Goldsmiths
UNIVERSITY OF LONDON

Computational Creativity Group

Assessing Progress in Building Autonomously Creative Systems

Simon Colton, Alison Pease*, Joseph Corneli, Michael Cook and Maria Teresa Llano

Computational Creativity Group
Department of Computing
Goldsmiths College, University of London, UK

*School of Computing
University of Dundee, UK

ccg.doc.gold.ac.uk
@GoldsmithsCCG

A Philosophical Perspective

- One overall aim of Computational Creativity research is to see creative software ubiquitously *embedded in society*, one day
- Creativity in people/software is a *secondary quality*
    - We project this adjective onto people/software as a short-hand for describing their behaviours
- The notion of creativity in people/software is *essentially contested* (so is “art”)
    - It’s a good thing to never reach agreement about creativity [thanks to Anna for this…]
- Assessment based on artefact quality alone may serve to emphasis the *humanity gap*
    - We want/expect a human connection when we consume created artefacts and hear of their creation
- We try to avoid talking about creativity of software and value of artefacts, in favour of addressing *progress* of a project with respect to *weak* and *strong* objectives

Strong and Weak Computational Creativity Aims

- Weak objectives in projects:
  - To produce artefacts of wonder with computational help
- Strong objectives in projects:
  - To convince people that what our software is doing - it’s behaviour - is worthy of being called ‘creative’
- Many projects have both strong and weak objectives
  - And the two types of objective will probably merge in time

Progress in Building Systems

- We want to have more formalism in describing how creative systems are built; how they operate at run-time; and the value of the artefacts they produce
    - So that we can use a more scientific approach to assessing progress in various Computational Creativity senses, as the engineering of a system progresses
- Ultimately, artefact-based assessment relies on the assumption that if a system is producing higher quality artefacts, then it must be getting more creative
    - This is sometimes a false assumption, and doesn't capture all notions of progress
    - We can easily 'cheat' to do this with stricter curation, hand-crafting, etc.
    - Assessment based on artefact quality alone may serve to emphasis the *humanity gap*
    - We know that handing over creative responsibility to software can mean decreasing artefact value in the short term (Latent heat effect, U-shaped creativity)
    - And we are increasingly in a more fluid situation with respect to the notion of "better quality", i.e., software can (and should?) invent new measures of value

Responsibilities

- Progress can also be measured in terms of the perceived creative autonomy that software systems have
  - And in general: in terms of how people perceive software behaviours with respect to issues of creativity, intelligence and autonomy
- It is *Time for Computational Creativity Researchers to Grow Up* and start talking about the creative responsibility they have in projects and the creative responsibilities they hand over to their software
  - Only this way will we be able to de-mystify and formalise creative acts in order to systematically measure progress in our field

Responsibilities, Responsibilities, Responsibilities

- Software isn’t another (human) jazz musician you play along with, it is actively and decisively programmed by us and others
  - This naturally changes the perception people have of software autonomy
  - So, we have to do more to address what software actually does, not just celebrate the fact that the ‘system’ came up with some brilliant creation
- We’re in a game of perceptions, and it doesn’t help to talk about *‘diffuse responsibilities’* or *‘murkier situations’* - in these cases, the default position is to project the autonomy/creativity onto the people in the loop
- People make decisions and do things, software makes decisions and does things. Each influences the other - these are things that can be at least approximately measured.
  - Things are murky right now, because we haven’t started formalising things
- See Colin’s talk tomorrow for an alternative perspective…

Overview of Approach

- We start with a simplifying assumption of a person/team building a single artefact generating system in a particular application domain, occasionally releasing a version, and/or producing output for a major milestone
- We've introduced a graphical formalism which helps to capture the creative acts which go into building and executing a system
  - This captures various timelines and enables visual comparison of systems, the effort that went into building them and their run-time behaviours
- These comparisons are coupled with an audience evaluation model, where audiences are asked about both the artefacts produced and the behaviours exhibited by the software and the programmer(s)
  - This enables an estimate of whether obvious or potential progress or regress has been made with respect to both strong and weak objectives

Background

- Please see the paper for some background in how researchers have historically assessed progress in Computational Creativity
- We build on the FACE model we introduced in earlier papers:
  - Creative acts are represented as tuples of generative acts, producing artefacts of the following forms: (F)raming information, (A)esthetic considerations, (C)oncepts, (E)xamples
  - We found this wasn't sufficient to capture software (+development) such as ANGELINA
- We also build on an enhanced version of the "creativity tripod": to (more easily) avoid the label of 'uncreative', software needs to exhibit behaviours onto which we can project the words:
  - Skill, appreciation, imagination, intentionality, learning, accountability, innovation, subjectivity and reflection (see AISB'14 paper)
  - This is an attempt to capture some of the higher level behaviours associated with creativity, such as autonomy, adaptability and self-awareness
  - **NB. These behaviours are necessary to avoid the label of 'uncreative'. That is not the same as being sufficient for gaining the label of 'creative' - I never said this…!**

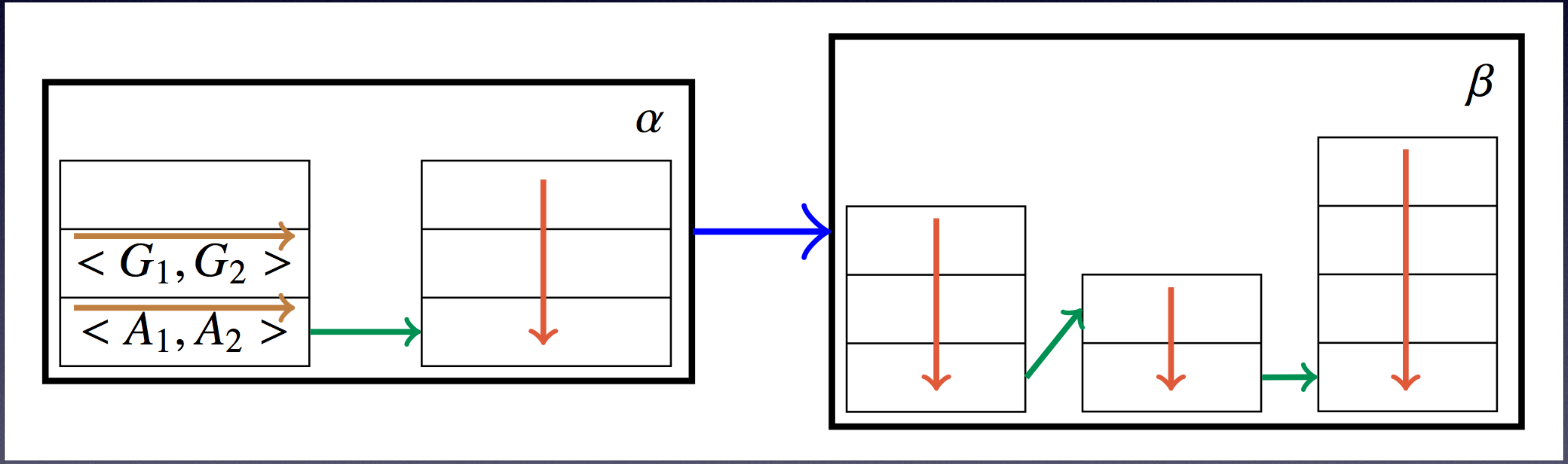
FACE Tuples

- We've extended the FACE model a little:
    - To include *administrative acts* as well as *generative acts*
        - Using [square brackets] and <pointy brackets>
        - Administration includes: (S)election of artefacts and (T)ranslation routines
    - To include three levels: (g)round, (p)rocess and (m)eta levels
    - To include repetition of creative acts with an asterix notation
- Examples:
    - Constraint solvers at run-time: $<\overline{C}_g, E_g>$
    - Machine learning systems: $<\overline{A}_g>$ then $<C_g, E^*_g, F_g=\overline{a}_g(e^*_g)>$ then $[\overline{T}(c_g)]$

