χ[G] denotes the expression obtained by substituting a goal G for λ in χ. χ may be referred to as the context of (this occurrence of) the goal G in χ[G]. Correspondingly, the expression $\chi[\chi']$ denotes the context obtained by substituting a context χ' for λ in χ. When λ occurs in a context χ of the form χ'[**and**(R, λ, S)$^{\sigma}_{V}$] or χ'[**or**(R, λ, S)], χ[] denotes the expression obtained by deleting λ.

The *environment* of a context, env(χ), is the conjunction of all constraints in all and-boxes surrounding the hole of χ.

- env(λ) = **true**
- env(**and**(R, χ, S)$^{\sigma}_{V}$) = env(χ) $\wedge$ σ
- env(**or**(R, χ, S)) = env(χ)
- env(**choice**(R, (χ % A), S)) = env(χ)
- env(**aggregate**(*u*, χ, *v*)) = env(χ)

By the environment of an occurrence of a goal G in another goal χ[G], we mean the environment of χ.

A goal G is a *subgoal* of any goal of the form χ[G].

4.4 GOAL TRANSITIONS

An *AKL goal transition system* w.r.t. a given program is a structure $\langle \Delta, \Rightarrow \rangle$, where Δ is the set of expressions generated by $\langle$goal$\rangle$, and $\Rightarrow \subseteq \Delta \times M \times C \times \Delta$ is a labelled transition relation, where M is the *mode* (one of D or N) and C is the set of contexts.

Read

$$G \underset{\chi}{\overset{m}{\Rightarrow}} G'$$

as saying that there is a transition from G to G' in mode *m* with context χ. The letters D and N stand for *determinate* and *nondeterminate* mode, respectively.

The rest of this section describes the rules defining the transition relation.

First, the *subgoal* rule

$$\frac{G \underset{\chi'[\chi]}{\overset{m}{\Rightarrow}} G'}{\chi[G] \underset{\chi'}{\overset{m}{\Rightarrow}} \chi[G']}$$

derives transitions of goals depending on transitions that can be made by the components of the goals.

4.4.1 Basic Statements

The *constraint atom* rule

$$\textbf{and}(R, A, S)^{\sigma}_{V} \underset{\chi}{\overset{D}{\Rightarrow}} \textbf{and}(R, S)^{A \wedge \sigma}_{V}$$

moves a constraint atom A to the constraint part of the and-box, thereby making it part of the constraint environment of the goals in the box.

The *program atom* rule

$$A \overset{D}{\underset{\chi}{\Rightarrow}} B$$

unfolds a program atom A using its definition A := B, where the arguments of A are substituted for the formal parameters, and bound variables in B are renamed as appropriate to avoid conflicts with the parameters of A.

The *composition* rule

$$\mathbf{and}(R, (A, B), S)^{\sigma}_{V} \overset{D}{\underset{\chi}{\Rightarrow}} \mathbf{and}(R, A, B, S)^{\sigma}_{V}$$

flattens compositions, making their components parts of the enclosing and-box.

The *hiding* rule

$$\mathbf{and}(R, (U : A), S)^{\sigma}_{W} \overset{D}{\underset{\chi}{\Rightarrow}} \mathbf{and}(R, B, S)^{\sigma}_{V \cup W}$$

introduces new variables. The variables of U in A are replaced by the variables in the set V, giving the new statement B. The set V is chosen to be disjoint from W, from all sets of local variables in R, S, and χ, and from the set of variables bound in A.

The *choice* rule

$$(U_1 : A_1 \,\%\, B_1 ; \cdots ; U_n : A_n \,\%\, B_n) \overset{D}{\underset{\chi}{\Rightarrow}} \mathbf{choice}(\mathbf{and}(A'_1)^{\mathbf{true}}_{V_1} \,\%\, B'_1, \ldots, \mathbf{and}(A'_n)^{\mathbf{true}}_{V_n} \,\%\, B'_n)$$

starts local computations in guards. The local variables U_i in A_i and B_i are replaced by the variables in the set V_i, giving new statements A'_i and B'_i. The sets V_i are chosen to be disjoint from each other, from all other sets of local variables in χ, and from the set of variables bound in A_i or B_i.

