

pCG: A CG Programming Language and Process Test-bed

*David Benn
Dan Corbett*

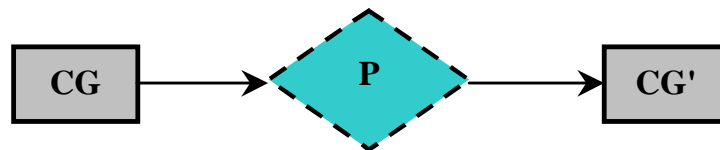
*University of South Australia
School of Computer and Information Science*

ICCS 2001



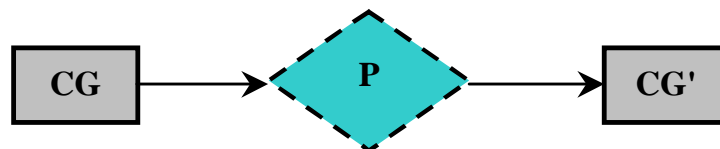
Plan

- **Origin of pCG**
- **Processes**
- **More on pCG**
- **Sisyphus-1 as Illustration**
- **Outcomes**
- **Some Future Directions**



Origin of pCG

- UniSA Computer & Information Science coursework Masters minor thesis under Dan Corbett.
- Test-bed for an implementation of Guy Mineau's Process mechanism (ICCS'98).
- One *possible* environment with which to deliver combined logic and imperative programming.
- Meaning?
 - programming with CGs
 - application of a process p to CG



pCG: A Taste

Example 1

```
option CGIParser = "cgp.translators.CGIParser";
option CGIgen = "cgp.translators.CGIGenerator";
if _ARGS.length == 1 then
  f = file (_ARGS[1]);
  graphs = f.readGraphStream();
  f.close();
  foreach g in graphs do
    println g.nocomments();
  end
end
```

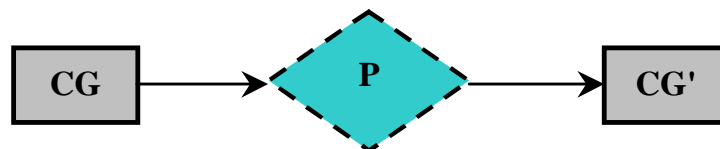
Example 2

```
g1 = `(Chrc [Person:'Fred'] [Age:+80])`;
g2 = `(Chrc [Person:'Fred'] [Gender:"Male"])`;
g3 = g1.join(g2);
println g3;
```



Processes

- **Considerable work to date on dynamic and executable CGs, eg actors, demons.**
- **Sowa (2000) classifies kinds of processes: discrete, continuous, and variants.**
- **Mineau's (1998) process mechanism: one approach to representing discrete processes.**



Processes: key notions

- **state transitions;**
- **knowledge-based *precondition* matching: conjunction of CGs;**
- ***post-conditions* modify Knowledge Base (KB) by graph assertion and retraction; may use matched information;**
- ***process* $p(\text{in } g1, \text{out } g2, \dots)$ is**
$$\{ r_i = \langle pre_i, post_i \rangle, \forall_i \in [1, n] \}$$
- **forward chaining.**



Processes

- **Generalisation of Delugach's (1991) *demons*, and Sowa's *actors or dataflow graphs*.**

process > demon > actor

Formalism	Parameters
Processes	Graphs
Demons	Concepts
Actors	Referents

- **Process mechanism also *uses* actors in practice when referent computation required.**



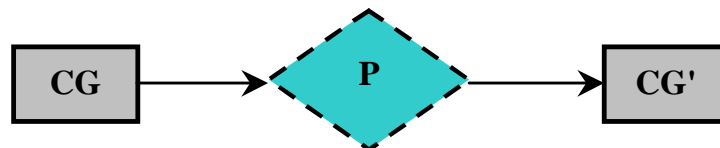
pCG in a Nutshell

- Permits process *definition* and *invocation*;
- Provides capabilities for working with conceptual graphs, processes, actors, and functions as *first class values*;
- *General-purpose* (albeit experimental) programming language; permits imperative, object-based, functional, and declarative styles.
- Available under Gnu Public License.



