# 1 Preference order

1. Look at acceptable/reject preferences to set list of candiates;
2. Look at better/worse prefences to reduce set of candidates (providing that they are not inconsistent (e.g. a tie));[1]
3. Look at best/worst preferences (providing it does not remove only candidate) to reduce set of candidates (providing that they are not inconsistent);
4. Look at indifferent (unary and bianry) preferences. If all indifferent then choose randomly, else if only one left choose candidate, otherwise `abort` as no preference can be chosen.

1. pass 1:
    - (a) remove all candidates that are not accepted;
    - (b) remove all candidates that are rejected;
2. pass 2:
    - (a) remove all candidates that are on the RHS of a better operator;
    - (b) remove all candidates that are on the LHS of a worse operator;
3. pass 3:
    - (a) partition candidates into best, normal, and worst lists, whilst marking if all the candidates in the set are indifferent;
    - (b) choose the first set that is non-empty;
4. clean-up ...

---

[1] $A > B$ and `best(`$B$`)`, $B$ will be discounted.

# 2 Model Design Notes

## 2.1 Transfering state

The main top-level processes within the CSP (Communicating Sequential Processes) soar model exchange information by essentially sending and recieivng a set of update messages. The obvious modelling strategy is to provide a single event that is parameterised by the set of update messages. This approach has a significant limitation, when considered in the context of an eager model checker, namely, that its compiler prepares a handler for every possible set of update messages, even if they are never used.

> **Aside 2.1:** The number of handlers prepared by an eager model checker's compiler is $2^n$, where $n$ is the potential number of update messages.

An alternative approach is to transmit each update message indvidually. In this case the eager model checker compiler prepares a single handler for each potential update message. This changes the compile space from exponeitial to linear. However, the cost here is in the permutations (orders) in which a collection of messages can be sent. Therefore, a technique to choose a single represntative sequence of update messages for each collection. This can be achived in a variety of ways, such as providing a total order over the collection of messages and transmiting them in that order.

> **Aside 2.2:** The FDR (Failure-Divergence and Refinement) tool provides the `chase` operator for choosing arbitrary sequence of hidden events. With care this can be used to pick a single representative sequence of update messages for each collection, as discussed in Appendix **??**.

In summary, the top-level processes are designed to transmit their outputs in a represetative stream of update messages, that are marked by special start and stop co-ordination events. This has the following implications for each of the top-level processes whilst in the input mode:

1. all update messages are accepted, though they can be immediately ignored (dropped);
2. the update messages can be received in any order;
3. an indvidual update message will only be sent once, in any input session.

# 3 Soar Translator

## 3.1 Kinds of rule

There are three kinds of rule in the CSP model, namely, proposal rules, elaboration rules, and operation rules. The consequent capabilities of a:

**proposal rule** may only contain capabilities of the form *propose.p_set.*(⟨op⟩, ⟨pri⟩, ⟨eqv⟩), where ⟨op⟩ is a valid operation, ⟨pri⟩ is an integer priority, and ⟨eqv⟩ is a boolean equivallence flag, which models the soar notion of operator indifference.

**elaboration rule** may only contain capabilities of the form *wme.add.*⟨path⟩.⟨val⟩, where ⟨path⟩ is a valid path (i.e. sequence of attribute names), and ⟨val⟩ is an appropriate value. Note that these elaborated capabilities are treated in the CSP model as having I-Support.

**operation rule** may only contain capabilities of the form *wme.add.*⟨path⟩.⟨val⟩ or *wme.del.*⟨path⟩.⟨val⟩, where ⟨path⟩ is a valid path, and ⟨val⟩ is an appropriate value. Note that these capabilities are treated in the CSP model as having O-Support.

The antecedent capabilities of a:

**proposal rule** may only contain capabilities of the form *wme.val.*⟨path⟩.⟨val⟩ or *wme.size.*⟨path⟩.⟨i⟩[2], where ⟨path⟩ is a valid path (i.e. sequence of attribute names), ⟨val⟩ is an appropriate value, and ⟨i⟩ is an integer representing the number of values contained in location ⟨path⟩.

**elaboration rule** may only contain capabilities of the form *wme.val.*⟨path⟩.⟨val⟩ or *decide.d_get.*⟨op⟩, where ⟨path⟩ is a valid path (i.e. sequence of attribute names), ⟨val⟩ is an appropriate value, and ⟨op⟩ is an appropriate operation.

**operation rule** may only contain capabilities of the form *decide.d_get.*⟨op⟩, where ⟨op⟩ is an appropriate operation; it may also contain capabilities of the forms available to the proposal rules.

The CSP soar model expects the soar2csp generator to produce three rule sets, one for each kind of rule.

## 3.2 Naming conventions

The CSP soar naming conventions are as follows:

- All soar values should be contained in a single CSP datatype whose top level branches end with the string “_v”.
- All soar attribute names should be contained in a single CSP datatype whose top level branches end with the string “_a”.
- All soar proposal rule names should be contained in a single CSP datatype whose top level branches end with the string “_prn”.
- The names of all soar proposal rule sets should end with the string “_pr”.
- All soar elaboration rule names should be contained in a single CSP datatype whose top level branches end with the string “_ern”.

---

[2] The paths in the size entries are limited to those that cannot have I-supported values. This is required for ensuring that Soar semantics are upheld within the CSP model.

- The names of all soar elaboration rule sets should end with the string "_er".
- All soar operation rule names should be contained in a single CSP datatype whose top level branches end with the string "_orn".
- The names of all soar operation rule sets should end with the string "_or".
- All other generated names should end with the string "_gen".

## 3.3 Interfacing with the CSP model

A soar rules should be of the form $\langle$rn$\rangle$, $\langle$AC$\rangle$, $\langle$CC$\rangle$, where $\langle$rn$\rangle$ is a valid rule name, $\langle$AC$\rangle$ is a set of antecedent capabilityies, and $\langle$CC$\rangle$ is a set of consequence capabilities.

The CSP module representing the working memory ought to be constructed using the following command: `WME = instance SoarHeap(PropRules,` ...

# List of Abbreviations

The entries in this list of abbreviations are either *general* or *product-specific*, where: the text for a *general* abbreviation is in lower-case words; and the text for a *product-specific* abbreviation is in capitalised words. The only exception to this rule is when an abbreviation contains another abbreviation, in which case the inner abbreviation is formatted as an upper case word.

| | |
|---|---|
| **CSP** | Communicating Sequential Processes |
| **FDR** | Failure-Divergence and Refinement |