4.4.2 Promotion

The *promotion* rule

$$\mathbf{and}(R, \mathbf{choice}(\sigma_V \,\%\, B), S)^{\theta}_{W} \overset{D}{\underset{\chi}{\Rightarrow}} \mathbf{and}(R, B, S)^{\sigma \wedge \theta}_{V \cup W}$$

promotes a single remaining guarded goal with a solved guard and-box. If % is '$\rightarrow$' or '|' it is required that σ is quiet with respect to $\theta \wedge \mathrm{env}(\chi)$ and V.

4.4.3 Pruning

The *condition* rule

$$\mathbf{choice}(R, \sigma_V \rightarrow B, S) \overset{D}{\underset{\chi}{\Rightarrow}} \mathbf{choice}(R, \sigma_V \rightarrow B)$$

may be applied if S is non-empty and σ is quiet with respect to $\mathrm{env}(\chi)$ and V.

The *commit* rule

$$\textbf{choice}(R, \sigma_V \mid B, S) \overset{D}{\underset{\chi}{\Rightarrow}} \textbf{choice}(\sigma_V \mid B)$$

may be applied if at least one of R or S is non-empty and σ is quiet with respect to env(χ) and V.

4.4.4 Failure

The *environment failure* rule

$$\textbf{and}(R)^{\sigma}_{V} \overset{D}{\underset{\chi}{\Rightarrow}} \textbf{fail}$$

fails an and-box if σ and env(χ) are incompatible.

The *goal failure* rule

$$\textbf{and}(R, \textbf{choice}(), S)^{\sigma}_{V} \overset{D}{\underset{\chi}{\Rightarrow}} \textbf{fail}$$

fails an and-box if it contains an empty choice-box.

The *guard failure* rule

$$\textbf{choice}(R, \textbf{fail} \ \% \ B, S) \overset{D}{\underset{\chi}{\Rightarrow}} \textbf{choice}(R, S)$$

removes a failed guarded goal.

4.4.5 Nondeterminism

The *choice splitting* rule

$$\textbf{and}(S_1, \textbf{choice}(T_1, T_2), S_2)^{\sigma}_{V} \overset{N}{\underset{\chi}{\Rightarrow}}$$
$$\textbf{or}(\ \textbf{and}(S_1, \textbf{choice}(T_1), S_2)^{\sigma}_{V}\ ,$$
$$\textbf{and}(S_1, \textbf{choice}(T_2), S_2)^{\sigma}_{V}\ )$$

distributes nondeterminism in an inner nondeterminate choice-box over an and-box, creating alternatives in an outer box. The following two conditions must be satisfied.

- T_1 must be a single guarded goal of the form (θ_W ? A) and T_2 must be non-empty. Choice splitting is said to be performed *with respect to* T_1, and in a goal matching the left hand side of the rule, T_1 is called a *candidate* for choice splitting.
- The rewritten and-box must be a subgoal of a *stable* goal. The notion of stability is explained in Section 4.5.

The *guard distribution* rule

$$\textbf{choice}(R, \textbf{or}(G, S) \ \% \ B, T) \overset{D}{\underset{\chi}{\Rightarrow}} \textbf{choice}(R, G \ \% \ B, \textbf{or}(S) \ \% \ B, T)$$

distributes alternatives in a guard over a guarded goal, making the or-branches different guarded goals in the containing choice-box.

4.4.6 Aggregates

The *aggregate* rule

$$\text{aggregate}(u, \text{A}, v) \underset{\chi}{\overset{\text{D}}{\Rightarrow}} \textbf{aggregate}(w, \textbf{or}(\textbf{and}(\text{B})^{\textbf{true}}_{\{w\}}), v)$$

introduces an aggregate-box. The variable *u* in A is replaced with a variable *w*, giving the new statement B. The variable *w* is chosen to not occur in any set of local variables in the context χ, nor in the set of variables bound in A.