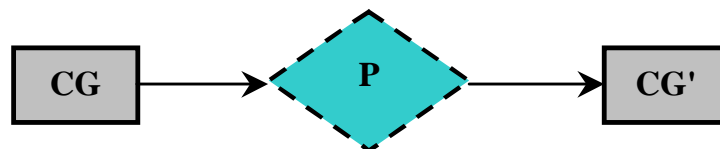
Why pCG?

- **Paper-based exercise \Rightarrow process programs.**
- **Design goals:**
 - Making those entities of primary interest to the developer first class. Could have implemented an API, but programs would not be concise.
 - Rapid development: interpreted, complex built-in types, GC.
 - Portability: Java, ANTLR: Java-based compiler development language.
 - Easy extensibility: attributes, operations, types can easily be added via Java code.
- **Emphasis upon CGs and processes.**
- **Simple language; focus upon new semantics.**
- **Possible to change source language: interpreter & library (with Notio's help) do the hard work.**



Sisyphus-I

- **Constraint satisfaction problem. A means by which to test knowledge acquisition and reasoning strategies.**
- **Research group of people with varied needs must be allocated rooms.**
- **Constraints detailed in problem description impose certain order on any solution, e.g.**
 - **allocate head of group to large room;**
 - **smokers & non-smokers can't share rooms.**

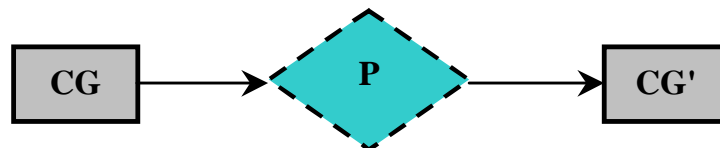


Sisyphus-I

- Perl script extracts most information from HTML, yielding simple CGs, e.g.

```
(Chrc [Person:'Michael T.'] [Role:'Researcher'])  
(Member [Person:'Michael T.'] [Project:'BABYLON Product'])  
(Chrc [Person:'Michael T.'] [Smoker:'No'])  
(Chrc [Person:'Michael T.'] [Hacker:'Yes'])  
(Coworker [Person:'Michael T.'] [Person:'Hans W.'])
```

- Graphs for different people separated by blank line.



Sisyphus-I

- pCG code reads all such CGs from file, yielding 1 graph per person via *Join*, e.g.

```
[Person:*a'Michael T. ']  
[Role:*b'Researcher ']  
[Project:*c'BABYLON Product ']  
[Smoker:*d'No ']  
[Hacker:*e'Yes ']  
[Person:*f'Hans W. ']
```

```
(Chrc?a?b)  
(Member?a?c)  
(Chrc?a?d)  
(Chrc?a?e)  
(Coworker?a?f)
```

```
[Person:'Michael T. ' -  
  (Chrc) -> [Role: 'Researcher ']  
  (Member) -> [Project: 'BABYLON Product ']  
  (Chrc) -> [Smoker: 'No ']  
  (Chrc) -> [Hacker: 'Yes ']  
  (Coworker) -> [Person: 'Hans W. ']
```

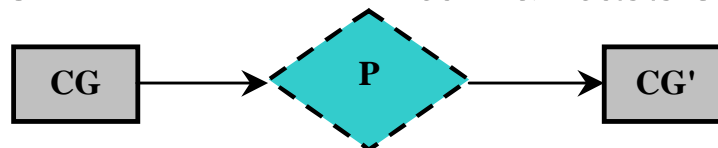


Sisyphus-I

- **Person and room CGs asserted:**

```
rooms = { `(Chrc [LargeRoom*a:'C5-117'] [Location:'Central'])  
          (Vacancy ?a [Integer:+2])`,  
          ...  
          `(Chrc [LargeRoom*a:'C5-120'] [Location:'NonCentral'])  
          (Vacancy ?a [Integer:+2])`,  
          ...  
          `(Chrc [SingleRoom*a:'C5-116'] [Location:'Central'])  
          (Vacancy ?a [Integer:+1])` };  
  
foreach room in rooms do  
  assert room;  
end
```

