Chopping Concurrent Programs

Dennis Giffhorn

Universität Karlsruhe (TH), Germany

Chopping

May statement s influence statement t?

- $chop(s,t)$ contains all statements which may convey effects from s to t

Example: $chop(1,5)$

```
1  int a = input();
2  int b = input();
3  b = b * a;
4  if (b < 0)
5     print(b);
```

- Intuitively: $chop(s,t) = forward\ slice(s) \cap backward\ slice(t)$

Chopping

May statement s influence statement t?

- $chop(s,t)$ contains all statements which may convey effects from s to t

Example: $chop(1,5)$

```
1  int a = input();
2  int b = input();
3  b = b * a;
4  if (b < 0)
5     print(b);
```

- Intuitively: $chop(s,t) = forward\ slice(s) \cap backward\ slice(t)$

Chopping

- Main application: preprocessing step
- The more precise the chop,
  - the more precise is the main analysis
  - the faster is the main analysis

Chopping

- Main application: preprocessing step
- The more precise the chop,
  - the more precise is the main analysis
  - the faster is the main analysis
- When we started our work
  - Precise chopping algorithms for seq. programs
  - No algorithm for conc. programs at all

Context-sensitive chopping

- Distinguish different calls of the same procedure

Example: $chop(2,3)$

```
1  void main()                 4  int foo()
2    int m = foo();            5    return 1;
3    int n = foo();
```

Context-sensitive chopping

- Distinguish different calls of the same procedure

Example: *forward slice*(2)

```
1  void main()                 4  int foo()
2    int m = foo();            5    return 1;
3    int n = foo();
```

Context-sensitive chopping

- Distinguish different calls of the same procedure

Example: *backward slice*(3)

```
1  void main()                4  int foo()
2    int m = foo();           5    return 1;
3    int n = foo();
```

Context-sensitive chopping

- Distinguish different calls of the same procedure

Example: $chop(2,3)$

```
1  void main()
2    int m = foo();
3    int n = foo();
```

```
4  int foo()
5    return 1;
```

Context-sensitive chopping

- Distinguish different calls of the same procedure

Example: $chop(2,3)$

```
1  void main()                  4  int foo()
2    int m = foo();             5    return 1;
3    int n = foo();
```

- Solved for sequential programs [*Reps and Rosay, FSE 1995*]
  ⇒ Extension to concurrent programs
- Resulting algorithm has same asymptotic running time:
  $O(|Edges| * MaxParams)$

Time-sensitive chopping

- Distinguish different interleavings between threads

Example: $chop(4,3)$

```
1  int x,y;
2  void thread1()                5  void thread2()
3    int a = y;                  6    int p = x;
4    x = a;                      7    y = p;
```

Time-sensitive chopping

- Distinguish different interleavings between threads

Example: *forward slice*(4)

```
1  int x,y;
2  void thread1()                 5  void thread2()
3    int a = y;                   6    int p = x;
4    x = a;                       7    y = p;
```

Time-sensitive chopping

- Distinguish different interleavings between threads

Example: *backward slice*(3)

```
1  int x,y;
2  void thread1()                 5  void thread2()
3    int a = y;                   6    int p = x;
4    x = a;                       7    y = p;
```

Time-sensitive chopping

- Distinguish different interleavings between threads

Example: $chop(4,3)$

```
1  int x,y;
2  void thread1()                5  void thread2()
3    int a = y;                  6    int p = x;
4    x = a;                      7    y = p;
```

Time-sensitive chopping

- Distinguish different interleavings between threads

Example: $chop(4,3)$

```
1  int x,y;
2  void thread1()                 5  void thread2()
3    int a = y;                   6    int p = x;
4    x = a;                       7    y = p;
```

- Extension of time-sensitive slicing ([*Krinke, FSE '03*], [*Nanda and Ramesh, TOPLAS '06*]) to time-sensitive chopping
- Same asymptotic running time as time-sensitive slicing: $\mathcal{O}(|Nodes|^{(maximal\ call\ depth)^{|threads|}})$

Evaluation – Average number of nodes per chop

Name	(nodes, edges, threads)	I	CS	TS
Logger	(9576, 50800, 2)	985	967	796
Maza	(10590, 60021, 2)	1543	1153	798
Barcode	(11025, 67849, 2)	711	541	469
Guitar	(13459, 89724, 2)	1734	1606	1476
J2MESafe	(15666, 127922, 2)	4027	3611	2423
Podcast	(23399, 191849, 3)	10423	10400	2310

- Context-sensitive chops up to 25% smaller, on average 10%
- Time-sensitive chops up to 80% smaller, on average 35%

Evaluation – Average number of nodes per chop

Name	(nodes, edges, threads)	I	CS	TS
Logger	(9576, 50800, 2)	985	967	796
Maza	(10590, 60021, 2)	1543	1153	798
Barcode	(11025, 67849, 2)	711	541	469
Guitar	(13459, 89724, 2)	1734	1606	1476
J2MESafe	(15666, 127922, 2)	4027	3611	2423
Podcast	(23399, 191849, 3)	10423	10400	2310

- Context-sensitive chops up to 25% smaller, on average 10%
- Time-sensitive chops up to 80% smaller, on average 35%

Evaluation – Average number of nodes per chop

Name	(nodes, edges, threads)	I	CS	TS
Logger	(9576, 50800, 2)	985	967	796
Maza	(10590, 60021, 2)	1543	1153	798
Barcode	(11025, 67849, 2)	711	541	469
Guitar	(13459, 89724, 2)	1734	1606	1476
J2MESafe	(15666, 127922, 2)	4027	3611	2423
Podcast	(23399, 191849, 3)	10423	10400	2310

- Context-sensitive chops up to 25% smaller, on average 10%
- Time-sensitive chops up to 80% smaller, on average 35%

Evaluation – Average time per chop in msec.

Name	(nodes, edges, threads)	I	CS	TS
Logger	(9576, 50800, 2)	14.5	31.6	77.9
Maza	(10590, 60021, 2)	25.9	53.6	2568.0
Barcode	(11025, 67849, 2)	14.8	16.6	88.2
Guitar	(13459, 89724, 2)	37.7	59.9	551.2
J2MESafe	(15666, 127922, 2)	60.4	180.0	7637.8
Podcast	(23399, 191849, 3)	56.1	283.7	9039.2

- CS chops up to 5 times slower, on average 3 times slower
- TS chops up to 161 times slower, on average 95 times slower

Future work ?

There is even more precision to gain (e.g. synchronization)

- Costs would further explode
- Algorithms are difficult to implement by now

⇒ People tend to use intersection-based chopping

- How can we benefit from this huge increase of precision in practice?