In the following, the expressions *unit*(*v*) and *collect*(*u*', *v*', *v*) stand for statements with one and three free variables, respectively. Nothing precludes having at the same time several different types of aggregates with different associated unit and collect statements, but for simplicity we restrict this exposition to one type of aggregate.

We may rewrite aggregate-boxes by the *unit* rule

$$\textbf{aggregate}(u, \textbf{fail}, v) \underset{\chi}{\overset{\text{D}}{\Rightarrow}} \mathit{unit}(v)$$

and by the *collect* rule

$$\textbf{and}(\text{R}_1, \textbf{aggregate}(u, \textbf{or}(\text{S}_1, \sigma_\text{V}, \text{S}_2), v), \text{R}_2)^{\theta}_{\text{W}} \underset{\chi}{\overset{\text{D}}{\Rightarrow}}$$
$$\textbf{and}(\text{R}_1, \textbf{aggregate}(u, \textbf{or}(\text{S}_1, \text{S}_2), v'), \mathit{collect}(u', v', v), \text{R}_2)^{\sigma' \wedge \theta}_{\text{V}' \cup \text{W}}$$

if σ is quiet w.r.t. $\theta \wedge \text{env}(\chi)$ and V. The local variables V in σ are replaced by the variables in the set V′, giving σ'. The set V' is chosen to be disjoint from W and all sets of local variables in R_1, R_2, and χ. In particular, the variable *u* is replaced by *u*'. If the aggregate is ordered it is also required that S_1 is empty.

The *or-flattening* rule

$$\textbf{or}(\text{R}, \textbf{or}(\text{S}), \text{T}) \underset{\chi}{\overset{\text{D}}{\Rightarrow}} \textbf{or}(\text{R}, \text{S}, \text{T})$$

unnests nested or-boxes, making the alternatives available for the collect rule.

4.4.7 Constraint Simplification

Simplification of constraints is not an essential part of the AKL model, but can be expressed as follows. With a constraint theory may be associated *constraint simplification rules* of the general form

$$\textbf{and}(\text{R})^{\sigma}_{\text{U}} \underset{\chi}{\overset{\text{D}}{\Rightarrow}} \textbf{and}(\text{R})^{\theta}_{\text{V}}$$

that replace [illegible] The set V contains all variables in U that occur in θ or R, and new variables that are in θ, but not in R, U or χ. The rules must satisfy the following condition

$$\textbf{TC} \quad \text{env}(\chi) \supset (\exists \text{W}\sigma \equiv \exists \text{W}\theta)$$

where W contains all variables in U and V not occurring in R, and they may not give rise to infinite sequences of simplifying transitions.