- **Other information re: YQT members gleaned from HTML and asserted.**



Sisyphus-I

```
rule allocateFirstSecretary
  pre
    `(Chrc [Person:*name] [Role:'Secretary'])`;

    `(Chrc [LargeRoom*a:*roomLabel] [Location:'Central'])
      (Vacancy ?a [Integer:+2])`;
  end

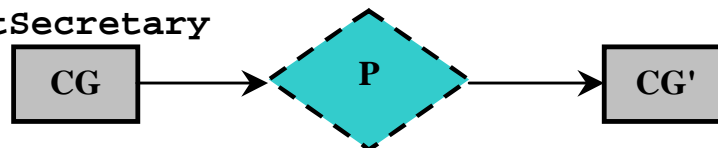
  post
    `[PROPOSITION:(Occupant [Person:*name] [Room:*roomLabel'])]`;
    option export;

    mkErasureGraph(_MATCHES[1]); // erase person; syntax change required!

    `[ERASURE:(Chrc [LargeRoom*a:*roomLabel] [Location:'Central'])
      (Vacancy ?a [Integer:+2])]`;

    `[PROPOSITION:(Chrc [LargeRoom*a:*roomLabel] [Location:'Central'])
      (Vacancy ?a [Integer:+1])]`;
  end
end // allocateFirstSecretary
```

Benn & Corbett



Sisyphus-I

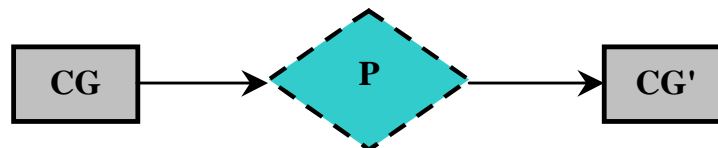
```
rule allocateFirstSecretary
  pre
    action
      println "Need to allocate room for first secretary?";
    end

    `(Chrc [Person:'*name'] [Role:'Secretary'])`; ...
  end

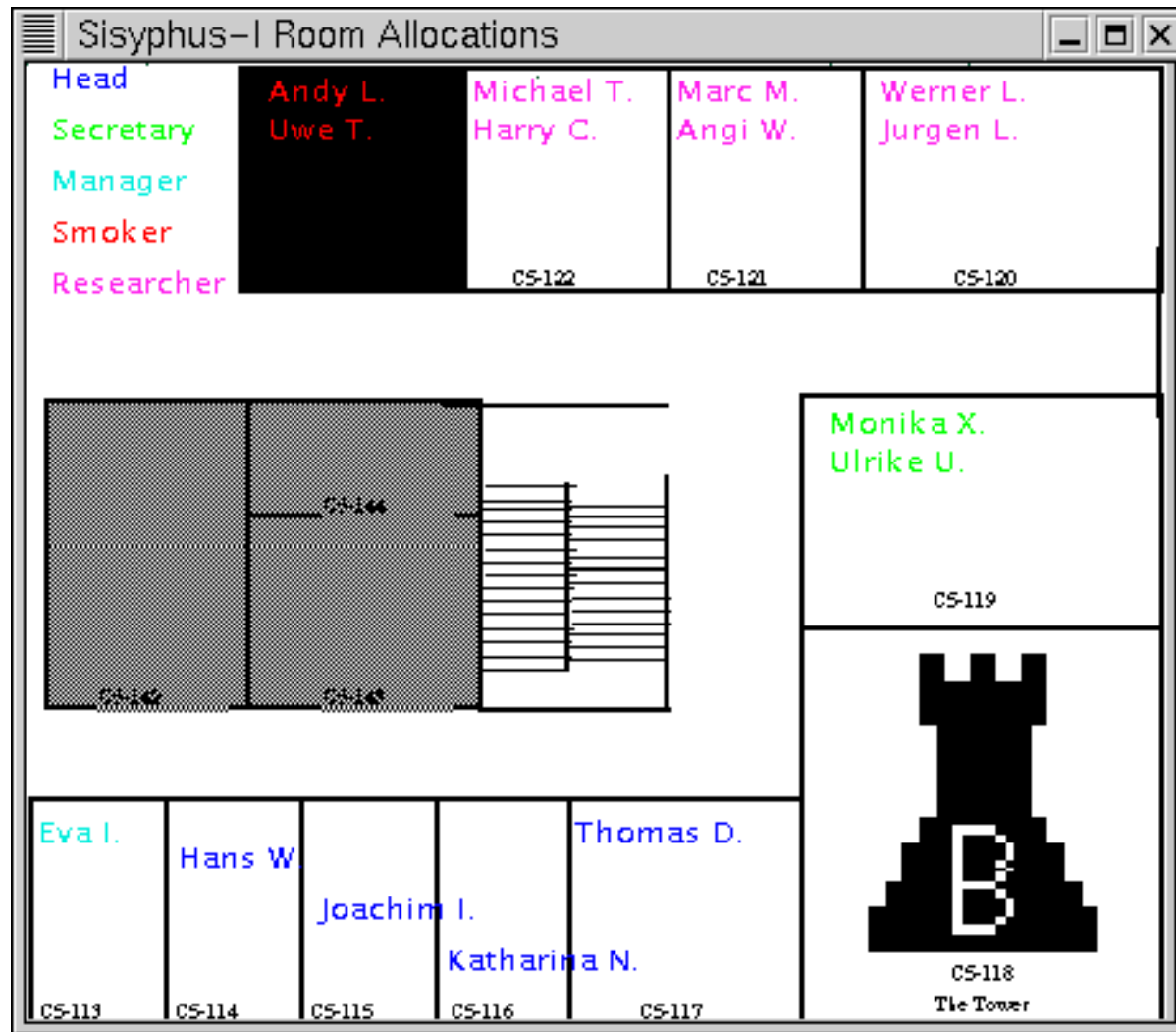
  post
    action
      label = getRoomLabel();
      name = getPersonName();
      showAllocation("First secretary", name, label);
      plotName(name, label, 1, secretaryColor);
    end

    `[PROPOSITION:(Occupant [Person:'*name'] [Room:'*roomLabel'])]`...
  end
end
```

