A Rewriting Prolog Semantics (Kulas, 2000)

Nathaniel Wesley Filardo

April 8, 2008

Outline

- Prolog deals in Terms.
- Recursive tree structures. Arms of recursive definition:
  - Variable (*e.g.* `X`).
  - Functor and subtrees (`edge(·,·,·)`).
- Substitutions on terms send variables to terms:
  - `edge(X,X,0)` subject to $[1/\texttt{X}]$ is `edge(1,1,0)`.
- Two terms *unify* to produce a substitution which sends both of them to the same term:
  - `edge(X,X,Y)` unifies with `edge(A,B,B)` to produce `[X/A][X/B][X/Y]` and `edge(X,X,X)`.
  - `edge(1,X,3)` unifies with `edge(A,Y,B)` to produce `[1/A][X/Y][3/B]` and `edge(1,X,3)`.

- Horn Clause:
  - Let x_i be positive literals.
  - A Horn clause is then any disjunction of x_i with at most one positive literal:

$$\neg x_1 \vee \neg x_2 \vee \neg x_3 \cdots \vee x_n$$

  - This can be interpreted as

$$(x_1 \wedge x_2 \wedge x_3 \wedge \ldots) \vdash x_n$$

- A Prolog program is a collection of Horn clauses.
- Written `head :- subgoal1, subgoal2, ... .`
  - The `,` character acts *conjunctively*.
- Disjunct thusly: `head :- way1.  head :- way2.`
- Evaluation is top-down, left-to-right.
  - The semantics of Prolog in one sentence.
- Base case: `p(1) :- true.` or just `p(1).`

- Given some subset ϕ of a program, the *resolvent* of X is the right hand side of the first rule in ϕ whose head unifies with X.
- In this context, X is termed a "goal"
- The driving force of a Prolog program is the initial goal.

- Prolog can be queried about failure of a goal (might not halt!).
- Called "failure as negation", denoted $\backslash +$.
- Can have somewhat odd effects with non-ground queries:
  - ```
    in(X) :- \+(out(X)).
    out(alice).
    ```
  - out(alice) and in(bob) succeed.
  - in(X) fails!
    - because out(alice) is true, out(X) is true for some X.
    ```

- Oh, sigh, the cut.
- The cut is a primitive operator which inhibits backtracking, denoted !.

```
r :- p(X) , !  , q(X).
p(1).
p(2).
q(2).
```

- In the above program, `r` will fail:
  - `X` will be set to 1.
  - `q(1)` is not provable.
  - So we will backtrack...
  - Across a cut, which fails.
- This is said to be a “red cut” (alters semantics).
  - Alternative is a “green cut”

- Meta-call exists to encapsulate the behavior of cuts.
- Consider: `r :- call((p(X) , !, q(X))), r(X).`
- It is OK for `r(X)` to fail, in which case the call will be retried.
- It is not OK for `q(X)` to fail – this will fail the call, and r by extension.

- Transfer of control up the stack (generalized cut).
  - Similar try/except/raise or raise/handle in other languages.
- Catch with catch$(Goal, Catcher, Recover)$.
- Throw with throw$(Ball)$.
  - *Ball* is unified with *Catcher* to see if the pattern matches.
  - If so, the substitution is used to process *Recover*.
  - If not, control moves further up the stack, to the outer catch goal.

- findall(*Instance*, *Goal*, *List*) succeds if List is unifiable with the list of all terms unifiable with Instance which make Goal provable.
- `bagof(Instance, Goal, List)` similar, except considers free variables in Goal (not occuring in Instance). Can be backtracked into to get new Lists for each grounding of free variables.
- `setof(Instance, Goal, List)` is like bagof except sorted and without duplicates.

- asserta (X) alters the program to behave as if X had been a rule.
- retract (X) alters the program to remove (the first ocurrence of) X.
- Note that choice points made earlier do not get updated in either case!
  - That is, upon backtracking, we will not consider newly asserted rules until we begin a new subgoals; even then the consideration will be confined to subgoals.
  - Also, deleting a rule we used to get to the current state does not alter the current state.

- repeatis a built-in, definable as
  `repeat.  repeat:- repeat.`
- Disjunctive syntactic sugar: `x :- X; Y; Z` same as `x :- X. x :- Y. x :- Z` except that cuts effect the whole rule.
- If-then-else syntactic sugar: `x :- If -> Then; Else` same as `x :- If, !, Then.  x :- Else.` except that any cuts in `Then` or `Else` extend beyond the if-then-else.

σ_i is a substitution.

$\mathbf{u}$ undoes a substitution.

ϵ denotes the identity substitution.

$\wedge$ denotes the empty string.

$\mathbf{P}[\![\Pi]\!]$ is the prolog program under definition.

G_0 is the top-level goal (G_i are subgoals).

$\mathbf{D}_{\phi_j}[\![G_j]\!]$ is the "derivation operator" with current goal G_i and context ϕ_j.

$\mathbf{C}_{\phi_i}[\![G_j]\!]$ is a continuation for goal G_j with context ϕ_i.

$\bullet$ denotes the end of the world.

- Goals inside derivations may be lists of derivations.
  - Pattern match head, tail as $X, \vec{Y}$.
- Rules sometimes write $\mathbf{C}_{(\phi)} [\![G]\!]$ to mean that ϕ is optional.
  - That is that the rewrite happens to both continuations with and without a context ϕ.

- Contexts are subsets of the database whose rules obey some property.
- Used to filter rules down to the set not yet tried.

- The algorithm we'll discuss later assumes a few primitives.
- The most general of these are

builtin (X)	is true if `X` is a builtin goal of the language.
var (X)	is true if `X` is a variable.
suffix (X, Y)	is true if `Y` unifies with a suffix of `X`.
fresh (X)	returns a copy of `X` with all variables new.
unify (X, Y)	returns the unifier substitution of X and Y.
resolve (X, ϕ)	resolves a goal `X` in context ϕ.

- Resolution returns:
  - The resolvent (that is, an additional list of goals).
  - The substitution required to continue.
  - The remaining context.

- Also assume the exitence of primitives for manipulating the program.
- addclause (X, Π) returns a new program obtained by adding clause X at the end of program Π.
- delfirstclause (X, Π) retuns a new program obtained by removing the first instance of rule X from Π.

- $X\sigma$ means "apply σ to term X."
- $\sigma_1\sigma_2$ composes left to right.
- $\sigma\mathbf{u}$ is equivalent to ϵ.

- We start the derivation of G_0 in Π with the string

$$\mathbf{P}\,[\![\Pi]\!]\,\mathbf{D}\,[\![G_0]\!]\,\bullet$$

- Unless important, frequently omit $\mathbf{P}\,[\![\Pi]\!]$.
- A successful derivation looks like

$$\mathbf{P}\,[\![\Pi]\!]\,\sigma_1\sigma_2\cdots\sigma_n\mathbf{C}_{\phi_{n-1}}\,[\![G_{n-1}]\!]\cdots\mathbf{C}_{\phi_1}\,[\![G_1]\!]\,\mathbf{C}_{\phi_0}\,[\![G_0]\!]$$

- Derivations with answers, or failure, are *stable*
  - they do not rewrite further

- The strings generated here will tend to look like this:

$$\underbrace{\mathbf{P}\,[\![\Pi]\!]}_{\text{program}}\ \underbrace{\sigma_1\epsilon\mathbf{u}\sigma_2\cdots\sigma_n\mathbf{u}}_{\text{substitutions}}\ \underbrace{\mathbf{D}_{\phi_k}\,[\![G_k]\!]}_{\text{derivation}}\ \underbrace{\mathbf{C}_{\phi_{k-1}}\,[\![G_{k-1}]\!]\cdots\mathbf{C}_{\phi_0}\,[\![G_0]\!]}_{\text{continuations}}\ \underbrace{\bullet}_{\text{end}}$$

- The rewrite rules will all be global substitutions.
  - But there will be only one place they can apply.
  - For example, any rule of the form $\sigma\mathbf{D}_{\phi}\,[\![Y]\!] \Rightarrow \cdots$ can only apply at the interface of the substitutions and the derivation.
  - Similarly, $\sigma\mathbf{C}_{\phi}\,[\![Y]\!] \Rightarrow \cdots$ can only apply when there is no derivation.

- Given a stable state S, define ExtractAnswer(S) as the exhaustive application of

$$\sigma\mathbf{u} \Rightarrow \wedge, \quad \mathbf{C}_{(\phi)}\,[\![X]\!] \Rightarrow \wedge$$

- That is:
  - erase all cancelled substitutions (backtracking).
  - erase all continuations (remaining options).

- Given a stable state S, define StartBacktracking(S) as the rewrite

$$\sigma \mathbf{C}_{(\phi)} [\![X]\!] \Rightarrow \sigma \mathbf{u} \mathbf{C}_{(\phi)} [\![X]\!]$$

- That is, undo the last substitution
  - This can only apply at the unique interface of substitutions and continuations.
- This will now rewrite and will return subsequent answers.

- Three rules for user predicates.
- First, contextualization:

$$\mathbf{D}\left[\!\left[X, \vec{Y}\right]\!\right] \Rightarrow \mathbf{D}_{\phi}\left[\!\left[X, \vec{Y}\right]\!\right]$$

  Where
  - Requires X not a builtin predicate.
  - ϕ is the definition of the predicate X.

- Three rules for user predicates.
- Second, resolution:

$$\mathbf{D}_{\phi} \left[\!\left[X, \vec{Y} \right]\!\right] \Rightarrow \sigma \mathbf{D} \left[\!\left[(R, \vec{Y})\sigma \right]\!\right] \mathbf{C}_{\phi'} \left[\!\left[X, \vec{Y} \right]\!\right]$$

  Where
  - Resolving X in context Φ yielded
    - a substitution σ,
    - a resolvent R,
    - a remaining context Φ'.

- Three rules for user predicates.
- Third, the possibility of failure:

$$\mathbf{D}_{\phi}\left[\!\left[X, \vec{Y}\right]\!\right] \Rightarrow \mathbf{u}$$

Where

- Resolving X in context Φ failed (yielded $\mathbf{u}$ for the substitution).

- Backtracking happens when there is no derivation and the last substitution is **u**.
  - That is, the derivation has failed.
- No options left (backtrack further) :

$$\mathbf{uC}_{\emptyset} [\![X]\!] \Rightarrow \mathbf{uu}$$

- No context (start fresh derivation of X):

$$\mathbf{uC} [\![X]\!] \Rightarrow \mathbf{uD} [\![X]\!]$$

- Try again:

$$\mathbf{uC}_{\phi} [\![X]\!] \Rightarrow \mathbf{uD}_{\phi} [\![X]\!]$$

- Easy to take care of some simple builtins of Prolog.
- Can also raise errors on variable goals, as with Prolog.
- Let **Halt** and **Error** be strings that do not rewrite.

`true`	$\mathbf{D}\left[\!\!\left[\text{true}, \vec{Y}\right]\!\!\right] \Rightarrow \mathbf{D}\left[\!\!\left[\vec{Y}\right]\!\!\right]$		"no operation"
`fail`	$\mathbf{D}\left[\!\!\left[\text{fail}, \vec{Y}\right]\!\!\right] \Rightarrow \mathbf{u}$		cf. StartBacktracking.
`halt`	$\mathbf{D}\left[\!\!\left[\text{halt}, \vec{Y}\right]\!\!\right] \Rightarrow \mathbf{Halt}$		
	$\mathbf{D}\left[\!\!\left[X, \vec{Y}\right]\!\!\right] \Rightarrow \mathbf{Error}$		if X is a variable.

- Empty derivations vanish: $\mathbf{D}[\![\wedge]\!] \Rightarrow \wedge$.

- At this point, we can give an example of a simple program (cf. Section 4.1).

