CnC Is Deterministic

David M. Peixotto[1] Jens. Palsberg[2]

[1]Rice University

[2]University of California, Los Angeles

CnC Workshop, 2010

The Whole Talk in one Slide

Contributions

Featherweight CnC

Syntax & Semantics

Proof* of Determinism

For Featherweight CnC

The Whole Talk in one Slide

Contributions

Featherweight CnC

Syntax & Semantics

Proof* of Determinism

For Featherweight CnC

Ongoing work

Problem with the Proof

Featherweight CnC is too powerful

Outline

Defining the Grammar of Featherweight CnC

- Coming up with the grammar
  - Reduce CnC to its minimal constructs
  - The smaller the better – proof must deal with each syntactic construct
  - Many revisions – at least 11 attempts
- The subset of CnC
  - Merge the coordination and computation languages
  - Get rid of named item collections
  - Get rid of tag collections
  - All tags and values are integers

Defining the Grammar of Featherweight CnC

- Coming up with the grammar
  - Reduce CnC to its minimal constructs
  - The smaller the better – proof must deal with each syntactic construct
  - Many revisions – at least 11 attempts
- The subset of CnC
  - Merge the coordination and computation languages
  - Get rid of named item collections
  - Get rid of tag collections
  - All tags and values are integers

Defining the Grammar of Featherweight CnC

- Coming up with the grammar
  - Reduce CnC to its minimal constructs
  - The smaller the better – proof must deal with each syntactic construct
  - Many revisions – at least 11 attempts
- The subset of CnC
  - Merge the coordination and computation languages
  - Get rid of named item collections
  - Get rid of tag collections
  - All tags and values are integers

Lesson 1

Take the time to get the syntax right

Lesson 2

Keep it simple stupid (KISS)

Syntax of Featherweight CnC

$$
\begin{array}{lrll}
\textit{Program}: & p & ::= & f_i(\textit{int } a)\{d_i\ s_i\}, i \in 1..m \\
\textit{Declaration}: & d & ::= & n = \textit{data.get}(e); d \\
 & & | & \epsilon \\
\textit{Statement}: & s & ::= & \textit{skip} \\
 & & | & \textit{if } (e > 0)\ s_1\ \textit{else}\ s_2 \\
 & & | & \textit{data.put}(e_1, e_2); s \\
 & & | & \textit{prescribe } f_i(e); s \\
\textit{Expression}: & e & ::= & c \qquad \text{(integer constant)} \\
 & & | & n \qquad \text{(local name)} \\
 & & | & a \qquad \text{(formal parameter name)} \\
 & & | & e_1 + e_2
\end{array}
$$

The Featherweight CnC Abstract Machine

The Featherweight CnC machine has two components (A, T)

Data (A)

- All item collections are merged into a single collection A
- A maps tags(integers) to data(integers)

Computation (T)

- Tree of concurrently executing steps
  $Tree: \quad T \quad ::= \quad T \parallel T \quad | \quad (d\ s)$
- We assert $\parallel$ is associative and commutative

The Featherweight CnC Abstract Machine

State

$\sigma = (A, T) \mid \text{error}$

Start State

(A_0, s)
A_0 maps all tags to $\perp$
s is a set of initial statements

Final State

success	$(A, skip)$
failure	error
deadlock	(A, T) where every occurrence in T of $(d\ s)$ has that property that d is of the form $n = data.get(e); d'$ and $A[[e]] = \perp$

Reduction Rules for Featherweight CnC (1/2)

Error Rule

$$\frac{(A, T_1) \rightarrow \text{error}}{(A, T_1 \parallel T_2) \rightarrow \text{error}} \quad (1)$$

Interleaving Rule

$$\frac{(A, T_1) \rightarrow (A', T_1')}{(A, T_1 \parallel T_2) \rightarrow (A', T_1' \parallel T_2)} \quad (2)$$

Reduction Rules for Featherweight CnC (2/2)

Prescribe

$$(A, \mathit{prescribe}\ f_i(e); s) \rightarrow (A, ((d_i\ s_i)[a := [\![e]\!]])\ \|\ s) \quad (\text{the body of } f_i \text{ is } (d_i\ s_i)) \tag{3}$$

Put Success

$$(A, \mathit{data.put}(e_1, e_2); s) \rightarrow (A[[\![e_1]\!] := [\![e_2]\!]], s) \quad (\text{if } A[[\![e_1]\!]] = \bot) \tag{4}$$

Put Failure

$$(A, \mathit{data.put}(e_1, e_2); s) \rightarrow \text{error} \quad (\text{if } A[[\![e_1]\!]] \neq \bot) \tag{5}$$