Benn & Corbett



Sisyphus-I



Sisyphus-I

Script started on Mon Nov 27 09:32:13 2000

```
[david@twist ~/cgp]$ pCG examples/sisyphus-1/scg-1.cgp
```

Asserting YQT member graphs.

Asserting more information about certain YQT members.

Need to allocate room for the head of YQT?

--> *Head of YQT 'Thomas D.'* allocated to C5-117

Need to allocate room for the head of YQT?

Need to allocate room for second secretary?

Need to allocate room for first secretary?

--> *First secretary 'Monika X.'* allocated to C5-119

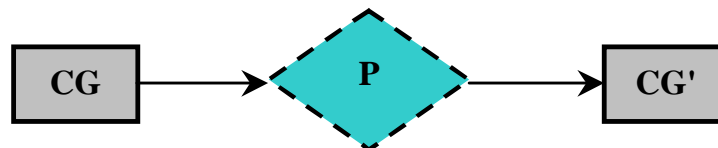
Need to allocate room for the head of YQT?

Need to allocate room for second secretary?

--> *Second secretary 'Ulrike U.'* allocated to C5-119

.

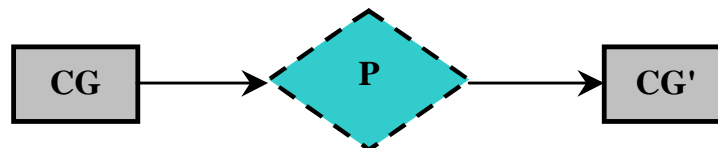
.