```
p(1) :- p(2), p(3).   %  K1
p(2) :- p(4).         %  K2
p(4).                 %  K3
```

- K_i are clause labels (for contexts).
- Our toplevel goal will be `p(X)`.

- ```
  p(1) :- p(2), p(3).   %  K1
  p(2) :- p(4).         %  K2
  p(4).                 %  K3
  ```
- Contextualize the derivation:

$$
\begin{aligned}
&\mathbf{D}\,[\![p(X)]\!]\bullet \\
&\Rightarrow \mathbf{D}_{\{K_1,K_2,K_3\}}\,[\![p(X)]\!]\bullet
\end{aligned}
$$
  ```

- ```
  p(1) :- p(2), p(3).   %  K1
  p(2) :- p(4).         %  K2
  p(4).                 %  K3
  ```
- Apply K_1:

$$\mathbf{D}\,[\![p(X)]\!]\,\bullet$$
$$\Rightarrow \mathbf{D}_{\{K_1,K_2,K_3\}}\,[\![p(X)]\!]\,\bullet$$
$$\Rightarrow [1/X]\,\mathbf{D}\,[\![p(2),p(3)]\!]\,\mathbf{C}_{\{K_2,K_3\}}\,[\![p(X)]\!]\,\bullet$$
  ```

- ```
  p(1) :- p(2), p(3).   %  K1
  p(2) :- p(4).         %  K2
  p(4).                 %  K3
  ```
- Contextualize and apply K_2:

$$
\begin{aligned}
&\mathbf{D}\,[\![p(X)]\!]\,\bullet \\
&\Rightarrow^{*}\ [1/X]\,\mathbf{D}\,[\![p(2),p(3)]\!]\,\mathbf{C}_{\{K_2,K_3\}}\,[\![p(X)]\!]\,\bullet \\
&\Rightarrow\ [1/X]\,\mathbf{D}_{\{K_1,K_2,K_3\}}\,[\![p(2),p(3)]\!]\,\mathbf{C}_{\{K_2,K_3\}}\,[\![p(X)]\!]\,\bullet \\
&\Rightarrow\ [1/X]\,\epsilon\mathbf{D}\,[\![p(4),p(3)]\!]\,\mathbf{C}_{\{K_3\}}\,[\![p(2),p(3)]\!]\,\mathbf{C}_{\{K_2,K_3\}}\,[\![p(X)]\!]\,\bullet
\end{aligned}
$$
  ```

- ```
  p(1) :- p(2), p(3).   %  K1
  p(2) :- p(4).         %  K2
  p(4).                 %  K3
  ```
- Contextualize and apply K_3:

$$\mathbf{D}\,[\![p(X)]\!]\,\bullet$$
$$\Rightarrow^{*}\ [1/X]\,\epsilon\mathbf{D}\,[\![p(4),p(3)]\!]\,\mathbf{C}_{\{K_3\}}\,[\![p(2),p(3)]\!]\,\mathbf{C}_{\{K_2,K_3\}}\,[\![p(X)]\!]\,\bullet$$
$$\Rightarrow\ [1/X]\,\epsilon\mathbf{D}_{\{K_1,K_2,K_3\}}\,[\![p(4),p(3)]\!]\,\mathbf{C}_{\{K_3\}}\,[\![p(2),p(3)]\!]\ldots$$
$$\Rightarrow\ [1/X]\,\epsilon\epsilon\mathbf{D}\,[\![\mathit{true},p(3)]\!]\,\mathbf{C}_{\emptyset}\,[\![p(4),p(3)]\!]\,\mathbf{C}_{\{K_3\}}\,[\![p(2),p(3)]\!]\ldots$$
  ```

