bound-t tool

Statically analyses an executable binary program, gives bounds on worst-case execution time (WCET)

Why?

Static analysis of the worst case ensures that real-time deadlines are always met

How?

Loops should be controlled by counters. Loop-bounds should be static, or derived from static subprogram arguments.

Problem is generally unsolvable; made tractable by coding rules and guidance from user-given assertions

Methods used in Bound-T include:
- Loop bounds: Presburger Arithmetic (Omega System)
- Worst case: Integer Linear Programming (lp_solve)

Coded with the GNU Ada Compiler GNAT from ACT.

Source code (C, Ada, Asm)

Libraries Kernel

User assertions on loop bounds, variable values, call counts, etc.

Compiler & linker

Executable binary

Analysis phases:
- Decode instructions
- Analyse control flow
- Analyse subprogram calls
- Analyse loop iterations
- Find worst-case path

Bound-T platforms: Sun Solaris (... Linux, Win NT)
Bound-T targets: ADSP-21020, Intel-8051, ...

Complex DSP instructions are a challenge for static analysis

Architectural pipelines
Zero-overhead loops
Delayed branches

Example:
One instruction expands to four flow-graph steps

Worst-Case Execution Time Analysis for Digital Signal Processors

#define VEC_LEN 100
float sum_vector(float vec[])
{
    float sum = 0.0;
    for (i = 0; i < VEC_LEN; i++)
        sum += vec[i];
    return sum;
}

int find(int *vec, int val)
{
    int low = 0, high, mid;
    high = VEC_LEN - 1;
    while (low <= high) {
        mid = (low + high) / 2;
        if (vec[mid] == val)
            return mid;
        else if (vec[mid] < val)
            low = mid + 1;
        else
            high = mid - 1;
    }
    return -1; /* not found */
}

int C = 0
for (i = 0; i < VEC_LEN; i++)
    C = C + 1;

end_loop

file "find.c"
subprogram "find"
loop repeats <= 7 times;
end_loop

No counter; user asserts bounds:

No counter; user asserts bounds:

file "find.c"
subprogram "find"
loop repeats <= 7 times;
end_loop

Libraries
Kernel

User assertions on loop bounds, variable values, call counts, etc.

Compiler & linker

Executable binary

Analysis phases:
- Decode instructions
- Analyse control flow
- Analyse subprogram calls
- Analyse loop iterations
- Find worst-case path

Bound-T platforms: Sun Solaris (... Linux, Win NT)
Bound-T targets: ADSP-21020, Intel-8051, ...

Complex DSP instructions are a challenge for static analysis

Architectural pipelines
Zero-overhead loops
Delayed branches

Example:
One instruction expands to four flow-graph steps

Worst-Case Execution Time Analysis for Digital Signal Processors

#define VEC_LEN 100
float sum_vector(float vec[])
{
    float sum = 0.0;
    for (i = 0; i < VEC_LEN; i++)
        sum += vec[i];
    return sum;
}

int find(int *vec, int val)
{
    int low = 0, high, mid;
    high = VEC_LEN - 1;
    while (low <= high) {
        mid = (low + high) / 2;
        if (vec[mid] == val)
            return mid;
        else if (vec[mid] < val)
            low = mid + 1;
        else
            high = mid - 1;
    }
    return -1; /* not found */
}

int C = 0
for (i = 0; i < VEC_LEN; i++)
    C = C + 1;

end_loop

file "find.c"
subprogram "find"
loop repeats <= 7 times;
end_loop

No counter; user asserts bounds:

No counter; user asserts bounds:

file "find.c"
subprogram "find"
loop repeats <= 7 times;
end_loop

Libraries
Kernel

User assertions on loop bounds, variable values, call counts, etc.

Compiler & linker

Executable binary

Analysis phases:
- Decode instructions
- Analyse control flow
- Analyse subprogram calls
- Analyse loop iterations
- Find worst-case path

Bound-T platforms: Sun Solaris (... Linux, Win NT)
Bound-T targets: ADSP-21020, Intel-8051, ...

Complex DSP instructions are a challenge for static analysis

Architectural pipelines
Zero-overhead loops
Delayed branches

Example:
One instruction expands to four flow-graph steps

Worst-Case Execution Time Analysis for Digital Signal Processors

#define VEC_LEN 100
float sum_vector(float vec[])
{
    float sum = 0.0;
    for (i = 0; i < VEC_LEN; i++)
        sum += vec[i];
    return sum;
}

int find(int *vec, int val)
{
    int low = 0, high, mid;
    high = VEC_LEN - 1;
    while (low <= high) {
        mid = (low + high) / 2;
        if (vec[mid] == val)
            return mid;
        else if (vec[mid] < val)
            low = mid + 1;
        else
            high = mid - 1;
    }
    return -1; /* not found */
}

int C = 0
for (i = 0; i < VEC_LEN; i++)
    C = C + 1;

end_loop

file "find.c"
subprogram "find"
loop repeats <= 7 times;
end_loop

No counter; user asserts bounds:

No counter; user asserts bounds:

file "find.c"
subprogram "find"
loop repeats <= 7 times;
end_loop

Libraries
Kernel

User assertions on loop bounds, variable values, call counts, etc.