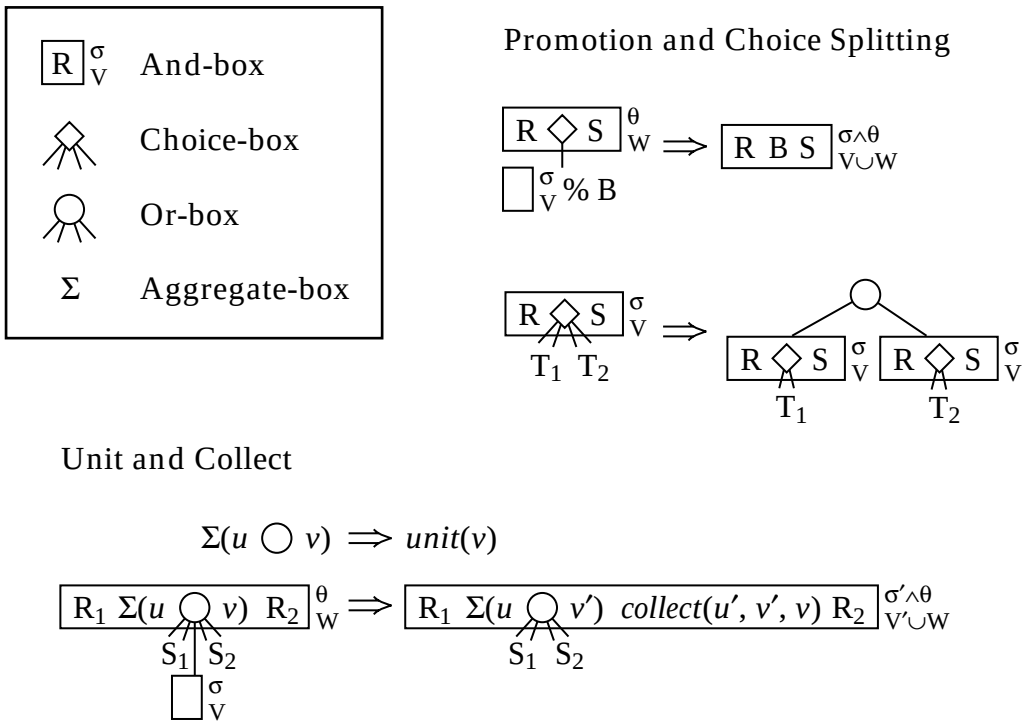

Figure 4.1. Graphical notation for boxes and rules

Simplification rules can be used to justify simplifications performed in an implementation, e.g., during constraint solving and garbage collection.

For the constraint system of rational trees, we have already, informally, introduced simplification by unification. A more precise description follows, for use in Chapter 5.

Simplification is relative, in that it is performed with respect to a given simplified constraint θ [Aït-Kaci, Podelski, and Smolka 1992]. As before, we assume that θ has substitution form, in which all equations of the form $u = v$ together form an acyclic relation over the variables. If we regard this relation over the variables as a partial order, the sets of variables known to be equal have unique maximal elements. Given a variable v, we refer to this as deref(v, θ). Below, u' is deref(u, θ) and v' is deref(v, θ).

We assume that σ∧**true** and **true**∧σ and σ∧**false** and **false**∧σ are always simplified to σ and **false**, respectively. Note below that if a constraint σ is simplified to σ′ relative to θ, then θ∧σ′ has the acyclic substitution form required above.

A constraint of the form $u = \mathrm{f}(v_1, \ldots, v_k)$ is simplified to **false** if u' is bound to a constructor expression with different constructors, to the simplified form of $u_1 = v_1 \wedge \cdots \wedge u_k = v_k$, if it has the same constructor and arguments u_i, and otherwise to $u' = \mathrm{f}(v_1, \ldots, v_k)$.

A constraint $u = v$ is simplified to **true** if u' is equal to v', to **false** if u' and v' are bound to constructor expressions with different constructors, to the simplified

form of $u_1 = v_1 \wedge \cdots \wedge u_k = v_k$ for their arguments u_i and v_i, if they have the same constructor, to $u' = v'$ if u' is not bound, and otherwise to $v' = u'$.

A constraint $\sigma_1 \wedge \sigma_2$ is simplified to $\sigma_1' \wedge \sigma_2'$, where σ_1 is simplified to σ_1' relative to θ and σ_2 is simplified to σ_2' relative to $\theta \wedge \sigma_1'$.

The above can be used for a rule that simplifies a constraint that has just been added by the constraint atom rule (relative to the conjunction of the remaining local store and the environment), and for a rule that simplifies an entire local store when constraints have been added to stores in its environment.

Simplification corresponding to garbage collection involves partitioning local stores σ in substitution form into two parts σ_1 and σ_2, where σ_2 binds variables in W that do not occur in σ_1. Clearly, $\exists W\sigma$ is equivalent to $\exists W\sigma_1$, and σ_2 can be garbage collected.

4.5 NONDETERMINISM AND STABILITY

Choice splitting potentially duplicates work. Therefore, it is delayed until it is necessary for the computation to proceed. Let us refer to the and-box which is to be rewritten by choice splitting as G_N and to its context as χ_N. Conditions that formalise the above "necessity" will be placed on a goal G and its context χ, for which

$$\chi_N = \chi[\chi'] \quad \text{and} \quad G = \chi'[G_N]$$

A first condition is to require that no determinate transitions are possible on G in χ. We say that a goal G is *quiescent in* χ if

$$\neg\exists G'(G \underset{\chi}{\overset{D}{\Rightarrow}} G')$$

However, future transitions in the context of a goal may produce constraints that either entail or are incompatible with constraints within the goal. Therefore, not only should it be required that no determinate transitions are applicable, but also that determinate transitions cannot become applicable as a result of further transitions in the context of the goal. Unfortunately, as stated, this condition is undecidable, and cannot be used for programming purposes.