- ```
  p(1) :- p(2), p(3).   %  K1
  p(2) :- p(4).         %  K2
  p(4).                 %  K3
  ```
- Apply the rule for true:

$$
\begin{aligned}
&\mathbf{D}\,[\![p(X)]\!]\,\bullet \\
&\Rightarrow^*\ [1/X]\,\epsilon\epsilon\mathbf{D}\,[\![\text{true}, p(3)]\!]\,\mathbf{C}_{\emptyset}\,[\![p(4), p(3)]\!]\,\mathbf{C}_{\{K_3\}}\,[\![p(2), p(3)]\!]\ldots \\
&\Rightarrow\ [1/X]\,\epsilon\epsilon\mathbf{D}\,[\![p(3)]\!]\,\mathbf{C}_{\emptyset}\,[\![p(4), p(3)]\!]\,\mathbf{C}_{\{K_3\}}\,[\![p(2), p(3)]\!]\ldots
\end{aligned}
$$
  ```

- ```
  p(1) :- p(2), p(3).   % K1
  p(2) :- p(4).         % K2
  p(4).                 % K3
  ```
- Contextualize and fail (no derivation of $p(3)$):

$$\mathbf{D}\,[\![p(X)]\!]\,\bullet$$
$$\Rightarrow^*\ [1/X]\,\epsilon\epsilon\mathbf{D}\,[\![p(3)]\!]\,\mathbf{C}_\emptyset\,[\![p(4),p(3)]\!]\,\mathbf{C}_{\{K_3\}}\,[\![p(2),p(3)]\!]\ldots$$
$$\Rightarrow\ [1/X]\,\epsilon\epsilon\mathbf{D}_{\{K_1,K_2,K_3\}}\,[\![p(3)]\!]\,\mathbf{C}_\emptyset\,[\![p(4),p(3)]\!]\ldots$$
$$\Rightarrow\ [1/X]\,\epsilon\epsilon\mathbf{u}\mathbf{C}_\emptyset\,[\![p(4),p(3)]\!]\ldots$$
  ```

- ```
  p(1) :- p(2), p(3).   %  K1
  p(2) :- p(4).         %  K2
  p(4).                 %  K3
  ```
- Backtrack, fail, and backtrack again:

$$\begin{aligned}
&\mathbf{D}\,[\![p(X)]\!]\,\bullet \\
&\Rightarrow^{*}\ [1/X]\,\epsilon\epsilon\mathbf{u}\mathbf{C}_{\emptyset}\,[\![p(4),p(3)]\!]\,\mathbf{C}_{\{K_3\}}\,[\![p(2),p(3)]\!]\,\mathbf{C}_{\{K_2,K_3\}}\,[\![p(X)]\!]\,\bullet \\
&\Rightarrow\ [1/X]\,\epsilon\epsilon\mathbf{uu}\mathbf{C}_{\{K_3\}}\,[\![p(2),p(3)]\!]\,\mathbf{C}_{\{K_2,K_3\}}\,[\![p(X)]\!]\,\bullet \\
&\Rightarrow\ [1/X]\,\epsilon\epsilon\mathbf{uu}\mathbf{D}_{\{K_3\}}\,[\![p(2),p(3)]\!]\,\mathbf{C}_{\{K_2,K_3\}}\,[\![p(X)]\!]\,\bullet \\
&\Rightarrow\ [1/X]\,\epsilon\epsilon\mathbf{uuu}\mathbf{C}_{\{K_2,K_3\}}\,[\![p(X)]\!]\,\bullet \\
&\Rightarrow\ [1/X]\,\epsilon\epsilon\mathbf{uuu}\mathbf{D}_{\{K_2,K_3\}}\,[\![p(X)]\!]\,\bullet
\end{aligned}$$
  ```

- ```
  p(1) :- p(2), p(3).   %  K1
  p(2) :- p(4).         %  K2
  p(4).                 %  K3
  ```
- Invoke K_2 (from top-level goal, having failed K_1):

$$
\begin{aligned}
&\mathbf{D}\,[\![p(X)]\!]\bullet \\
&\Rightarrow^{*}\ [1/X]\,\epsilon\epsilon\mathbf{uuuD}_{\{K_2,K_3\}}\,[\![p(X)]\!]\bullet \\
&\Rightarrow\ \ [1/X]\,\epsilon\epsilon\mathbf{uuu}\,[2/X]\,\mathbf{D}\,[\![p(4)]\!]\,\mathbf{C}_{\{K_3\}}\,[\![p(X)]\!]\bullet
\end{aligned}
$$
  ```

- ```
  p(1) :- p(2), p(3).   %  K1
  p(2) :- p(4).         %  K2
  p(4).                 %  K3
  ```
- Contextualize and apply K_3 and rule for true:

$$\mathbf{D}\,[\![p(X)]\!]\,\bullet$$
$$\Rightarrow^*\ [1/X]\,\epsilon\epsilon\mathbf{uuu}\,[2/X]\,\mathbf{D}\,[\![p(4)]\!]\,\mathbf{C}_{\{K_3\}}\,[\![p(X)]\!]\,\bullet$$
$$\Rightarrow\ [1/X]\,\epsilon\epsilon\mathbf{uuu}\,[2/X]\,\mathbf{D}_{\{K_1,K_2,K_3\}}\,[\![p(4)]\!]\,\mathbf{C}_{\{K_3\}}\,[\![p(X)]\!]\,\bullet$$
$$\Rightarrow\ [1/X]\,\epsilon\epsilon\mathbf{uuu}\,[2/X]\,\epsilon\mathbf{D}\,[\![\text{true}]\!]\,\mathbf{C}_{\emptyset}\,[\![p(4)]\!]\,\mathbf{C}_{\{K_3\}}\,[\![p(X)]\!]\,\bullet$$
$$\Rightarrow\ [1/X]\,\epsilon\epsilon\mathbf{uuu}\,[2/X]\,\epsilon\mathbf{C}_{\emptyset}\,[\![p(4)]\!]\,\mathbf{C}_{\{K_3\}}\,[\![p(X)]\!]\,\bullet$$

- This does not step. It therefore represents an answer.
  ```

- ```
  p(1) :- p(2), p(3).   %  K1
  p(2) :- p(4).         %  K2
  p(4).                 %  K3
  ```
- So let's extract it!

$$\mathbf{D}\,[\![p(X)]\!]\,\bullet \Rightarrow^* [1/X]\,\epsilon\epsilon\mathbf{uuu}\,[2/X]\,\epsilon\mathbf{C}_\emptyset\,[\![p(4)]\!]\,\mathbf{C}_{\{K_3\}}\,[\![p(X)]\!]\,\bullet$$

$$[1/X]\,\epsilon\epsilon\mathbf{uuu}\,[2/X]\,\epsilon\mathbf{C}_\emptyset\,[\![p(4)]\!]\,\mathbf{C}_{\{K_3\}}\,[\![p(X)]\!]\,\bullet$$