Compiler & linker

Executable binary

Analysis phases:
- Decode instructions
- Analyse control flow
- Analyse subprogram calls
- Analyse loop iterations
- Find worst-case path

Bound-T platforms: Sun Solaris (... Linux, Win NT)
Bound-T targets: ADSP-21020, Intel-8051, ...

Complex DSP instructions are a challenge for static analysis

Architectural pipelines
Zero-overhead loops
Delayed branches

Example:
One instruction expands to four flow-graph steps

Worst-Case Execution Time Analysis for Digital Signal Processors

#define VEC_LEN 100
float sum_vector(float vec[])
{
    float sum = 0.0;
    for (i = 0; i < VEC_LEN; i++)
        sum += vec[i];
    return sum;
}

int find(int *vec, int val)
{
    int low = 0, high, mid;
    high = VEC_LEN - 1;
    while (low <= high) {
        mid = (low + high) / 2;
        if (vec[mid] == val)
            return mid;
        else if (vec[mid] < val)
            low = mid + 1;
        else
            high = mid - 1;
    }
    return -1; /* not found */
}

int C = 0
for (i = 0; i < VEC_LEN; i++)
    C = C + 1;

end_loop

file "find.c"
subprogram "find"
loop repeats <= 7 times;
end_loop

No counter; user asserts bounds:

No counter; user asserts bounds:

file "find.c"
subprogram "find"
loop repeats <= 7 times;
end_loop

Libraries
Kernel

User assertions on loop bounds, variable values, call counts, etc.

Compiler & linker

Executable binary

Analysis phases:
- Decode instructions
- Analyse control flow
- Analyse subprogram calls
- Analyse loop iterations
- Find worst-case path

Bound-T platforms: Sun Solaris (... Linux, Win NT)
Bound-T targets: ADSP-21020, Intel-8051, ...

Complex DSP instructions are a challenge for static analysis

Architectural pipelines
Zero-overhead loops
Delayed branches

Example:
One instruction expands to four flow-graph steps

Worst-Case Execution Time Analysis for Digital Signal Processors

#define VEC_LEN 100
float sum_vector(float vec[])
{
    float sum = 0.0;
    for (i = 0; i < VEC_LEN; i++)
        sum += vec[i];
    return sum;
}

int find(int *vec, int val)
{
    int low = 0, high, mid;
    high = VEC_LEN - 1;
    while (low <= high) {
        mid = (low + high) / 2;
        if (vec[mid] == val)
            return mid;
        else if (vec[mid] < val)
            low = mid + 1;
        else
            high = mid - 1;
    }
    return -1; /* not found */
}

int C = 0
for (i = 0; i < VEC_LEN; i++)
    C = C + 1;

end_loop

file "find.c"
subprogram "find"
loop repeats <= 7 times;
end_loop

No counter; user asserts bounds:

No counter; user asserts bounds:

file "find.c"
subprogram "find"
loop repeats <= 7 times;
end_loop

Libraries
Kernel

User assertions on loop bounds, variable values, call counts, etc.

Compiler & linker

Executable binary

Analysis phases:
- Decode instructions
- Analyse control flow
- Analyse subprogram calls
- Analyse loop iterations
- Find worst-case path

Bound-T platforms: Sun Solaris (... Linux, Win NT)
Bound-T targets: ADSP-21020, Intel-8051, ...

Complex DSP instructions are a challenge for static analysis

Architectural pipelines
Zero-overhead loops
Delayed branches

Example:
One instruction expands to four flow-graph steps

Worst-Case Execution Time Analysis for Digital Signal Processors

#define VEC_LEN 100
float sum_vector(float vec[])
{
    float sum = 0.0;
    for (i = 0; i < VEC_LEN; i++)
        sum += vec[i];
    return sum;
}

int find(int *vec, int val)
{
    int low = 0, high, mid;
    high = VEC_LEN - 1;
    while (low <= high) {
        mid = (low + high) / 2;
        if (vec[mid] == val)
            return mid;
        else if (vec[mid] < val)
            low = mid + 1;
        else
            high = mid - 1;
    }
    return -1; /* not found */
}

int C = 0
for (i = 0; i < VEC_LEN; i++)
    C = C + 1;

end_loop

file "find.c"
subprogram "find"
loop repeats <= 7 times;
end_loop

No counter; user asserts bounds:

No counter; user asserts bounds:

file "find.c"
subprogram "find"
loop repeats <= 7 times;
end_loop

Libraries
Kernel

User assertions on loop bounds, variable values, call counts, etc.

Compiler & linker

Executable binary

Analysis phases:
- Decode instructions
- Analyse control flow
- Analyse subprogram calls
- Analyse loop iterations
- Find worst-case path

Bound-T platforms: Sun Solaris (... Linux, Win NT)
Bound-T targets: ADSP-21020, Intel-8051, ...

Complex DSP instructions are a challenge for static analysis

Architectural pipelines
Zero-overhead loops
Delayed branches

Example:
One instruction expands to four flow-graph steps