Timelines

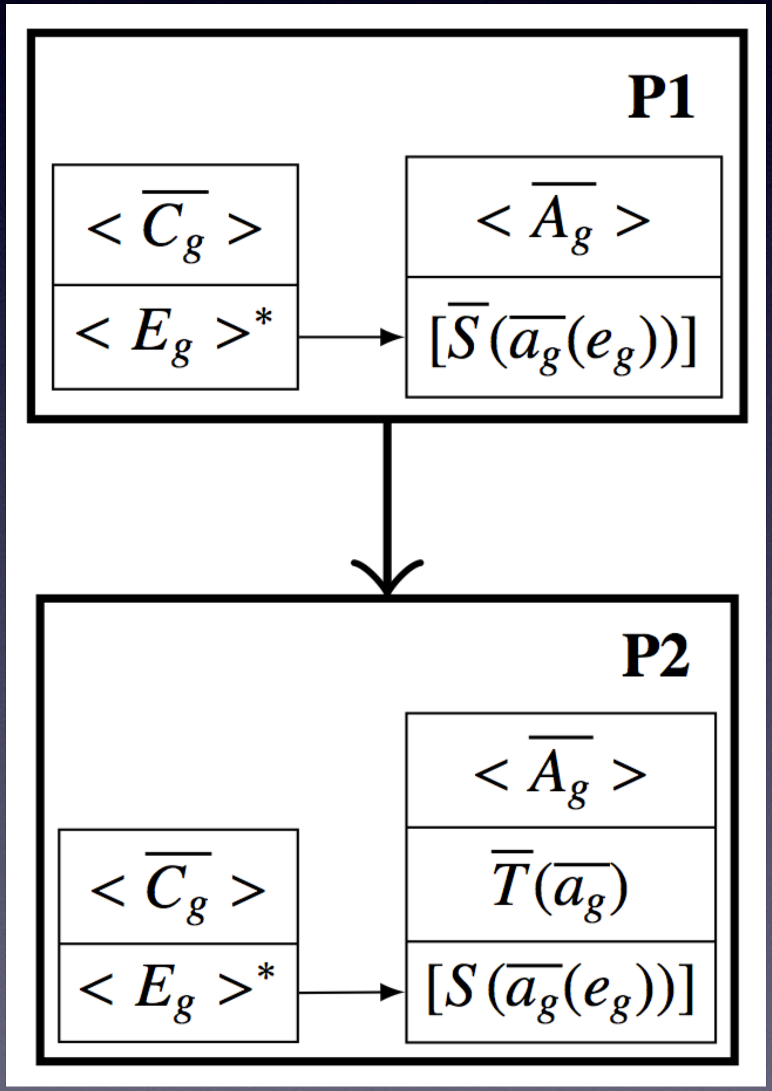
- We want to (start to) separate the creative responsibilities of the programmer(s) from those of the software, by looking at the creative acts performed
- There are four fairly obvious timelines:
  - System epochs: new versions of software
  - Development periods: coding and feedback
  - Subprocess level: passing of data around modules
  - Creative and/or administrative acts

Capturing Timelines



Poetry Generation Example

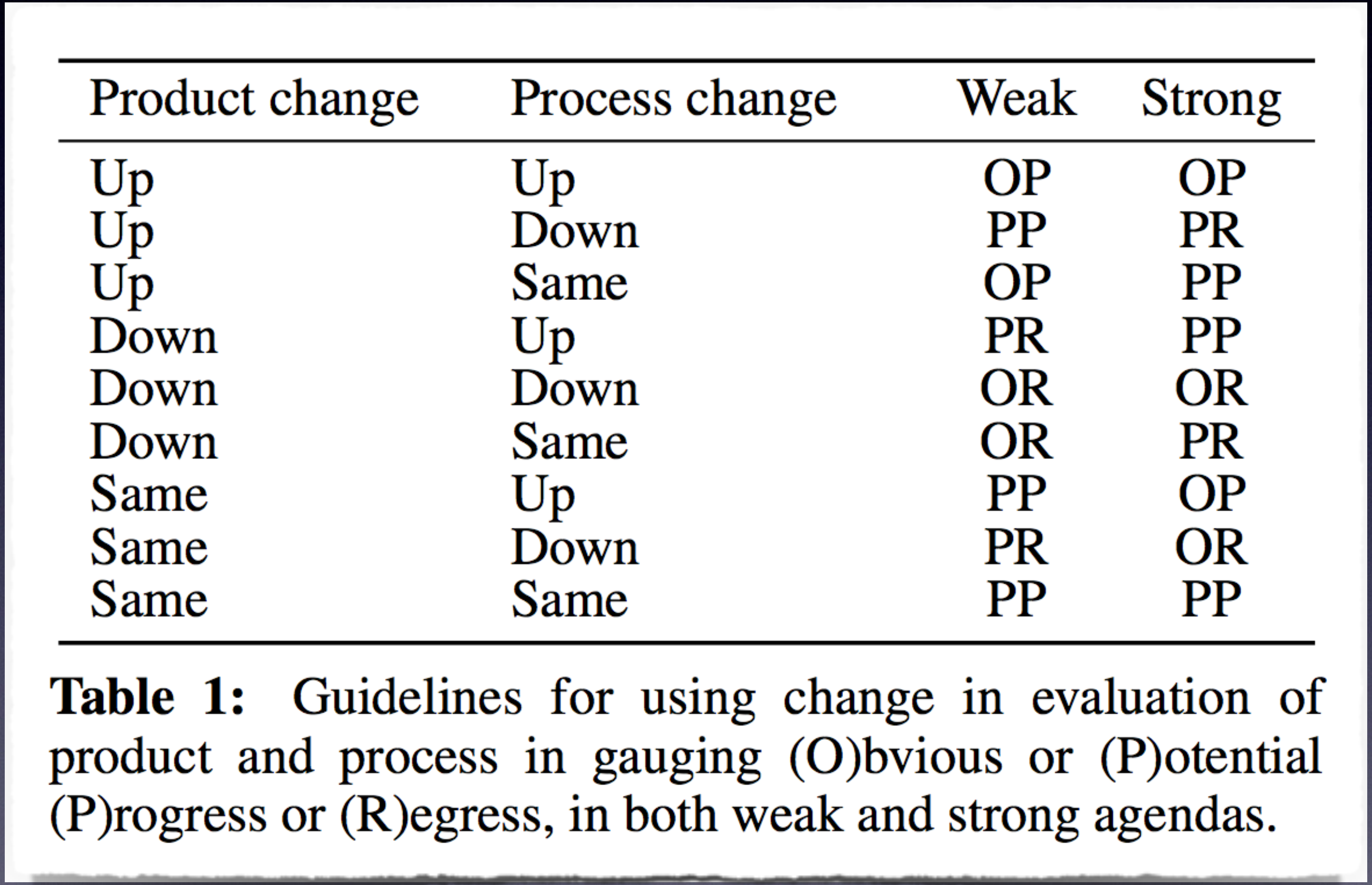
- Both programs operate in two stages
- In P1, the programmer employs his/her aesthetic to choose from the output of the system
- In P2, the programmer has translated this aesthetic into code, so that the software can do the choosing for them