Sisyphus-I

Room allocation graphs

```
[Person"Thomas D."]-1->(Occupant)-2->[Room"C5-117"]
[Person"Monika X."]-1->(Occupant)-2->[Room"C5-119"]
[Person"Ulrike U."]-1->(Occupant)-2->[Room"C5-119"]
[Person"Eva I."]-1->(Occupant)-2->[Room"C5-113"]
[Person"Hans W."]-1->(Occupant)-2->[Room"C5-114"]
[Person"Joachim I."]-1->(Occupant)-2->[Room"C5-115"]
[Person"Katharina N."]-1->(Occupant)-2->[Room"C5-116"]
[Person"Werner L."]-1->(Occupant)-2->[Room"C5-120"]
[Person"Jurgen L."]-1->(Occupant)-2->[Room"C5-120"]
[Person"Marc M."]-1->(Occupant)-2->[Room"C5-121"]
[Person"Angi W."]-1->(Occupant)-2->[Room"C5-121"]
[Person"Michael T."]-1->(Occupant)-2->[Room"C5-122"]
[Person"Harry C."]-1->(Occupant)-2->[Room"C5-122"]
[Person"Andy L."]-1->(Occupant)-2->[Room"C5-123"]
[Person"Uwe T."]-1->(Occupant)-2->[Room"C5-123"]
```



Sisyphus-I

```
concept Person;
```

```
...
```

```
concept Room > SingleRoom;
```

```
concept Room > LargeRoom;
```

```
relation Attr;
```

```
relation Chrc;
```

```
relation Member;
```

```
relation Coworker;
```

```
relation Vacancy;
```

```
relation Occupant;
```

```
rule allocateRemainingResearcher
```

```
pre
```

```
  `(Chrc [Person: '*name'] [Role: 'Researcher'])`;
```

```
  `(Chrc [Room: '*roomLabel'] [Location: '*somewhere'])`;
```

```
end
```

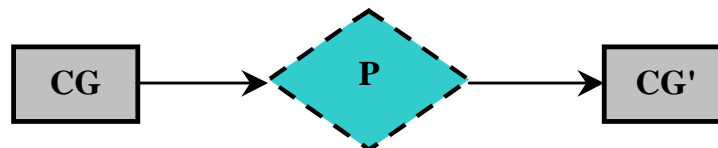
```
post
```

```
  `[PROPOSITION: (Occupant [Person: '*name'] [Room: '*roomLabel'])]`;
```

```
  option export;
```

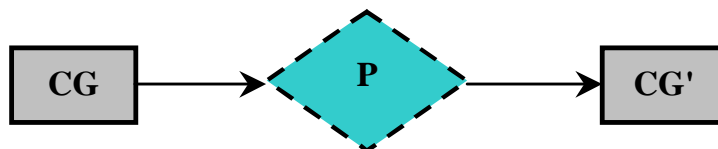
```
end
```

```
end // allocateRemainingResearcher
```



Outcomes

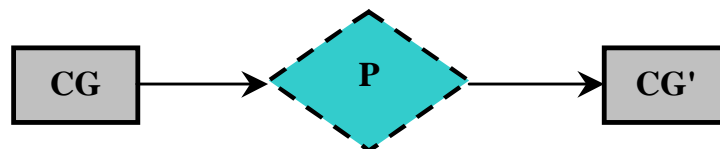
- ***Refinement*** of Mineau's process formalism in the course of developing pCG and applying it, eg negation vs retraction, process statement minutiae, actors in processes.
- ***Application*** of pCG to examples other than Mineau's 1998 iterative factorial test process, eg the *Sisyphus-I* room allocation problem (Linster, 1999).



Outcomes

- **Uniform representation of actor and process invocation via *standard actor node* as per notation of the proposed ANSI CG Standard instead of non-standard (double-diamond) node, eg**

```
n = 4;  
mySqrGrStr = "<sqr [Num: " + n + "]" | [Num: '*result']>";  
g = activate mySqrGrStr.toGraph();  
  
g = `[PROPOSITION:*a(to_do [Line:'#L0'])]  
    [PROPOSITION:*b[Integer: '*z5']]  
    <fact?a/?b>`;   
x = activate g;
```



Some Future Directions

- Consider conjunction of preconditions in a rule together, eg for Sisyphus-I constraint re: room sharing and project members.
- Process constraints for truth maintenance (Mineau 1999).
- Conformity relation, more types/attributes.
- Process algorithm needs to be reconciled with the Rules of Inference; incorporate into pCG.
- Aspects of processes bear re-examination, eg stopping criteria, graph morphology, π efficiency, syntax.
- Environment: IDE, Emacs mode.
- Expts/Apps: invoking a process within a process, temporal logic, business processes.
- Treat pCG as a target language.