Instead, we say that a goal is *stable* in a given context if it is quiescent, and remains so even if arbitrary constraints compatible with the environment, are added to the and-boxes in the context of the goal. We thereby avoid reasoning about the constraints that may be produced by future computations, and need only consider the relationship between [illegible] and its environment. It still depends on the constraint theory whether this condition is computationally tractable, but it is well-behaved for rational trees.

Formally, a goal G is stable in context χ if it is quiescent and

$$\forall\chi'\forall\tau\,(G{=}\chi'[] \supset [\mathbf{TC}\ \text{env}(\chi) \wedge \tau \supset \exists V\ \text{env}(\chi')])$$

$$\forall\chi'\forall\tau\,\forall\sigma\forall U\,(G{=}\chi'[\sigma] \supset [\mathbf{TC}\ \text{env}(\chi) \wedge \tau \supset \exists V\,(\text{env}(\chi') \wedge \neg\exists U\sigma)])$$

where V is the set of variables local to and-boxes in G.

Note that the notation χ[include solved and-boxes. The first condition states that environment failure will not become applicable. The second condition states that no solved and-box will become quiet. This is stronger than required, by only considering solved and-boxes for which quietness can cause further transitions. Even stronger, but probably more useful, conditions are

$$\forall\chi' \, (G{=}\chi'[] \supset [\mathbf{TC} \; \mathrm{env}(\chi) \supset \exists V \; \mathrm{env}(\chi')])$$

$$\forall\chi'\forall\sigma\forall U \, (G{=}\chi'[\sigma_U] \supset [\mathbf{TC} \; \mathrm{env}(\chi) \supset \exists V \, (\mathrm{env}(\chi') \wedge \neg\exists U\sigma)])$$

For the constraint theory of rational trees, the first of these conditions implies the second and is also equivalent with the above formulation [Franzén 1994]. If relative simplification is performed as suggested, a "constructive" formulation of this condition is that G should not contain equations of the form $v = f(\ldots)$ or $u = v$, where u and v are not in V.

Additional conditions may, and should, be imposed on the choice splitting rule, for example that it must be applied with respect to the leftmost candidate in an innermost stable box. However, it is not the intention that AKL should be based on a single such rule. Instead, it should be chosen for the given program. The topic of how to make this selection (e.g., using the concept of engines) is outside the scope of this dissertation, where we assume that a *leftmost* candidate be chosen.

4.6 CONFIGURATIONS AND COMPUTATIONS

Two sets of local variables in a goal *interfere* if their intersection is non-empty and either they occur in different goals in an and-box, or one occurs in the and-box of the other. A goal is *well-formed* if no sets of variables interfere.

A well-formed global goal is a *configuration* iff all variables occurring free anywhere in the goal occur in the set of local variables of some and-box in the goal. The letter γ stands for configurations.

OBSERVATION. *Transitions take well-formed goals to well-formed goals, and configurations to configurations.*

That goals are taken to goals is verified by inspection of the rules.

Variables are introduced in the hiding, choice, aggregate, and constraint simplification rules, and then as non-interfering sets of local variables.

Choice splitting duplicates sets of local variables, but not in a way that puts them in different goals in an and-box.

Promotion and the collect rule move variables between sets.

The set of local variables in the and-box promoted by the promotion rule does not interfere with the sets in the and-box to which it is promoted, since they are in different goals in this and-box, nor with the sets in any and-box containing it. Thus, the union will not interfere with any other set.

The set of local variables in the set promoted by the collect rule is renamed to avoid conflicts.

