*:
  - holds initially
  - is preserved by the loop body
  - is not trivially false

Example

To illustrate the tool, consider this simple Dafny method

Input

```
method DoubleIncrement(n: int) returns (i: int, j: int)
{
  i := 0;
  j := 0;
  while (i < n && j < 2*n)
    invariant (-i <= 0 && -j <= 0 && -i - j <= 0)
  {
    i := i + 1;
    j := j + 2;
  }
}
```

Result

```
{
  "file": "week3\\linearproblem2.dfy",
  "used_boolean": true,
  "invariants": [
    "(-i <= 0 && -j <= 0 && -i - j <= 0)"
  ],
  "out_file": "week9\\linearproblem2_fixed.dfy",
  "ran_dafny": true,
```

```
  "dafny": {
    "command": [
      "dafny",
      "week9\\linearproblem2_fixed.dfy"
    ],
    "returncode": 0,
    "output": "Warning: this way of using the CLI is deprecated. Use 'dafny --help' to see help for the new Dafny CLI format\n\nDafny program verifier finished with 1 verified, 0 errors\nCompiled assembly into linearproblem2_fixed.dll\n"
  }
}
```