Methodology: Usage of Diagrams from Version v1 to v2 of software

- During development by programmer
    - Compare diagrams informally to see if the creative acts by software have changed for the better, based on (i) the removal of bars and (ii) introduction of novel types of generative acts
- During audience evaluations
    - Work out in advance whether you primarily have strong or weak objectives in going from v1 to v2
    - Tell audience members about how v1 and v2 were developed and what they do at run-time. Show the users the diagrams to clarify things if needed
    - Ask audience members pointed questions about the creativity tripod behaviours with respect to the processes in v1 and v2. Ask them if their perceived level (or level of interest in) these behaviours has increased or decreased
    - Expose the audience to the artefacts produced by the system. Ask them if their perceived level of value of the artefacts has increased or decreased
    - Average somehow over the audience measures of process and product change
    - Use our guidelines (and adapt them to fit your purposes) to estimate obvious or potential progress or regress from v1 to v2

Guidelines

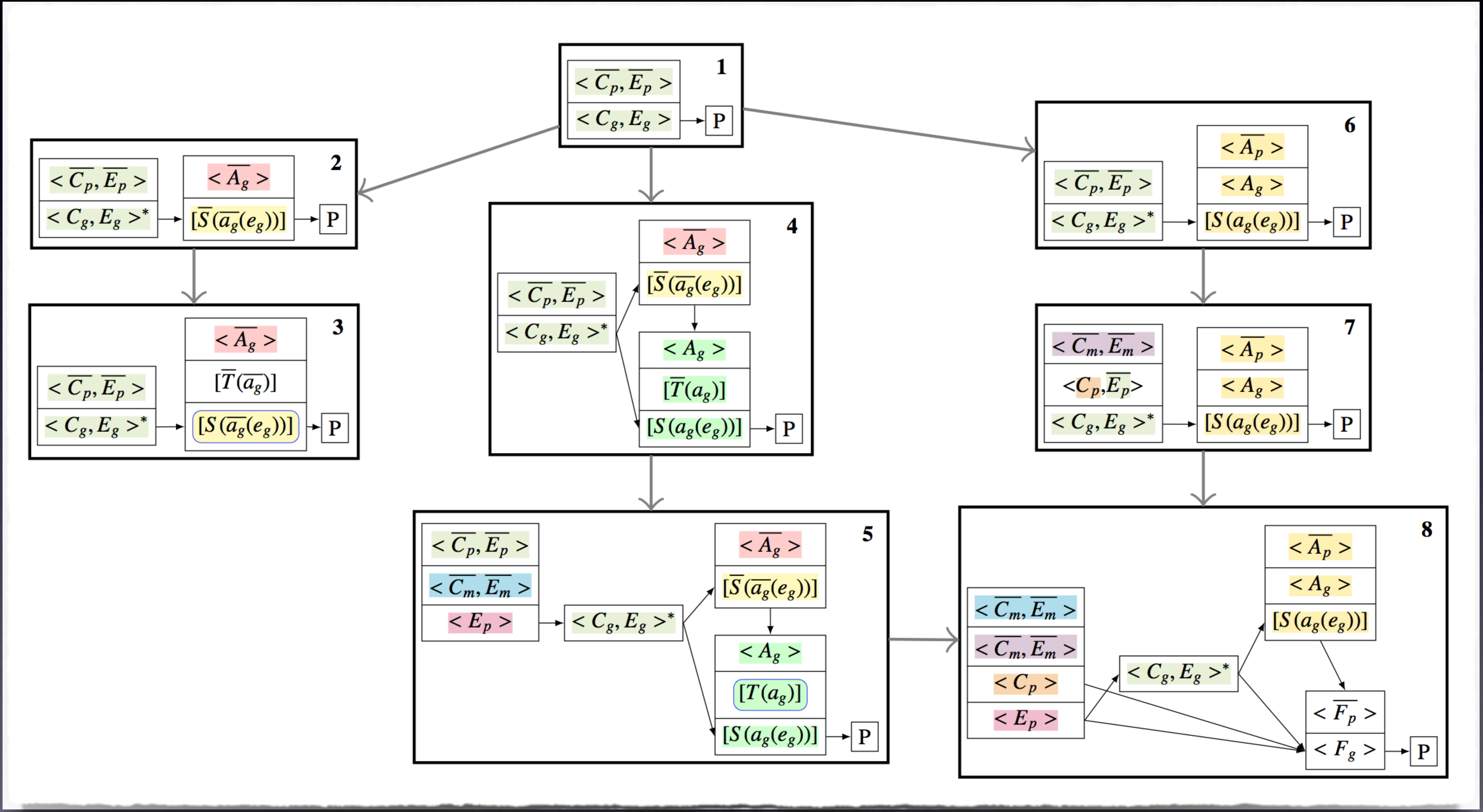


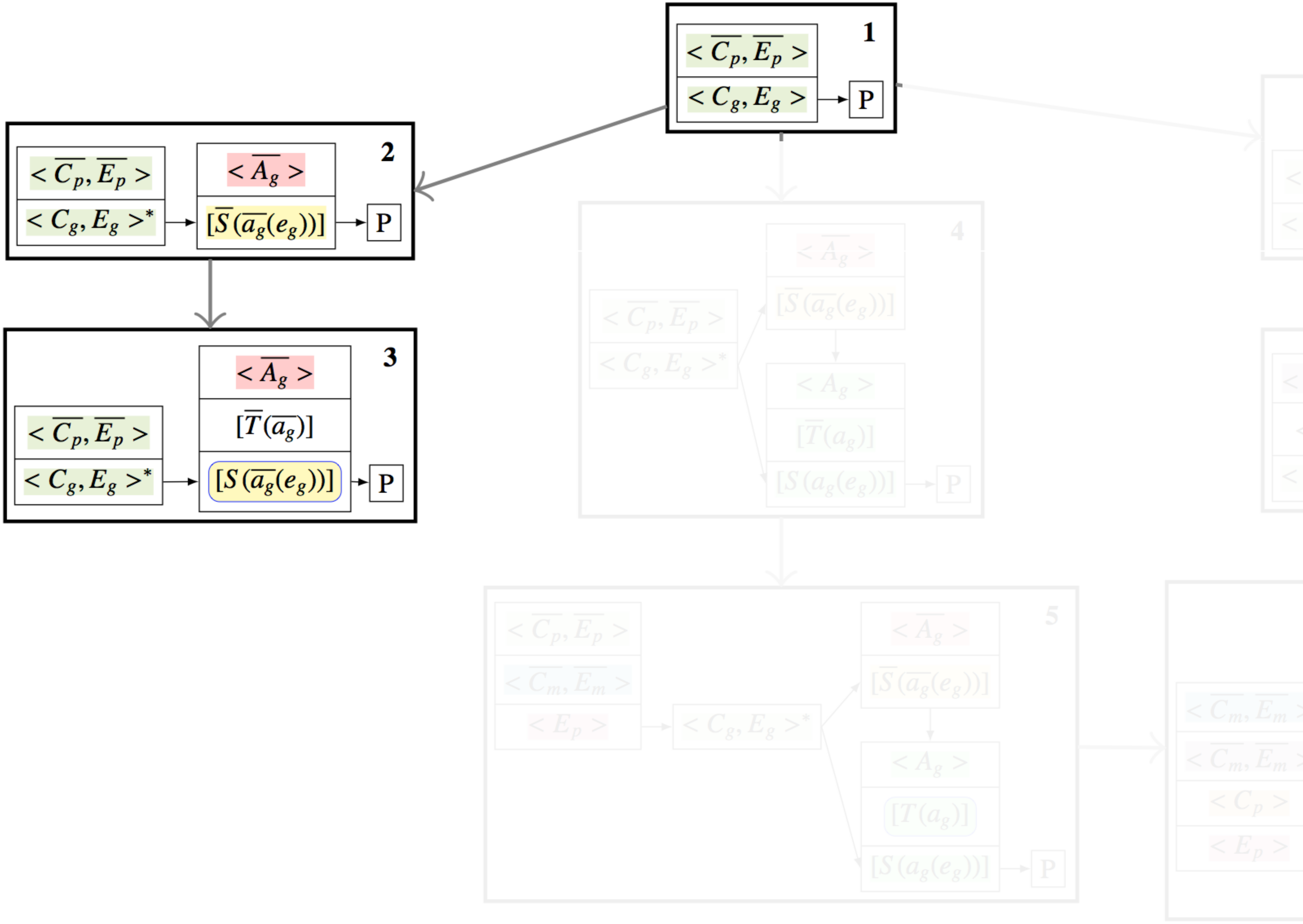
Product change	Process change	Weak	Strong
Up	Up	OP	OP
Up	Down	PP	PR
Up	Same	OP	PP
Down	Up	PR	PP
Down	Down	OR	OR
Down	Same	OR	PR
Same	Up	PP	OP
Same	Down	PR	OR
Same	Same	PP	PP

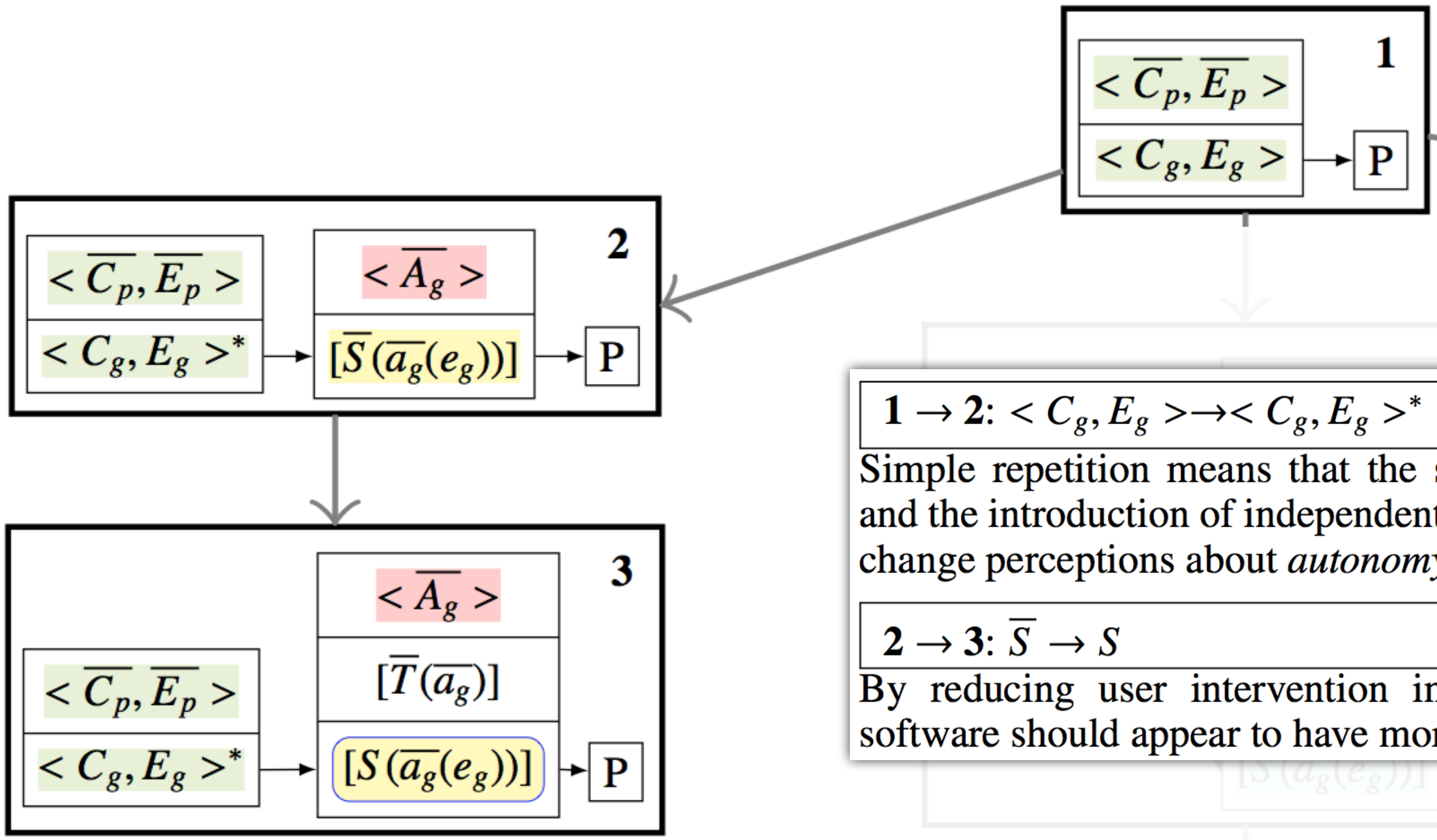
Table 1: Guidelines for using change in evaluation of product and process in gauging (O)bvious or (P)otential (P)rogress or (R)egress, in both weak and strong agendas.