Get

$$(A, n = \mathit{data.get}(e); d\ s) \rightarrow (A, (d\ s)[n := A[[\![e]\!]]]) \quad (\text{if } A[[\![e]\!]] \neq \bot) \tag{6}$$

Reduction Rules for Featherweight CnC (2/2)

Prescribe

$$(A, \mathit{prescribe}\ f_i(e); s) \rightarrow (A, ((d_i\ s_i)[a := [\![e]\!]])\ \|\ s) \quad (\text{the body of } f_i \text{ is } (d_i\ s_i)) \tag{3}$$

Put Success

$$(A, \mathit{data.put}(e_1, e_2); s) \rightarrow (A[[\![e_1]\!] := [\![e_2]\!]], s) \quad (\text{if } A[[\![e_1]\!]] = \bot) \tag{4}$$

Put Failure

$$(A, \mathit{data.put}(e_1, e_2); s) \rightarrow \text{error} \quad (\text{if } A[[\![e_1]\!]] \neq \bot) \tag{5}$$

Get

$$(A, n = \mathit{data.get}(e); d\ s) \rightarrow (A, (d\ s)[n := A[[\![e]\!]]]) \quad (\text{if } A[[\![e]\!]] \neq \bot) \tag{6}$$

Reduction Rules for Featherweight CnC (2/2)

Prescribe

$$(A, \mathit{prescribe}\ f_i(e); s) \rightarrow (A, ((d_i\ s_i)[a := [\![e]\!]])\ \|\ s) \quad (\text{the body of } f_i \text{ is } (d_i\ s_i)) \tag{3}$$

Put Success

$$(A, \mathit{data.put}(e_1, e_2); s) \rightarrow (A[[\![e_1]\!] := [\![e_2]\!]], s) \quad (\text{if } A[[\![e_1]\!]] = \bot) \tag{4}$$

Put Failure

$$(A, \mathit{data.put}(e_1, e_2); s) \rightarrow \text{error} \quad (\text{if } A[[\![e_1]\!]] \neq \bot) \tag{5}$$

Get

$$(A, n = \mathit{data.get}(e); d\ s) \rightarrow (A, (d\ s)[n := A[[\![e]\!]]]) \quad (\text{if } A[[\![e]\!]] \neq \bot) \tag{6}$$

Reduction Rules for Featherweight CnC (2/2)

Prescribe

$$(A, \mathit{prescribe}\ f_i(e); s) \rightarrow (A, ((d_i\ s_i)[a := [\![e]\!]])\ \|\ s) \quad (\text{the body of } f_i \text{ is } (d_i\ s_i)) \tag{3}$$

Put Success

$$(A, \mathit{data.put}(e_1, e_2); s) \rightarrow (A[[\![e_1]\!] := [\![e_2]\!]], s) \quad (\text{if } A[[\![e_1]\!]] = \bot) \tag{4}$$

Put Failure

$$(A, \mathit{data.put}(e_1, e_2); s) \rightarrow \text{error} \quad (\text{if } A[[\![e_1]\!]] \neq \bot) \tag{5}$$

Get

$$(A, n = \mathit{data.get}(e); d\ s) \rightarrow (A, (d\ s)[n := A[[\![e]\!]]]) \quad (\text{if } A[[\![e]\!]] \neq \bot) \tag{6}$$

Properties of Featherweight CnC

Captures many of the properties of Full CnC

- Single assignment property (side condition on rule 4,5)
- Control dependence (rule 3)
- Data dependence (rule 6)
- Gets before Puts (grammar)