$$[1/X]\,\epsilon\epsilon\mathbf{uuu}\,[2/X]\,\epsilon\bullet$$

$$[1/X]\,\epsilon\mathbf{uu}\,[2/X]\,\epsilon\bullet$$

$$[1/X]\,\mathbf{u}\,[2/X]\,\epsilon\bullet$$

$$[2/X]\,\epsilon\bullet$$
  ```

- ```
  p(1) :- p(2), p(3).   %  K1
  p(2) :- p(4).         %  K2
  p(4).                 %  K3
  ```
- Let's get the next answer.

$$\mathbf{D}\,[\![p(X)]\!]\,\bullet$$
$$\Rightarrow^*\ [1/X]\,\epsilon\epsilon\mathbf{uuu}\,[2/X]\,\epsilon\mathbf{C}_\emptyset\,[\![p(4)]\!]\,\mathbf{C}_{\{K_3\}}\,[\![p(X)]\!]\,\bullet$$

$$\text{StartBacktracking}(\cdots)$$
$$=\ [1/X]\,\epsilon\epsilon\mathbf{uuu}\,[2/X]\,\epsilon\mathbf{uC}_\emptyset\,[\![p(4)]\!]\,\mathbf{C}_{\{K_3\}}\,[\![p(X)]\!]\,\bullet$$
  ```

-
  ```
  p(1) :- p(2), p(3).   %  K1
  p(2) :- p(4).         %  K2
  p(4).                 %  K3
  ```
- Backtrack

$$\begin{aligned} & \text{StartBacktracking}(\cdots) \\ = \ & [1/X]\,\epsilon\epsilon\mathbf{uuu}\,[2/X]\,\epsilon\mathbf{u}\mathbf{C}_\emptyset\,[\![p(4)]\!]\,\mathbf{C}_{\{K_3\}}\,[\![p(X)]\!]\bullet \\ \Rightarrow \ & [1/X]\,\epsilon\epsilon\mathbf{uuu}\,[2/X]\,\epsilon\mathbf{uu}\mathbf{C}_{\{K_3\}}\,[\![p(X)]\!]\bullet \\ \Rightarrow \ & [1/X]\,\epsilon\epsilon\mathbf{uuu}\,[2/X]\,\epsilon\mathbf{uu}\mathbf{D}_{\{K_3\}}\,[\![p(X)]\!]\bullet \end{aligned}$$

- ```
  p(1) :- p(2), p(3).   %  K1
  p(2) :- p(4).         %  K2
  p(4).                 %  K3
  ```
- Apply K_3 and the rule for true:

$$\begin{aligned}
&\text{StartBacktracking}(\cdots) \\
&\Rightarrow^* [1/X]\, \epsilon\epsilon\mathbf{uuu}\, [2/X]\, \epsilon\mathbf{uuD}_{\{K_3\}}\, [\![p(X)]\!]\, \bullet \\
&\Rightarrow\ [1/X]\, \epsilon\epsilon\mathbf{uuu}\, [2/X]\, \epsilon\mathbf{uu}\, [4/X]\, \mathbf{D}\, [\![\text{true}]\!]\, \mathbf{C}_\emptyset\, [\![p(X)]\!]\, \bullet \\
&\Rightarrow\ [1/X]\, \epsilon\epsilon\mathbf{uuu}\, [2/X]\, \epsilon\mathbf{uu}\, [4/X]\, \mathbf{C}_\emptyset\, [\![p(X)]\!]\, \bullet
\end{aligned}$$
  ```

- ```
  p(1) :- p(2), p(3).   %   K1
  p(2) :- p(4).         %   K2
  p(4).                 %   K3
  ```
- Extract the second answer:

$$\text{StartBacktracking}(\cdots)$$
$$\Rightarrow^* [1/X]\,\epsilon\epsilon\mathbf{uuu}\,[2/X]\,\epsilon\mathbf{uu}\,[4/X]\,\mathbf{C}_\emptyset\,[\![p(X)]\!]\bullet$$

$$[1/X]\,\epsilon\epsilon\mathbf{uuu}\,[2/X]\,\epsilon\mathbf{uu}\,[4/X]\,\mathbf{C}_\emptyset\,[\![p(X)]\!]\bullet$$
$$[1/X]\,\epsilon\epsilon\mathbf{uuu}\,[2/X]\,\epsilon\mathbf{uu}\,[4/X]\bullet$$
$$[1/X]\,\epsilon\mathbf{uu}\,[2/X]\,\epsilon\mathbf{uu}\,[4/X]\bullet$$
$$[1/X]\,\epsilon\mathbf{uu}\,[2/X]\,\mathbf{u}\,[4/X]\bullet$$
$$[1/X]\,\epsilon\mathbf{uu}\,[4/X]\bullet$$
$$[1/X]\,\mathbf{u}\,[4/X]\bullet$$
$$[4/X]\bullet$$
  ```

- ```
  p(1) :- p(2), p(3).   %   K1
  p(2) :- p(4).         %   K2
  p(4).                 %   K3
  ```
- Backtrack once more?

$$\begin{aligned}&\text{StartBacktracking}(\cdots)\\ &\Rightarrow^* [1/X]\,\epsilon\epsilon\mathbf{uuu}\,[2/X]\,\epsilon\mathbf{uu}\,[4/X]\,\mathbf{C}_\emptyset\,[\![p(X)]\!]\,\bullet\end{aligned}$$

$$\begin{aligned}&\text{StartBacktracking}(\cdots)\\ &= \quad [1/X]\,\epsilon\epsilon\mathbf{uuu}\,[2/X]\,\epsilon\mathbf{uu}\,[4/X]\,\mathbf{uC}_\emptyset\,[\![p(X)]\!]\,\bullet\\ &\Rightarrow \quad [1/X]\,\epsilon\epsilon\mathbf{uuu}\,[2/X]\,\epsilon\mathbf{uu}\,[4/X]\,\mathbf{uu}\bullet\end{aligned}$$