A Case Study in Evolutionary Art

(Please see paper for more details)




1
$< \overline{C_p}, \overline{E_p} >$
$< C_g, E_g >$
P
2
$< \overline{C_p}, \overline{E_p} >$
$< C_g, E_g >^*$
$< \overline{A_g} >$
$[\overline{S}(\overline{a_g}(e_g))]$
P
3
$< \overline{C_p}, \overline{E_p} >$
$< C_g, E_g >^*$
$< \overline{A_g} >$
$[\overline{T}(\overline{a_g})]$
$[S(\overline{a_g}(e_g))]$
P
4
$< \overline{C_p}, \overline{E_p} >$
$< C_g, E_g >^*$
$< \overline{A_g} >$
$[\overline{S}(\overline{a_g}(e_g))]$
$< A_g >$
$[\overline{T}(a_g)]$
$[S(a_g(e_g))]$
P
5
$< \overline{C_p}, \overline{E_p} >$
$< \overline{C_m}, \overline{E_m} >$
$< E_p >$
$< C_g, E_g >^*$
$< \overline{A_g} >$
$[\overline{S}(\overline{a_g}(e_g))]$
$< A_g >$
$[T(a_g)]$
$[S(a_g(e_g))]$
P
$< \overline{C_m}, \overline{E_m} >$
$< \overline{C_m}, \overline{E_m} >$
$< C_p >$
$< E_p >$


1
$< \overline{C}_p, \overline{E}_p >$
$< C_g, E_g >$
P
2
$< \overline{C}_p, \overline{E}_p >$
$< C_g, E_g >^*$
$< \overline{A}_g >$
$[\overline{S}(\overline{a_g}(e_g))]$
P
3
$< \overline{C}_p, \overline{E}_p >$
$< C_g, E_g >^*$
$< \overline{A}_g >$
$[\overline{T}(\overline{a_g})]$
$[S(\overline{a_g}(e_g))]$
P

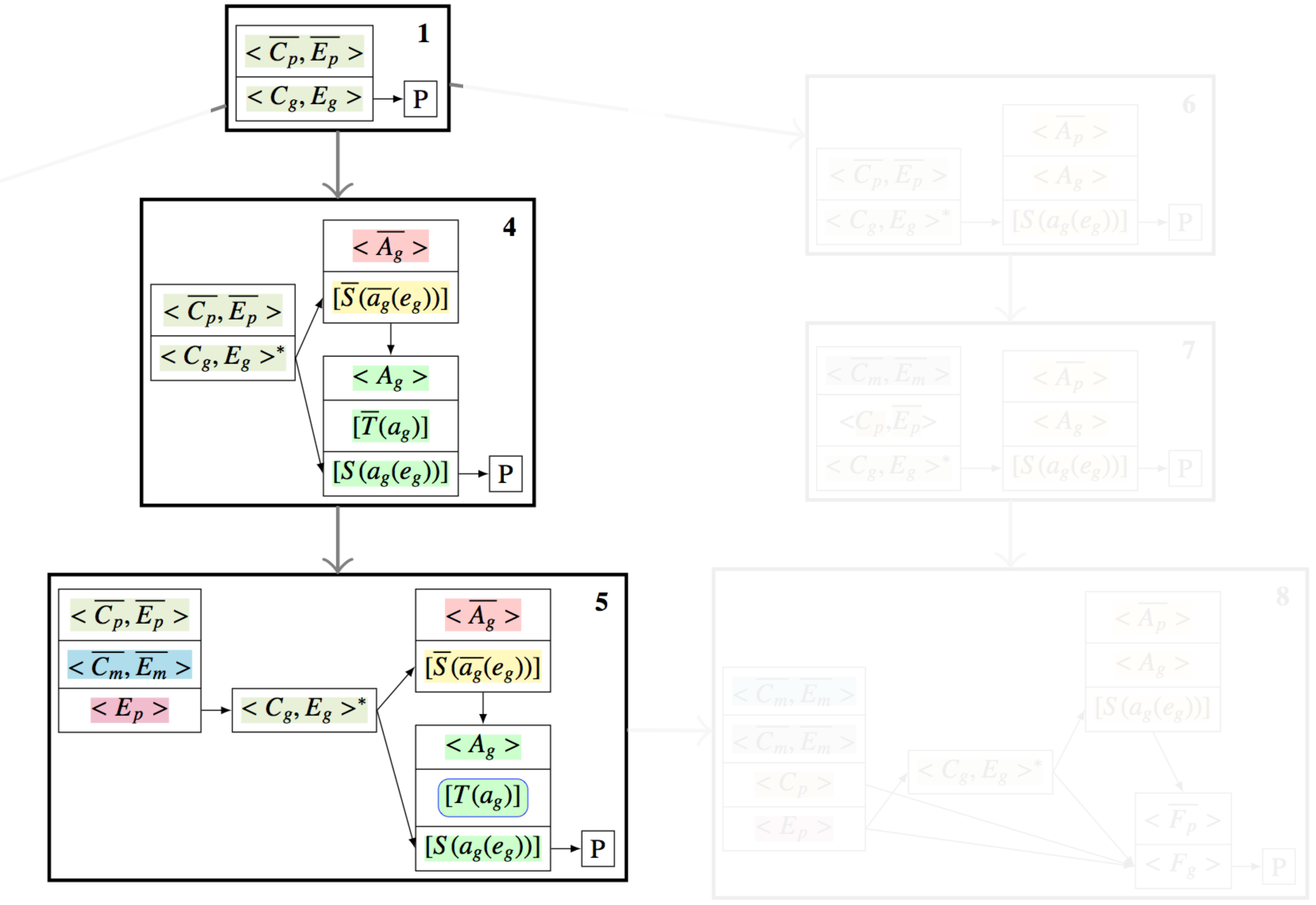
1 → 2: $< C_g, E_g > \to < C_g, E_g >^*$

Simple repetition means that the software has more *skill*, and the introduction of independent user selection shouldn't change perceptions about *autonomy*.