Other rules do not add or move variables. ♦

The *AKL computation model* is a structure $\langle \Delta, \rightarrow \rangle$, where Δ is the set of configurations, and $\rightarrow \subseteq \Delta \times \Delta$ is a transition relation, defined as the subset of goal transitions that are transitions on configurations.

$$\gamma \rightarrow \gamma' \equiv \gamma \overset{m}{\underset{\lambda}{\Rightarrow}} \gamma'$$

A *derivation* is a finite or infinite transition sequence

$$\gamma_0 \rightarrow \gamma_1 \rightarrow \gamma_2 \rightarrow \cdots$$

A configuration γ which satisfies $\neg\exists\gamma'.\ \gamma \rightarrow \gamma'$ is *terminal*.

Initial configurations are of the form

$$\mathbf{or}(\mathbf{and}(\mathrm{A})_{\mathrm{V}}^{\mathbf{true}})$$

where V is the set of variables occurring free in the statement A. The statement A is also referred to as the *query*.

Final configurations are of the form

$$\mathbf{or}(\mathbf{and}()_{\mathrm{V}_1}^{\sigma_1}, \ldots, \mathbf{and}()_{\mathrm{V}_n}^{\sigma_n})$$

in which all and-boxes are solved. A terminal configuration that is not final is *stuck*.

A *computation* is a derivation beginning with an initial configuration and, if finite, ending with a terminal configuration.

The sequence of and-boxes in a final configuration may be empty, in which case we talk of a *failed* computation. Otherwise, the constraints σ_i are referred to as the *answers* of the computation (also referred to as the answers to the query).

4.7 POSSIBLE EXTENSIONS

The computation model presented can be extended in many directions. By adding new guard operators, corresponding to new forms of choice statements, expressiveness is added to AKL with a minimum of effort, for programmers and implementors alike.

In the following, two variants of conditional choice are presented. The first, the logical condition, admits a cleaner logical interpretation. The second, cut, is unclean, but highly useful for executing Prolog programs in the AKL environment.

4.7.1 Logical Conditions

Due to the guard distribution rule, the condition operator of AKL prunes not only other clauses in the original choice, but also alternative solutions of the

guards. This has advantages, e.g., for a metainterpreter and for the implementation, but an operator which does not is subject to a less restrictive logical interpretation, as shown by Franzén [1994].

We introduce the *logical condition* operator $\rightarrow_L$ (called *soft cut* by Franzén) to which the guard distribution rule does not apply. To this operator corresponds a *logical conditional choice* statement.

We also need the notion of *or-component*, which is defined as follows. (1) If G is an and-box, G is an or-component of G. (2) If G is an or-component of G', then G is an or-component of **or**(R, G', S).

The *logical condition* rule

$$\textbf{choice}(G \rightarrow_L B, S) \underset{\chi}{\overset{D}{\Rightarrow}} \textbf{choice}(G \ ?\ B)$$

may be applied if σ_V is an or-component of G and σ is quiet with respect to env(χ) and V.

We may also add a rule

$$\textbf{choice}(R, G \rightarrow_L B, S) \underset{\chi}{\overset{D}{\Rightarrow}} \textbf{choice}(R, G \rightarrow_L B)$$

which prunes alternatives more eagerly if the above conditions hold and R and S are both non-empty.

4.7.2 Cut

Prolog provides the *cut* operation, which is noisy in the sense that pruning is performed without a quietness condition. In Prolog, synchronisation is given by the sequential execution order. Therefore, the state in which cut is applied is well-defined. (However, the noisiness of cut creates problems when *freeze* and similar coroutining constructs are added, as in the SICStus Prolog implementation [Carlsson et al 1993].)

In AKL, an operation very similar to cut can be introduced by adding a noisy form of the condition operator. This operation becomes very unpredictable unless some other form of synchronisation is added. The solution is to regard noisy pruning as a form of nondeterminate action, which is allowed in the same circumstances as choice splitting, i.e., in *stable* goals.

The syntax is augmented with a new guard operator, let us say '!', to which also corresponds a *cut choice* statement. The existing rules will work appropriately together with the following.