- This reduces to $\mathbf{u}\bullet$. There are no more answers.
  ```

- Unification either
  - fails (triggering backtracking), or
  - provides a substitution which applies to the answer *and* subsequent goals.

$$
\mathbf{D}\left[\!\!\left[(X = Y), \vec{Z}\right]\!\!\right] \Rightarrow \begin{cases} \sigma \mathbf{D}\,[\![Z\sigma]\!]\,\mathbf{C}_{\emptyset}\,[\![(X = Y), Z]\!] & \text{if unify}(X, Y) = \sigma \\ \mathbf{u} & \text{if unify}(X, Y) \text{ failed} \end{cases}
$$

- Suffix criterion:
  - Given that the current goal is $X, \vec{Y}$, the *parent* of X is the leftmost continuation whose argument does not have a suffix unifiable with $X, \vec{Y}$.
- Basic idea:
  - Empty all continuation contexts up until the parent of the cut.
- Example:

$$\mathbf{D}\, [\![!, p(X)]\!]\, \mathbf{C}_{\phi_1}\, [\![q(Y), !, p(Y)]\!]\, \underbrace{\mathbf{C}_{\phi_2}\, [\![a(X)]\!]}_{\text{parent}} \bullet$$

- To do this in rewrite rules, we first want some markers:

Pending (X)	Placeholder marker for rewrite in progress.
Parent (Y)	Rightwards-moving search and replace marker.
Return	Leftwards-moving completion marker.

- The rules are as expected from these definitions:

$$\mathbf{D}\left[\!\left[!,\vec{Y}\right]\!\right] \Rightarrow \mathbf{Pending}\left(\mathbf{D}\left[\!\left[\vec{Y}\right]\!\right]\right)\mathbf{Parent}\left((!,\vec{Y})\right)$$

$$\mathbf{Parent}\,(G)\,\mathbf{C}_{(\phi)}\,[\![L]\!] \Rightarrow \begin{cases}\mathbf{C}_{\emptyset}\,[\![L]\!]\,\mathbf{Parent}\,(G) & \text{if suffix}(L,G)\\ \mathbf{ReturnC}_{\emptyset}\,[\![L]\!] & \text{otherwise}\end{cases}$$

$$\mathbf{Parent}\,(G)\bullet \Rightarrow \mathbf{Return}\bullet$$

$$\mathbf{C}_{\emptyset}\,[\![X]\!]\,\mathbf{Return} \Rightarrow \mathbf{ReturnC}_{\emptyset}\,[\![X]\!]$$

$$\mathbf{Pending}\,(G)\,\mathbf{Return} \Rightarrow G$$

- Actually, this has a bug. Anybody see it?
- What if
  - the program was `a(X) :- q(X), !, p(X)` and
  - the initial goal was `call(a(Y)), !, p(Y)`?
- We would mistakenly conclude that the ! in the `a` rule had no parent!
- We would empty all contexts back to •.
- Failure on the inside of the near cut would cause the entire program to fail.
- (We can fix this by using something like gensyms for !.)

- "Ought" to be as simple as

$$\mathbf{D}\left[\!\left[\mathrm{call}\,(X)\,,\vec{Y}\right]\!\right] \Rightarrow \mathbf{D}\left[\!\left[X,\vec{Y}\right]\!\right]$$

- Except that that is transparent to a cut.
- Use "insulating layer" (empty continuation):

$$\mathbf{D}\left[\!\left[\mathrm{call}\,(X)\,,\vec{Y}\right]\!\right] \Rightarrow \epsilon\mathbf{D}\left[\!\left[X,\vec{Y}\right]\!\right]\mathbf{C}_{\emptyset}\left[\!\left[\mathrm{call}\,(X)\,,\vec{Y}\right]\!\right]$$

- once (G) takes the first possible solution to G and prohibits backtracking into other options.
- It may be thought of as `once(G) :- G, !.`
- As a builtin,

$$\mathbf{D}\left[\!\left[\text{once}\,(X)\,,\,\vec{Y}\right]\!\right] \Rightarrow \mathbf{D}\left[\!\left[\text{call}\,((X,!))\,,\,\vec{Y}\right]\!\right]$$

- Want to ensure that the cut won't interfere with parent of once (X).

- Expanding once more,

$$\mathbf{D}\left[\!\left[\text{once}\,(X)\,,\vec{Y}\right]\!\right] \Rightarrow^2 \epsilon\mathbf{D}\left[\!\left[X, !, \vec{Y}\right]\!\right]\mathbf{C}_\emptyset\left[\!\left[\text{call}\,((X, !))\,,\vec{Y}\right]\!\right]$$

- The cut's parent is the continuation shown.
- Therefore, if we backtrack out of $\vec{Y}$, we will keep going to the parent of the once (X).
  - Rather than just aborting, as we would with

$$\mathbf{D}\left[\!\left[\text{once}\,(X)\,,\vec{Y}\right]\!\right] \stackrel{!}{\Rightarrow} \mathbf{D}\left[\!\left[X, !, \vec{Y}\right]\!\right]$$

- Repeat

$$\mathbf{D}\left[\!\!\left[\text{repeat}, \vec{Y}\right]\!\!\right] \Rightarrow \epsilon\mathbf{D}\left[\!\!\left[\vec{Y}\right]\!\!\right]\mathbf{C}\left[\!\!\left[\text{repeat}, \vec{Y}\right]\!\!\right]$$

- Failure-as-negation:

$$\mathbf{D}\left[\!\!\left[\backslash + (G), \vec{Y}\right]\!\!\right] \Rightarrow \epsilon\mathbf{D}\,[\![\text{call}(G), !, \text{fail}]\!]\,\mathbf{C}\left[\!\!\left[\vec{Y}\right]\!\!\right]$$

- Failure-as-negation isn't so obvious.
- Let's say G is true (succeeds):

$$
\begin{aligned}
&\mathbf{D}\left[\!\!\left[\backslash+(\text{true}),\vec{Y}\right]\!\!\right]\\
\Rightarrow\ &\epsilon\mathbf{D}\,[\![\text{call}(\text{true}),!,\text{fail}]\!]\,\mathbf{C}\left[\!\!\left[\vec{Y}\right]\!\!\right]\\
\Rightarrow\ &\epsilon\epsilon\mathbf{D}\,[\![\text{true},!,\text{fail}]\!]\,\mathbf{C}_\emptyset\,[\![\text{call}(G),!,\text{fail}]\!]\,\mathbf{C}\left[\!\!\left[\vec{Y}\right]\!\!\right]\\
\Rightarrow^*\ &\epsilon\epsilon\mathbf{D}\,[\![!,\text{fail}]\!]\,\mathbf{C}_\emptyset\,[\![\text{call}(G),!,\text{fail}]\!]\,\mathbf{C}\left[\!\!\left[\vec{Y}\right]\!\!\right]\\
\Rightarrow^*\ &\epsilon\epsilon\mathbf{D}\,[\![\text{fail}]\!]\,\mathbf{C}_\emptyset\,[\![\text{call}(G),!,\text{fail}]\!]\,\mathbf{C}_\emptyset\left[\!\!\left[\vec{Y}\right]\!\!\right]\\
\Rightarrow\ &\epsilon\epsilon\mathbf{u}\mathbf{C}_\emptyset\,[\![\text{call}(G),!,\text{fail}]\!]\,\mathbf{C}_\emptyset\left[\!\!\left[\vec{Y}\right]\!\!\right]\\
\Rightarrow\ &\epsilon\epsilon\mathbf{u}\mathbf{u}\mathbf{C}_\emptyset\left[\!\!\left[\vec{Y}\right]\!\!\right]\\
\Rightarrow\ &\epsilon\epsilon\mathbf{u}\mathbf{u}\mathbf{u}
\end{aligned}
$$

- Let's say G is fail:

$$\begin{aligned}
&\mathbf{D}\left[\!\left[\backslash+(\text{fail}), \vec{Y}\right]\!\right] \\
&\Rightarrow\ \epsilon\mathbf{D}\,[\![\text{call}(\text{fail}), !, \text{fail}]\!]\,\mathbf{C}\left[\!\left[\vec{Y}\right]\!\right] \\
&\Rightarrow\ \epsilon\epsilon\mathbf{D}\,[\![\text{fail}, !, \text{fail}]\!]\,\mathbf{C}_{\emptyset}\,[\![\text{call}(G), !, \text{fail}]\!]\,\mathbf{C}\left[\!\left[\vec{Y}\right]\!\right] \\
&\Rightarrow^{*}\ \epsilon\epsilon\mathbf{u}\mathbf{C}_{\emptyset}\,[\![\text{call}(G), !, \text{fail}]\!]\,\mathbf{C}\left[\!\left[\vec{Y}\right]\!\right] \\
&\Rightarrow\ \epsilon\epsilon\mathbf{uu}\mathbf{C}\left[\!\left[\vec{Y}\right]\!\right] \\
&\Rightarrow\ \epsilon\epsilon\mathbf{uu}\mathbf{D}\left[\!\left[\vec{Y}\right]\!\right]
\end{aligned}$$

- Prolog: catch $(Goal, Catcher, Recover)$ and throw $(Ball)$.
- Introduce one more marker, **ThrowAnc** $(\cdot,\cdot)$.
- The rules are similar to those for cut:

$$\begin{aligned}
\mathbf{D}\,[\![\mathrm{catch}\,(G,B,R,,,X)]\!] &\Rightarrow \epsilon\mathbf{D}\,[\![G,X]\!]\,\mathbf{C}\,[\![\mathrm{catch}\,(G,B,R,,,X)]\!]\\
\mathbf{D}\,[\![\mathrm{throw}\,(B)\,,X]\!] &\Rightarrow \mathbf{ThrowAnc}\,(Z,Z)\\
&\quad\ \text{where } Z \text{ is } (\mathrm{throw}\,(B)\,,X)\\
\mathbf{ThrowAnc}\,(T,X)\bullet &\Rightarrow \mathbf{Error}
\end{aligned}$$

$$\mathbf{ThrowAnc}\,(T, X)\,\mathbf{C}_{(\phi)}\,[\![C]\!]$$
$$\Rightarrow \begin{cases} \mathbf{uThrowAnc}\,(T, X) & \text{if suffix}\,(C, X) \\ \mathbf{u}\sigma\mathbf{D}\,[\![(R, Z)\sigma]\!]\,\mathbf{C}_{\emptyset}\,[\![C]\!] & \text{if } \neg\text{suffix}\,(C, X) \\ & \&T = (\text{throw}\,(B)\,, Y) \\ & \&C = (\text{catch}\,(G, B', R)\,, Z) \\ & \&\text{unify}\,(\text{fresh}\,(B), B') = \sigma \\ \mathbf{uThrowAnc}\,(T, C) & \text{otherwise} \end{cases}$$

- The freshening operation here is to ensure that unifications into Ball that have happened so far are undone. (*i.e.* can only pass ground structure up the stack).

- Simplified example from Section 4.2 (which is unnecessarily complex).
- ```
  r(X) :- throw(X).
  p :- catch(true, _, fail), r(q).
  ```
- The goal here will be `catch(p, C, true).` Here goes:

$$
\begin{aligned}
&\mathbf{D}\,[\![\mathrm{catch}\,(p, C, \mathrm{true})]\!]\,\bullet\\
&\Rightarrow^* \epsilon\mathbf{D}\,[\![p]\!]\,\mathbf{C}_\emptyset\,[\![\mathrm{catch}\,(p, C, \mathrm{true})]\!]\,\bullet\\
&\Rightarrow^* \epsilon\epsilon\mathbf{D}\,[\![\mathrm{catch}\,(\mathrm{true}, \_, \mathrm{fail})\,, r(q)]\!]\,\mathbf{C}_\emptyset\,[\![p]\!]\,\mathbf{C}_\emptyset\,[\![\mathrm{catch}\,(p, C, \mathrm{true})]\!]\,\bullet\\
&\Rightarrow^* \epsilon\epsilon\epsilon\mathbf{D}\,[\![\mathrm{true}, r(q)]\!]\,\mathbf{C}\,[\![\mathrm{catch}\,(\mathrm{true}, \_, \mathrm{fail})\,, r(q)]\!]\,\mathbf{C}_\emptyset\,[\![p]\!]\ldots\\
&\Rightarrow^* \epsilon\epsilon\epsilon\mathbf{D}\,[\![r(q)]\!]\,\mathbf{C}\,[\![\mathrm{catch}\,(\mathrm{true}, \_, \mathrm{fail})\,, r(q)]\!]\,\mathbf{C}_\emptyset\,[\![p]\!]\ldots\\
&\Rightarrow^* \epsilon\epsilon\epsilon\mathbf{D}\,[\![\mathrm{throw}\,(q)]\!]\,\mathbf{C}_\emptyset\,[\![r(q)]\!]\,\mathbf{C}\,[\![\mathrm{catch}\,(\mathrm{true}, \_, \mathrm{fail})\,, r(q)]\!]\ldots
\end{aligned}
$$
  ```

- ```
  r(X) :- throw(X).
  p :- catch(true, _, fail), r(q).
  ```
- Empty suffixes do not count, so the default rule matches repeatedly:

  $\mathbf{D}\,[\![\text{catch}\,(p, C, \text{true})]\!]\,\bullet$

  $\Rightarrow^{*}\ \epsilon\epsilon\epsilon\mathbf{D}\,[\![\text{throw}\,(q)]\!]\,\mathbf{C}_{\emptyset}\,[\![r(q)]\!]$
  $\quad\mathbf{C}\,[\![\text{catch}\,(\text{true}, \_, \text{fail})\,, r(q)]\!]\,\mathbf{C}_{\emptyset}\,[\![p]\!]\,\mathbf{C}_{\emptyset}\,[\![\text{catch}\,(p, C, \text{true})]\!]\,\bullet$

  $\Rightarrow\ \epsilon\epsilon\epsilon\mathbf{ThrowAnc}\,((\text{throw}\,(()\,q)), (\text{throw}\,(()\,q)))\,\mathbf{C}_{\emptyset}\,[\![r(q)]\!]$
  $\quad\mathbf{C}\,[\![\text{catch}\,(\text{true}, \_, \text{fail})\,, r(q)]\!]\,\mathbf{C}_{\emptyset}\,[\![p]\!]\,\mathbf{C}_{\emptyset}\,[\![\text{catch}\,(p, C, \text{true})]\!]\,\bullet$

  $\Rightarrow\ \epsilon\epsilon\epsilon\mathbf{uThrowAnc}\,((\text{throw}\,(()\,q)), r(q))$
  $\quad\mathbf{C}\,[\![\text{catch}\,(\text{true}, \_, \text{fail})\,, r(q)]\!]\,\mathbf{C}_{\emptyset}\,[\![p]\!]\,\mathbf{C}_{\emptyset}\,[\![\text{catch}\,(p, C, \text{true})]\!]\,\bullet$

  $\Rightarrow\ \epsilon\epsilon\epsilon\mathbf{uuThrowAnc}\,((\text{throw}\,(()\,q)), \text{catch}\,(\text{true}, \_, \text{fail})\,, r(q))$
  $\quad\mathbf{C}_{\emptyset}\,[\![p]\!]\,\mathbf{C}_{\emptyset}\,[\![\text{catch}\,(p, C, \text{true})]\!]\,\bullet$

  $\Rightarrow\ \epsilon\epsilon\epsilon\mathbf{uuuThrowAnc}\,((\text{throw}\,(()\,q)), p)$
  $\quad\mathbf{C}_{\emptyset}\,[\![\text{catch}\,(p, C, \text{true})]\!]\,\bullet$
  ```

- ```
  r(X) :- throw(X).
  p :- catch(true, _, fail), r(q).
  ```
- Now the catch rule applies:

$$
\begin{aligned}
&\Rightarrow^* \epsilon\epsilon\epsilon\mathbf{uuuThrowAnc}\,((\mathrm{throw}\,(()\,q)), p) \\
&\quad \mathbf{C}_\emptyset\, [\![\mathrm{catch}\,(p, C, \mathrm{true})]\!] \bullet \\
&\Rightarrow \epsilon\epsilon\epsilon\mathbf{uuuu}[q/C]\mathbf{D}\,[\![\mathrm{true}[q/C]]\!]\,\mathbf{C}_\emptyset\,[\![C]\!] \bullet \\
&\Rightarrow \epsilon\epsilon\epsilon\mathbf{uuuu}[q/C]\mathbf{C}_\emptyset\,[\![C]\!] \bullet
\end{aligned}
$$

  - The substitution $[q/C]$ tells us that all went as anticipated
  ```

- Dynamic updates are another of these marker-moving rewrite games.
- Markers:

Pending (X)	Placeholder marker for rewrite in progress.
Update (X)	Carrier of an update to the program store.
Return (X)	Return message from program store.