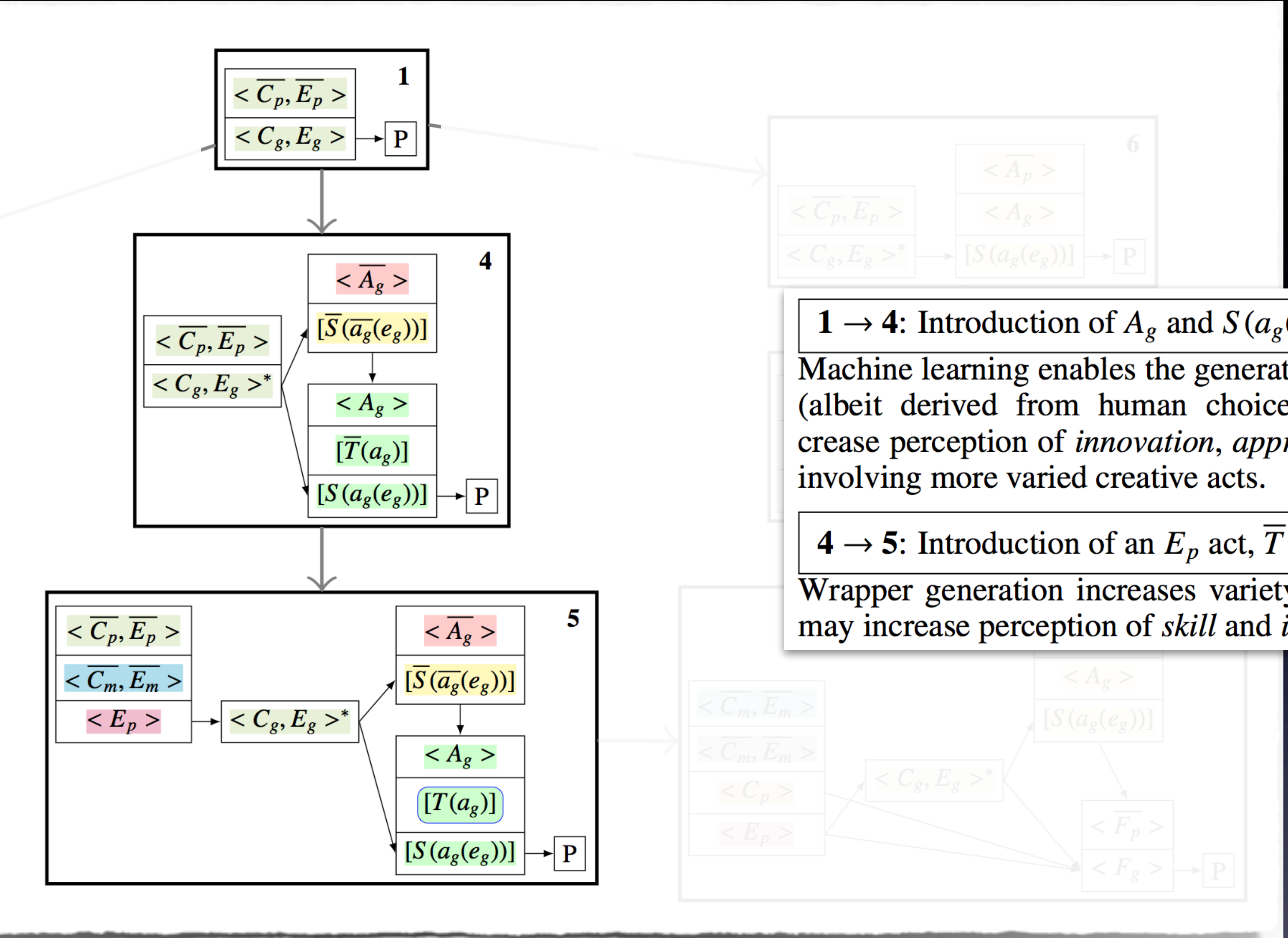
2 → 3: $\overline{S} \to S$

By reducing user intervention in choosing images, the software should appear to have more *skill* and *autonomy*.

ID	Event	Explanation
1.1.1	$\overline{C}_p$	The programmer invents the idea of crossing over two sets of mathematical functions to produce a new set of mathematical functions.
1.1.1	$\overline{E}_p$	The programmer implements a wrapper method that takes a set of mathematical functions and applies them to each (x, y) co-ordinate in an image to produce an RGB colour.
1.1.2	C_g	The software generates a new set of functions by crossing over two pairs of functions.
1.1.2	E_g	The software applies these functions to the (x, y) co-ordinates of an image, to produce a piece of abstract art.
2.2.1	$\overline{A}_g$	The programmer had in mind a particular aesthetic (symmetry) for the images.
2.2.2	$\overline{S}(\overline{a_g}(e_g))$	The programmer uses his/her aesthetic to select a preferred image for printing.
3.2.2	$\overline{T}(\overline{a_g})$	The programmer took their aesthetic and turned it into code that can calculate a value for images.
3.2.3	$S(\overline{a_g}(e_g))$	The software applies the aesthetic to select one of a set of images produced by crossover and the wrapper.



1
$<\overline{C_p},\overline{E_p}>$
$<C_g,E_g>$
P
4
$<\overline{C_p},\overline{E_p}>$
$<C_g,E_g>^*$
$<\overline{A_g}>$
$[\overline{S}(\overline{a_g}(e_g))]$
$<A_g>$
$[\overline{T}(a_g)]$
$[S(a_g(e_g))]$
P
5
$<\overline{C_p},\overline{E_p}>$
$<\overline{C_m},\overline{E_m}>$
$<E_p>$
$<C_g,E_g>^*$
$<\overline{A_g}>$
$[\overline{S}(\overline{a_g}(e_g))]$
$<A_g>$
$[T(a_g)]$
$[S(a_g(e_g))]$
P
6
$<\overline{C_p},\overline{E_p}>$
$<C_g,E_g>^*$
$<\overline{A_p}>$
$<A_g>$
$[S(a_g(e_g))]$
P
7
$<\overline{C_m},\overline{E_m}>$
$<\overline{C_p},\overline{E_p}>$
$<C_g,E_g>^*$
$<\overline{A_p}>$
$<A_g>$
$[S(a_g(e_g))]$
P
8
$<\overline{C_m},\overline{E_m}>$
$<\overline{C_m},\overline{E_m}>$
$<C_p>$
$<E_p>$
$<C_g,E_g>^*$
$<\overline{A_p}>$
$<A_g>$
$[S(a_g(e_g))]$
$<\overline{F_p}>$
$<F_g>$
P



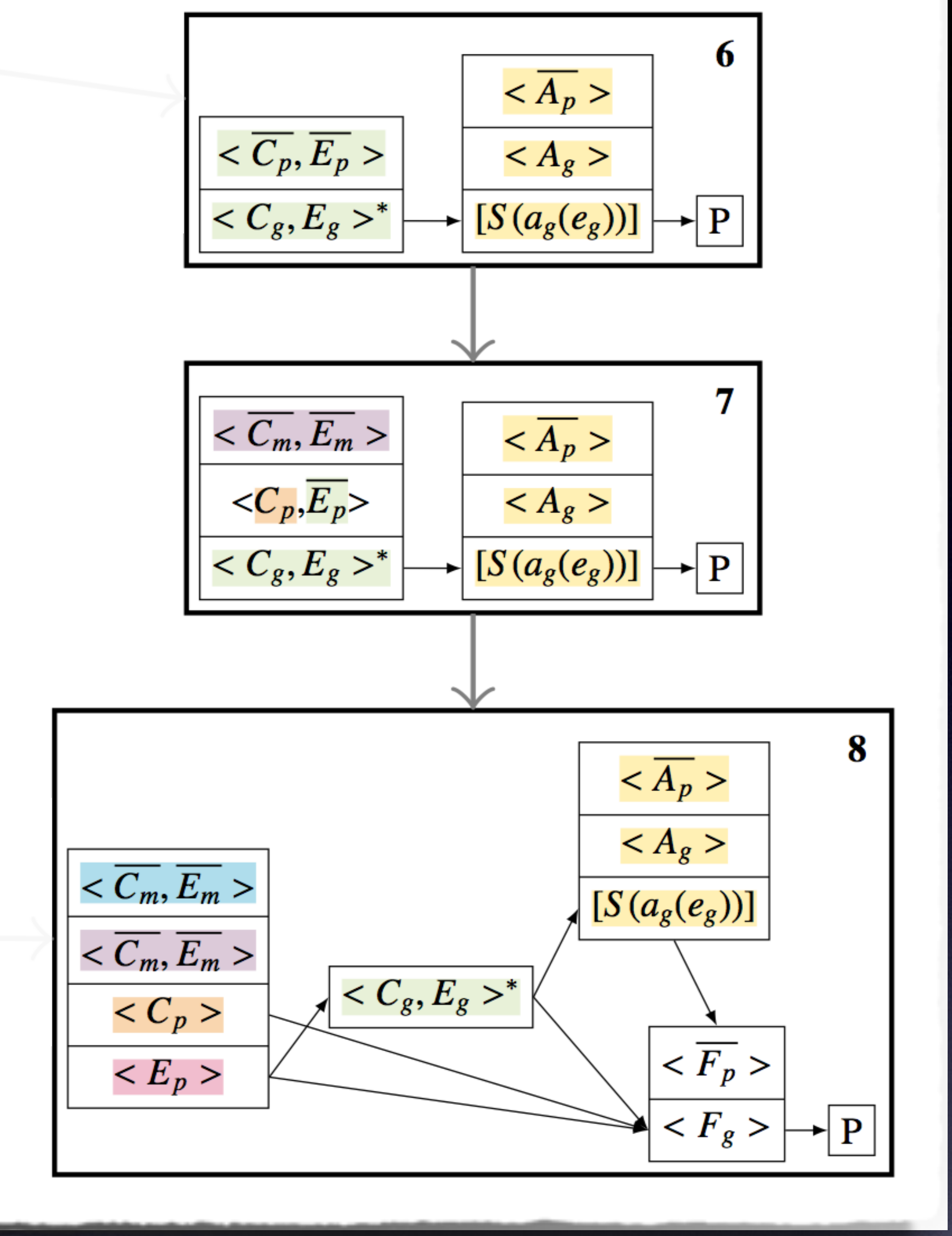
1 → 4: Introduction of A_g and $S(a_g(e_g))$ acts