The *noisy cut* rule

$$\textbf{choice}(R, \sigma_V\ !\ B, S) \underset{\chi}{\overset{N}{\Rightarrow}} \textbf{choice}(R, \sigma_V\ !\ B)$$

may be applied within a stable goal if S is non-empty, and the *quiet cut* rule

$$\textbf{choice}(R, \sigma_V\ !\ B, S) \underset{\chi}{\overset{D}{\Rightarrow}} \textbf{choice}(R, C_V\ !\ B)$$

may be applied if S is non-empty and σ is quiet with respect to env(χ) and V.

The cut operation is highly useful when translating Prolog into AKL, and this application is discussed in Chapter 8.

4.8 FORMAL ASPECTS

Franzén [1994] presents soundness and completeness results for a logical interpretation of a subset of AKL, and a confluence result for a similar subset of AKL with certain restrictions on the choice splitting rule. These results are summarised here, partly informally, for the completeness of this exposition. Formal definitions and proofs are found in [Franzén 1994].

4.8.1 Logical Interpretation

Definitions and statements are interpreted (by *) as

$$(A := B)^* \Rightarrow A \equiv B^*$$

$$A^* \Rightarrow A \quad \text{(atom A)}$$

$$(A, B)^* \Rightarrow A^* \wedge B^*$$

$$(V : A)^* \Rightarrow \exists V\, A^*$$

$$(\, V_1 : A_1 \,\%\, B_1 \,;\, \cdots \,;\, V_n : A_n \,\%\, B_n \,)^* \Rightarrow$$
$$\exists V_1\, (A_1^* \wedge B_1^*) \vee \cdots \vee \exists V_n\, (A_n^* \wedge B_n^*) \quad (\% \in \{\,|\,, ?\})$$

$$(\, V_1 : A_1 \rightarrow B_1 \,;\, \cdots \,;\, V_n : A_n \rightarrow B_n \,)^* \Rightarrow$$
$$\exists V_1\, (A_1^* \wedge B_1^*) \vee \cdots$$
$$\vee ((\neg\exists V_1\, A_1^*) \wedge \cdots \wedge (\neg\exists V_{n\text{-}1}\, A_{n\text{-}1}^*) \wedge \exists V_n\, (A_n^* \wedge B_n^*))$$

and goals, correspondingly, as

$$\textbf{fail}^* \Rightarrow \textbf{false}$$

$$(\textbf{fail} \,\%\, B)^* \Rightarrow \textbf{false}$$

$$(\textbf{or}(G_1, \ldots, G_n) \,\%\, B)^* \Rightarrow (G_1 \,\%\, B)^* \vee \cdots \vee (G_n \,\%\, B)^* \quad (\% \in \{\,|\,, ?\})$$

$$(\textbf{or}(G_1, \ldots, G_n) \rightarrow B)^* \Rightarrow$$
$$(G_1 \rightarrow B)^* \vee \cdots \vee (\neg G_1^* \wedge \cdots \wedge \neg G_{n\text{-}1}^* \wedge (G_n \rightarrow B)^*)$$

$$\textbf{or}(G_1, \ldots, G_n)^* \Rightarrow G_1^* \vee \cdots \vee G_n^*$$

$$(\textbf{and}(G_1, \ldots, G_n)_V^{\sigma} \,\%\, B)^* \Rightarrow$$
$$\exists(V \setminus U)\, (\sigma \wedge G_1^* \wedge \cdots \wedge G_n^* \wedge B^*)$$

$$(\textbf{and}(G_1, \ldots, G_n)_V^{\sigma})^* \Rightarrow \exists(V \setminus U)\, (\sigma \wedge G_1^* \wedge \cdots \wedge G_n^*)$$

$$\textbf{choice}(G_1, \ldots, G_n)^* \Rightarrow G_1^* \vee \cdots \vee G_n^* \quad (\% \in \{\,|\,, ?\})$$

$$\textbf{choice}(G_1 \rightarrow B_1, \ldots, G_n \rightarrow B_n)^* \Rightarrow$$
$$(G_1 \rightarrow B_1)^* \vee \cdots \vee (\neg G_1^* \wedge \cdots \wedge \neg G_{n\text{-}1}^* \wedge (G_n \rightarrow B_n)^*))$$