- Rules for derivation:

$$\mathbf{D}\left[\!\left[\text{asserta}\,(K)\,,\,\vec{Y}\right]\!\right] \Rightarrow \mathbf{Update}\,(\text{asserta}\,(K))\,\mathbf{Pending}\left(\mathbf{D}\left[\!\left[\text{asserta}\,(K)\,,\,\vec{Y}\right]\!\right]\right)$$

$$\mathbf{D}\left[\!\left[\text{retract}\,(K)\,,\,\vec{Y}\right]\!\right] \Rightarrow \mathbf{Update}\,(\text{retract}\,(K))\,\mathbf{Pending}\left(\mathbf{D}\left[\!\left[\text{retract}\,(K)\,,\,\vec{Y}\right]\!\right]\right)$$

- Rules for updates:

$$
\begin{aligned}
&\mathbf{P}\,[\![\Pi]\!]\,\mathbf{Update}\,(\text{asserta}\,(K)) \\
&\Rightarrow \mathbf{P}\,[\![\Pi']\!]\,\mathbf{Return}\,(\epsilon) && \text{if addclause}\,(K,\Pi)=\Pi' \\
&\mathbf{P}\,[\![\Pi]\!]\,\mathbf{Update}\,(\text{retract}\,(K)) \\
&\Rightarrow \mathbf{P}\,[\![\Pi']\!]\,\mathbf{Return}\,(\sigma) && \text{if delfirstclause}\,(K,\Pi)=(\sigma,\Pi') \\
&\mathbf{P}\,[\![\Pi]\!]\,\mathbf{Update}\,(\text{retract}\,(K)) \\
&\Rightarrow \mathbf{P}\,[\![\Pi]\!]\,\mathbf{Return}\,(\mathbf{u}) && \text{if delfirstclause}\,(K,\Pi)\ \text{failed} \\
&X\,\mathbf{Update}\,(H) \\
&\Rightarrow \mathbf{Update}\,(H)\,X && \text{if } X \neq \mathbf{P}\,[\![\Pi]\!]
\end{aligned}
$$

- Rules for returns:

$$\textbf{Return}(\sigma)\,\textbf{Pending}(\textbf{D}\,[\![\text{asserta}(K), X]\!])$$
$$\Rightarrow \textbf{D}\,[\![X]\!]$$
$$\textbf{Return}(\sigma)\,\textbf{Pending}(\textbf{D}\,[\![\text{retract}(K), X]\!])$$
$$\Rightarrow \sigma\textbf{D}\,[\![X\sigma]\!]\,\textbf{C}\,[\![\text{retract}(K), X]\!]$$
$$\textbf{Return}(\textbf{u})\,\textbf{Pending}(\textbf{D}\,[\![\text{retract}(K), X]\!])$$
$$\Rightarrow \textbf{u}$$
$$\textbf{Return}(\sigma)\,X$$
$$\Rightarrow X\textbf{Return}(\sigma) \qquad \text{if } X \neq \textbf{Pending}(\cdot)$$

- A twist on the marker game which leaves markers in the derivation!
- Markers:

Mark (X)	Leave answer that X.
SetMark (X)	Metacommand to produce a mark.
Collect (X)	Label inside continuations.
SweepMarks (X)	Left-moving marker to collect answers.
Return (L)	Right-moving marker with answer list.

- Core rules:

$$\mathbf{D}\left[\!\!\left[\text{findall}\,(X, G, L)\,, \vec{Y}\right]\!\!\right] \Rightarrow \mathbf{StartMark}\epsilon\mathbf{D}\,[\![\text{call}\,(G)\,, \mathbf{SetMark}\,(X)\,, \text{fail}]\!]\,\mathbf{C}\left[\!\!\left[\mathbf{Collect}\,(\text{findall}\,(X, G, L))\,, \vec{Y}\right]\!\!\right]$$

$$\mathbf{D}\left[\!\!\left[\mathbf{SetMark}\,(X)\,, \vec{Y}\right]\!\!\right] \Rightarrow \mathbf{D}\left[\!\!\left[\vec{Y}\right]\!\!\right] \qquad \text{where } X_1 = \text{fresh}\,(X)$$

$$\mathbf{D}\left[\!\!\left[\mathbf{Collect}\,(\text{findall}\,(X, G, L))\,, \vec{Y}\right]\!\!\right] \Rightarrow \mathbf{SweepMarks}\,([])\,\mathbf{Pending}\left(\mathbf{D}\left[\!\!\left[\text{findall}\,(X, G, L)\,, \vec{Y}\right]\!\!\right]\right)$$

$$\mathbf{SweepMarks}\,(L) \Rightarrow \mathbf{SweepMarks}\,([X|L])$$

$$\mathbf{StartMarkSweepMarks}\,(L) \Rightarrow \mathbf{Return}\,(L)$$

$$\mathbf{Return}\,(L1) \Rightarrow \begin{cases} \sigma\mathbf{D}\left[\!\!\left[\vec{Y}\sigma\right]\!\!\right]\,\mathbf{C}_{\emptyset}\left[\!\!\left[\text{findall}\,(X, G, L)\,, \vec{Y}\right]\!\!\right] & \text{if unify}\,(L, L1) = \sigma \\ \mathbf{u} & \text{if unify}\,(L, L1)\ \text{failed} \end{cases}$$

- Disjunction would be easy, except for the transparency of cuts internal.
- The solution is using a disolved syntax as markers.
  - The paper creatively uses markers '(', ';', and ')'.
- Rules:

$$\mathbf{D}\left[\!\left[(, X, ;, Y,), \vec{R} \right]\!\right] \Rightarrow \epsilon \mathbf{D}\left[\!\left[X, ;, Y,), \vec{R} \right]\!\right] \mathbf{C}\left[\!\left[;, X, ;, Y,), \vec{R} \right]\!\right]$$

$$\mathbf{D}\left[\!\left[;, Y, \vec{R} \right]\!\right] \Rightarrow \mathbf{D}\left[\!\left[ ), \vec{R} \right]\!\right]$$

$$\mathbf{uC}\left[\!\left[;, X, ;, Y,), \vec{R} \right]\!\right] \Rightarrow \mathbf{D}\left[\!\left[ ), \vec{R} \right]\!\right]$$

$$\mathbf{D}\left[\!\left[ ), \vec{R} \right]\!\right] \Rightarrow \mathbf{D}\left[\!\left[\vec{R} \right]\!\right]$$

- (For those following along, this is Section 4.4.)
- A popular idiom in Prolog is the repeat/cut/fail loop:

```
q :- repeat, p(X), (X = b, !; fail).
p(a).
p(b).
p(c).
```

- What this program ought to do is iterate over `p(X)` until `X = b`, then abort.
  - It will never consider `p(c)`.
- Let's get this started using the basic rule:

$$\mathbf{D}\,[\![q]\!] \Rightarrow\ \epsilon\mathbf{D}\,[\![\text{repeat}, p(X), (X = b, !; \text{fail})]\!]\,\mathbf{C}_{\emptyset}\,[\![q]\!]$$

- ```
  q :- repeat, p(X), (X = b, !; fail).
  p(a).
  p(b).
  p(c).
  ```
- Apply the rule for repeat, then find $p(A)$:

$$
\begin{aligned}
&\mathbf{D}\,[\![q]\!] \\
&\Rightarrow\ \epsilon\mathbf{D}\,[\![\text{repeat}, p(X), (X = b, !; \text{fail})]\!]\,\mathbf{C}_{\emptyset}\,[\![q]\!] \\
&\Rightarrow\ \epsilon\epsilon\mathbf{D}\,[\![p(X), (X = b, !; \text{fail})]\!]\,\mathbf{C}\,[\![\text{repeat}, \ldots]\!]\,\mathbf{C}_{\emptyset}\,[\![q]\!] \\
&\Rightarrow\ \epsilon\epsilon[a/X]\mathbf{D}\,[\![\mathit{true}, (a = b, !; \text{fail})]\!]\,\mathbf{C}_{\phi}\,[\![p(X), (X = b, !; \text{fail})]\!] \ldots \\
&\Rightarrow\ \epsilon\epsilon[a/X]\mathbf{D}\,[\![(a = b, !; \text{fail})]\!]\,\mathbf{C}_{\phi}\,[\![p(X), (X = b, !; \text{fail})]\!] \ldots
\end{aligned}
$$
  ```