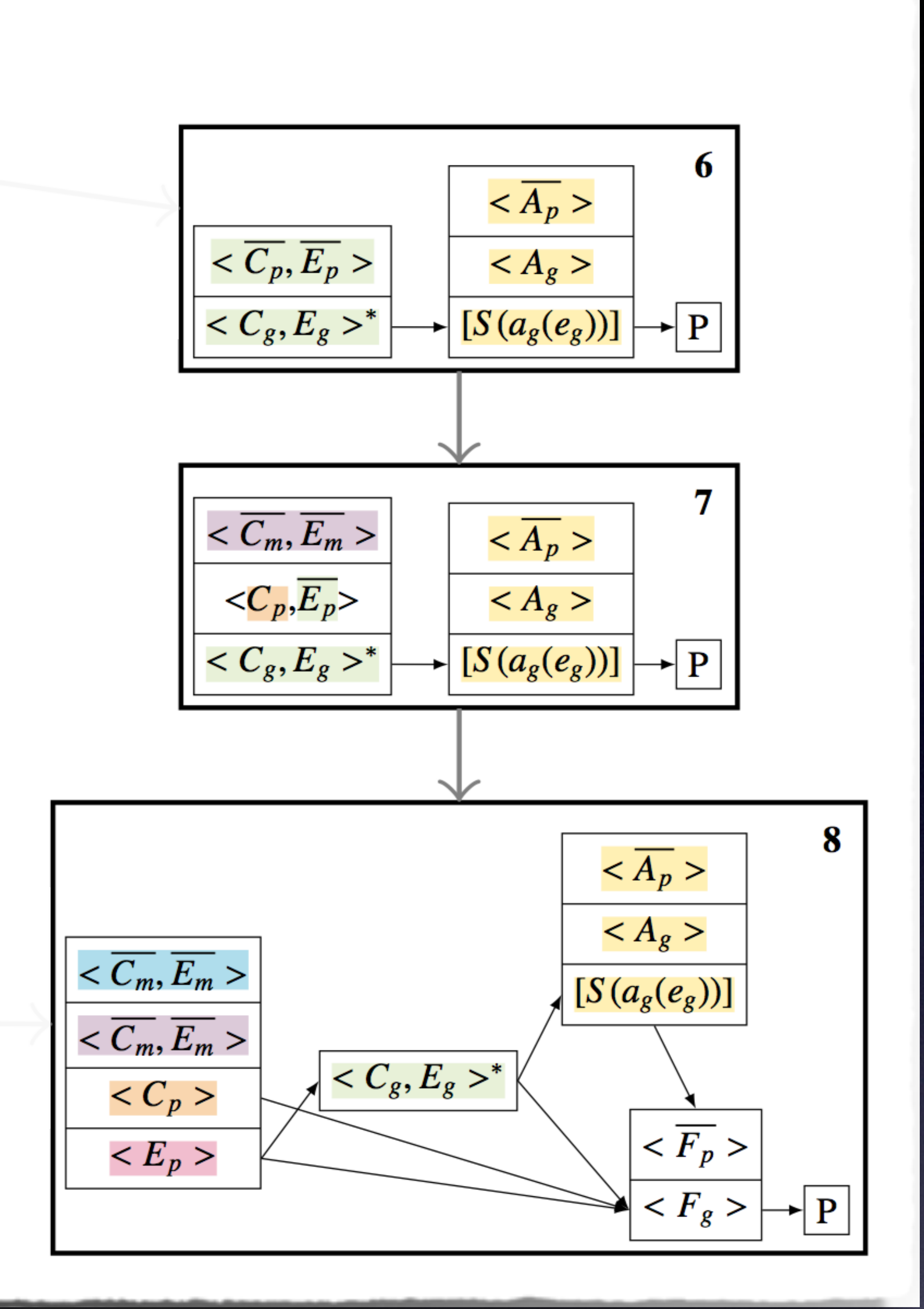
Machine learning enables the generation of novel aesthetics (albeit derived from human choices), which should increase perception of *innovation*, *appreciation* and *learning*, involving more varied creative acts.

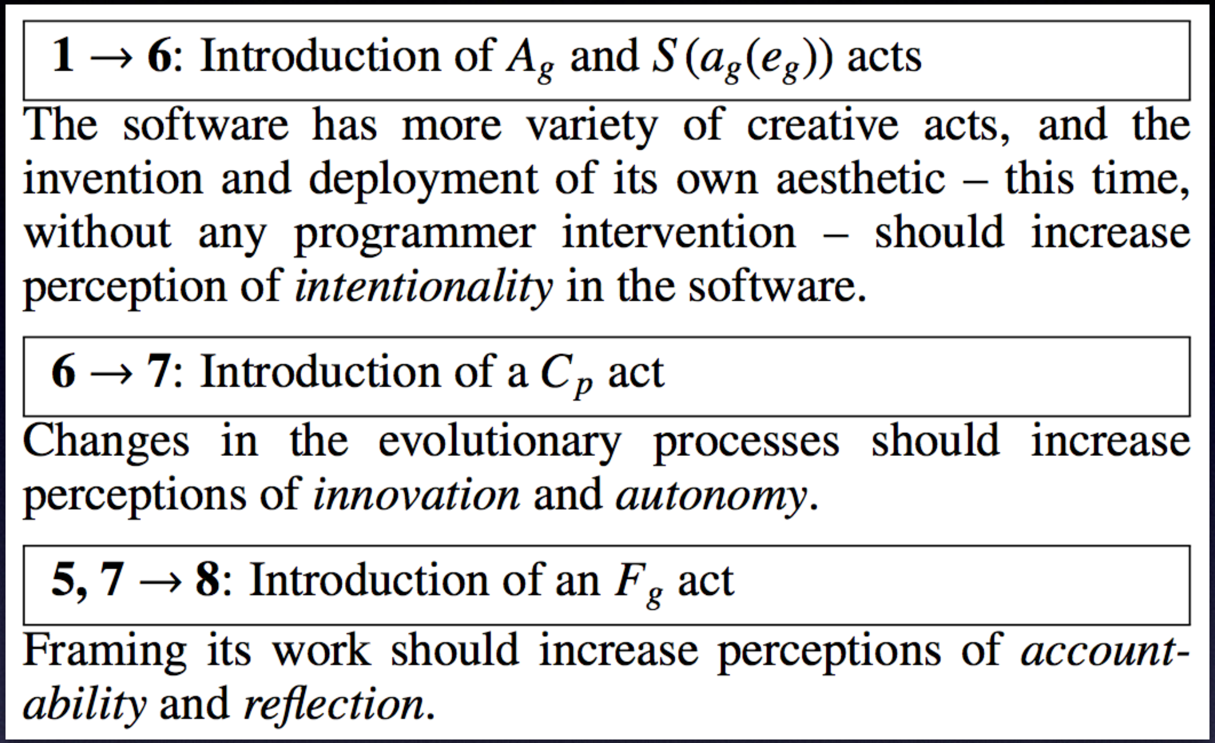
4 → 5: Introduction of an E_p act, $\overline{T} \rightarrow T$

Wrapper generation increases variety of creative acts, and may increase perception of *skill* and *imagination*.