Some “controversial” properties

- No memoization of steps
- No equality comparison for multiple puts

Outline

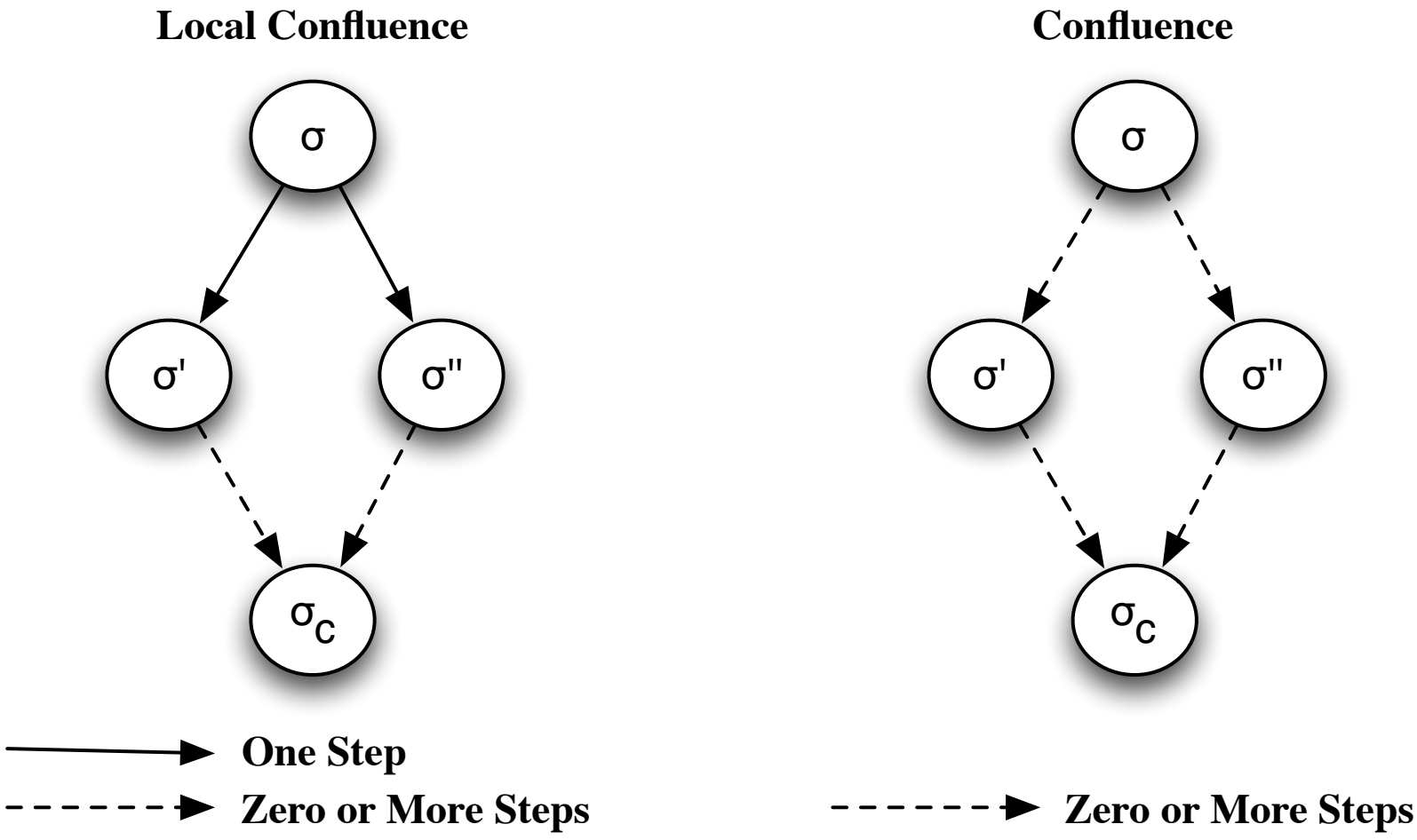
Statement of Determinism for Featherweight CnC

Theorem (Determinism)

If $\sigma \rightarrow^* \sigma'$ *and* $\sigma \rightarrow^* \sigma''$*, and* σ', σ'' *are both final states, then* $\sigma' = \sigma''$*.*

We use confluence to prove this theorem

Definition of Confluence



From Confluence to Determinism

1. Prove local confluence by case enumeration (single-assignment gives montonicty of σ)
2. Prove confluence = Local confluence + strongly normalizing (Newton's Lemma)
3. Determinism follows easily from confluence

Theorem (Determinism)

If $\sigma \to^* \sigma'$ and $\sigma \to^* \sigma''$, and σ', σ'' are both final states, then $\sigma' = \sigma''$.

Proof of Determinism.

Confluence gives that there exists σ_c such that $\sigma' \to^* \sigma_c$ and $\sigma'' \to^* \sigma_c$. Given that neither σ' or σ'' have any outgoing transitions, we must have $\sigma' = \sigma_c$ and $\sigma'' = \sigma_c$, hence $\sigma' = \sigma''$. □

A slight problem

Problem

Featherweight CnC is not strongly normalizing

```
f_1(a){prescribe f_1(a+1)}
```

Solution 1

Remove function calls

Solution 2

Rework the proof of confluence with a different technique

Solution 3

Restrict Featherweight CnC to terminating programs

Summary

- Definition of Featherweight CnC
- Interleaving semantics for Featherweight CnC
- Proof (in progress) that Featherweight CnC is deterministic

Backup Slides

Why a Formal Proof

Why a Formal Proof

- Credibility
- Insight
- Connection with Programming Languages (PL) community

Featherweight CnC to Full CnC

Named item collections

Allow `put` and `get` operations on collections other than `data`.

Named tag collections

Introduce tag collections to the grammar. A put on the tag collection calls `prescribe` on each function the collection prescribes.

Separate computation language

Current language is Turing complete
Other step languages can violate single-assignment