- ```
  q :- repeat, p(X), (X = b, !; fail).
  p(a).
  p(b).
  p(c).
  ```
- Apply the rule for '(' then backtrack because $a \neq b$:

$$
\begin{aligned}
&\mathbf{D}\,[\![q]\!] \\
&\Rightarrow^{*} \epsilon\epsilon[a/X]\mathbf{D}\,[\![(a = b, !; \mathrm{fail})]\!]\,\mathbf{C}_{\phi}\,[\![p(X), (X = b, !; \mathrm{fail})]\!] \ldots \\
&\Rightarrow\ \epsilon\epsilon[a/X]\epsilon\mathbf{D}\,[\![a = b, !; \mathrm{fail})]\!]\,\mathbf{C}\,[\![; a = b, !; \mathrm{fail})]\!] \ldots \\
&\Rightarrow\ \epsilon\epsilon[a/X]\epsilon\mathbf{uC}\,[\![; a = b, !; \mathrm{fail})]\!]\,\mathbf{C}_{\phi}\,[\![p(X), (X = b, !; \mathrm{fail})]\!] \ldots \\
&\Rightarrow\ \epsilon\epsilon[a/X]\epsilon\mathbf{uD}\,[\![\mathrm{fail})]\!]\,\mathbf{C}_{\phi}\,[\![p(X), (X = b, !; \mathrm{fail})]\!] \ldots \\
&\Rightarrow^{2} \epsilon\epsilon[a/X]\epsilon\mathbf{uuD}_{\phi}\,[\![p(X), (X = b, !; \mathrm{fail})]\!]\,\mathbf{C}\,[\![\mathrm{repeat}, \ldots]\!]\,\mathbf{C}_{\emptyset}\,[\![q]\!]
\end{aligned}
$$
  ```

- ```
  q :- repeat, p(X), (X = b, !; fail).
  p(a).
  p(b).
  p(c).
  ```
- Find $p(b)$, apply rule for '(', unification succeeds:

$$
\begin{aligned}
&\mathbf{D}\,[\![q]\!] \\
&\Rightarrow^{*}\ \epsilon\epsilon[a/X]\epsilon\mathbf{uuD}_{\phi}\,[\![p(X),(X=b,!;\mathrm{fail})]\!]\,\mathbf{C}\,[\![\mathrm{repeat},\ldots]\!]\,\mathbf{C}_{\emptyset}\,[\![q]\!] \\
&\Rightarrow\ \epsilon\epsilon[a/X]\epsilon\mathbf{uu}[b/X]\mathbf{D}\,[\![(b=b,!;\mathrm{fail})]\!]\,\mathbf{C}_{\emptyset}\,[\![p(X),(X=b,!;\mathrm{fail})]\!]\ldots \\
&\Rightarrow\ \epsilon\epsilon[a/X]\epsilon\mathbf{uu}[b/X]\mathbf{D}\,[\![(b=b,!;\mathrm{fail})]\!]\,\mathbf{C}_{\emptyset}\,[\![p(X),(X=b,!;\mathrm{fail})]\!]\ldots \\
&\Rightarrow\ \epsilon\epsilon[a/X]\epsilon\mathbf{uu}[b/X]\epsilon\mathbf{D}\,[\![b=b,!;\mathrm{fail})]\!]\,\mathbf{C}\,[\![b=b,!;\mathrm{fail})]\!]\ldots \\
&\Rightarrow\ \epsilon\epsilon[a/X]\epsilon\mathbf{uu}[b/X]\epsilon\epsilon\mathbf{D}\,[\![!;\mathrm{fail})]\!]\,\mathbf{C}\,[\![b=b,!;\mathrm{fail})]\!]\ldots
\end{aligned}
$$
  ```

- ```
  q :- repeat, p(X), (X = b, !; fail).
  p(a).
  p(b).
  p(c).
  ```
- Apply the rule for cut; see how high it goes:

$$
\begin{aligned}
&\mathbf{D}\,[\![q]\!] \\
&\Rightarrow^* \epsilon\epsilon[a/X]\epsilon\mathbf{uu}[b/X]\epsilon\epsilon\mathbf{D}\,[\![!;\mathrm{fail})]\!]\,\mathbf{C}\,[\![b = b, !; \mathrm{fail})]\!] \\
&\qquad \mathbf{C}_\emptyset\,[\![p(X), (X = b, !; \mathrm{fail})]\!]\,\mathbf{C}\,[\![\mathrm{repeat}, \ldots]\!]\,\mathbf{C}_\emptyset\,[\![q]\!] \\
&\Rightarrow^* \epsilon\epsilon[a/X]\epsilon\mathbf{uu}[b/X]\epsilon\epsilon\mathbf{D}\,[\![;\mathrm{fail})]\!]\,\mathbf{C}_\emptyset\,[\![b = b, !; \mathrm{fail})]\!] \\
&\qquad \mathbf{C}_\emptyset\,[\![p(X), (X = b, !; \mathrm{fail})]\!]\,\mathbf{C}_\emptyset\,[\![\mathrm{repeat}, \ldots]\!]\,\mathbf{C}_\emptyset\,[\![q]\!] \\
&\Rightarrow^2 \epsilon\epsilon[a/X]\epsilon\mathbf{uu}[b/X]\epsilon\epsilon\mathbf{C}_\emptyset\,[\![b = b, !; \mathrm{fail})]\!] \\
&\qquad \mathbf{C}_\emptyset\,[\![p(X), (X = b, !; \mathrm{fail})]\!]\,\mathbf{C}_\emptyset\,[\![\mathrm{repeat}, \ldots]\!]\,\mathbf{C}_\emptyset\,[\![q]\!]
\end{aligned}
$$
  ```

- Mostly to show we can; nothing new here.
- More markers (just two new ones):

Pending (X)	Placeholder marker for rewrite in progress.
Return	Leftwards-moving completion marker.
ThisBranch (P)	Marker of conditional branch in progress.
The (R, O)	Rightwards-moving marker for C-rewriting.

- Uses exploded syntax of disjunction; has higher precendence than previous disjunction rules to deal with If/Then.
- The rules are as expected:

$$\mathbf{D}\left[\!\left[(,\ \mathit{If}\ \rightarrow\ \mathit{Then},\ ;\ ,\ \mathit{Else},\ ),\ \vec{Y}\right]\!\right] \Rightarrow \mathbf{D}\left[\!\left[(,\ \text{once}\,(\mathit{If})\,,\ \mathbf{ThisBranch}\,(\text{once}\,(\mathit{If}))\,,\ \mathit{Then},\ ;\ ,\ \mathit{Else},\ ),\ \vec{Y}\right]\!\right]$$

$$\mathbf{D}\left[\!\left[\mathbf{ThisBranch}\,(()\ \mathit{Pre}),\ \vec{Y}\right]\!\right] \Rightarrow \mathbf{Pending}\left(\mathbf{D}\left[\!\left[\vec{Y}\right]\!\right]\right) \mathbf{The}\left(\mathit{Pre},\ \vec{Y}\right)$$

$$\mathbf{The}\,(\mathit{Pre},\ \mathit{Post})\ \mathbf{C}\left[\!\left[;\ ,\ \vec{Y}\right]\!\right] \Rightarrow \mathit{Return}\mathbf{C}_{\emptyset}\left[\!\left[;\ ,\ \vec{Y}\right]\!\right] \qquad \text{if append}(\mathit{Pre},\ [\mathbf{ThisBranch}\,(\mathit{Pre})\,|\mathit{Post}]) = Y$$

$$\mathbf{The}\,(\mathit{Pre},\ \mathit{Post})\ \mathbf{C}\left[\!\left[\vec{Y}\right]\!\right] \Rightarrow \mathbf{C}\left[\!\left[\vec{Y}\right]\!\right] \mathbf{The}\,(\mathit{Pre},\ \mathit{Post}) \qquad \text{otherwise}$$