4.3.1	A_g	The software uses machine learning techniques to approximage the programmer's aesthetic.
4.3.2	$\overline{T}(a_g)$	The programmer hand-translates the machine learned aesthetic into code.
4.3.3	$S(a_g(e_g))$	The software applies the new aesthetic to choosing the best image from those produced.
5.1.2	$\overline{C_m}$	The programmer has the idea of getting the software to search through a space of wrapper routines.
5.1.2	$\overline{E_m}$	The programmer implements this idea.
5.1.3	E_p	The software invents a new wrapper.
5.4.2	$T(a_g)$	The software translates the machine-learned aesthetic itself into code.


6
$< \overline{C_p}, \overline{E_p} >$
$< C_g, E_g >^*$
$< \overline{A_p} >$
$< A_g >$
$[S(a_g(e_g))]$
P
7
$< \overline{C_m}, \overline{E_m} >$
$< C_p, \overline{E_p} >$
$< C_g, E_g >^*$
$< \overline{A_p} >$
$< A_g >$
$[S(a_g(e_g))]$
P
8
$< \overline{C_m}, \overline{E_m} >$
$< \overline{C_m}, \overline{E_m} >$
$< C_p >$
$< E_p >$
$< C_g, E_g >^*$
$< \overline{A_p} >$
$< A_g >$
$[S(a_g(e_g))]$
$< \overline{F_p} >$
$< F_g >$
P





$1 \rightarrow 6$: Introduction of A_g and $S(a_g(e_g))$ acts

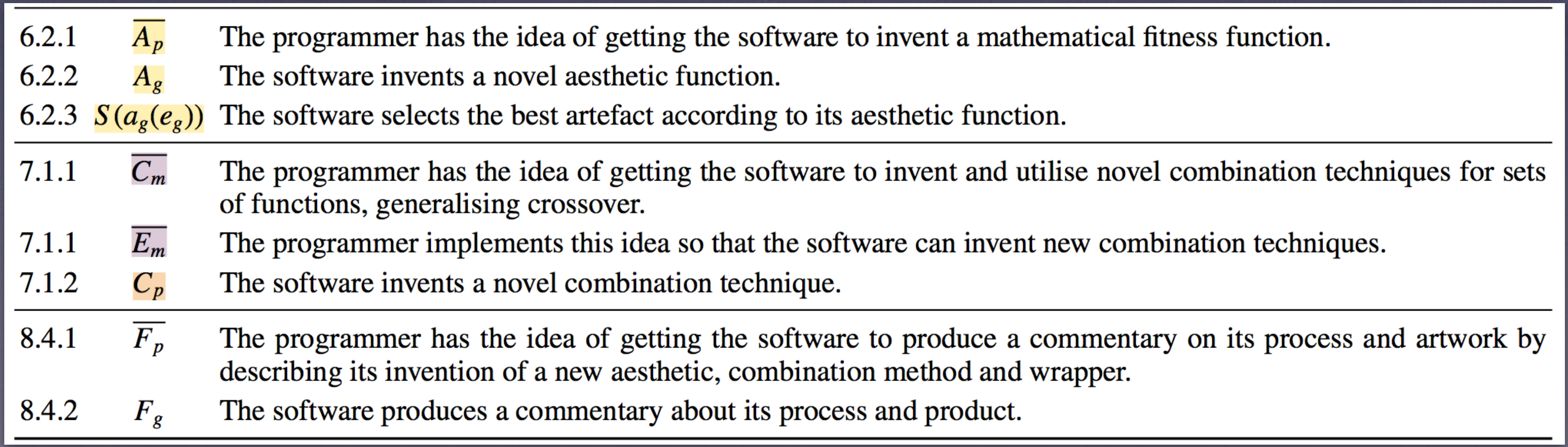
The software has more variety of creative acts, and the invention and deployment of its own aesthetic – this time, without any programmer intervention – should increase perception of *intentionality* in the software.

$6 \rightarrow 7$: Introduction of a C_p act

Changes in the evolutionary processes should increase perceptions of *innovation* and *autonomy*.

$5, 7 \rightarrow 8$: Introduction of an F_g act

Framing its work should increase perceptions of *accountability* and *reflection*.



6.2.1	$\overline{A_p}$	The programmer has the idea of getting the software to invent a mathematical fitness function.
6.2.2	A_g	The software invents a novel aesthetic function.
6.2.3	$S(a_g(e_g))$	The software selects the best artefact according to its aesthetic function.
7.1.1	$\overline{C_m}$	The programmer has the idea of getting the software to invent and utilise novel combination techniques for sets of functions, generalising crossover.
7.1.1	$\overline{E_m}$	The programmer implements this idea so that the software can invent new combination techniques.
7.1.2	C_p	The software invents a novel combination technique.
8.4.1	$\overline{F_p}$	The programmer has the idea of getting the software to produce a commentary on its process and artwork by describing its invention of a new aesthetic, combination method and wrapper.
8.4.2	F_g	The software produces a commentary about its process and product.

Application of Formalism

- We've applied this formalism, largely successfully to:
    - Compare/contrast versions of the HR system and the AM/Eurisko systems
    - The full-FACE poetry generation systems, and other poetry systems
    - Aspects of The Painting Fool's development
    - Timelines in the development and deployment of ANGELINA
    - Our own evolutionary art software ELVIRA
- Response to the reviewer: yes this can be applied to software which runs industrial factories, etc. (i.e., not designed for creative purposes)
    - But that doesn't detract from the fact that the formalism helps identify the creative acts and responsibilities undertaken by the programmer and software, unlike other formalisms

Future Work

- It's time to tie up this framework with those of others:
  - Wiggins' creative systems framework, particularly for being more precise about the kind of search that systems undertake, and the search spaces they explore
  - Ritchie's criteria for evaluating artefacts, particularly for improving acuity of the audience evaluation aspects
  - Our IDEA model of an ideal audience, cognitive expenditure and change in well-being
- We plan to see whether audience appreciation of what software does is increased with exposure to the diagrams, and whether they are of value in practice to developers of creative systems
  - Motivating as well as describing development paths
- We also have a series of hypotheses relating to our philosophical position which are mature enough now to be ready for scientific evaluation
  - Existence/nature exacerbation of the humanity gap; value of commentary generation; etc.

To Wrap Up…

- The framework can successfully capture aspects of the development, processing and output of (abstracted versions) of creative systems in various application domains
    - But it still has a number of limitations and fudge factors
- We'd like to help you apply it to describe (in rationalised terms) how your software was developed, how it runs and what it produces
    - In order to find further limitations of the approach

Questions…?