Implementing β-Reduction by Hypergraph Rewriting

Sabine Kuske [1]

Fachbereich Mathematik und Informatik
Universität Bremen
D–28334 Bremen, Germany
email: kuske@informatik.uni-bremen.de

Abstract

The aim of this paper is to implement the β-reduction in the λ-calculus with a hypergraph rewriting mechanism called collapsed λ-tree rewriting. It turns out that collapsed λ-tree rewriting is sound with respect to β-reduction and complete with respect to the Gross-Knuth strategy. As a consequence, there exists a normal form for a collapsed λ-tree if and only if there exists a normal form for the represented λ-term.

1 Introduction

The λ-calculus (see [3,13,4]) can be considered as the computational basis for functional programming. Graph reduction for the λ-calculus was studied first in [20] and later in e.g. [17,16,10,1] improving the performance of implementations of functional languages. One main advantage of representing λ-terms by graphs is that common subterms can be shared such that several redexes can be reduced in parallel. Within the well developed theory of graph rewriting (see [5,8,9,7,18] for a survey), hypergraph rewriting was shown to be a suitable formalism for the implementation of term rewriting systems and logic programming (see [12,14,19,6]). The aim of this paper is to show how to implement the β-reduction in the λ-calculus with a hypergraph rewriting mechanism called collapsed λ-tree rewriting.

It is assumed that the reader is familiar with the basic concepts of the λ-calculus. Due to the space restrictions proofs are omitted here; they will appear in the long version of this paper.

[1] This work has been supported by the Deutsche Forschungsgemeinschaft and the ESPRIT Basic Research Working Group 7183, COMPUGRAPH II.

2 Representing λ-terms by collapsed λ-trees

Collapsed λ-trees are acyclic directed hypergraphs with one root where each node represents a λ-term. Before introducing collapsed λ-trees, we recall some definitions concerning hypergraphs.

Let Σ be a set of *labels*. Then a *hypergraph* over Σ is a system $H = (V, E, s, t, l)$ where V is a finite set of *nodes*, E is a finite set of *hyperedges*, $s: E \to V$, $t: E \to V^*$ and $l: E \to \Sigma$ are three mappings assigning to each hyperedge a *source* [2], a sequence of *targets*, and a *label*, respectively. A hyperedge with label X will be called an X-hyperedge. The components of H will also be referred to as V_H, E_H, s_H, t_H, and l_H.

For $v \in V$, $outdegree_H(v)$ denotes the number of hyperedges in H with source v. Given two nodes $v, v' \in V$, a *path from v to v'* is a finite sequence $(\langle e_1, i_1\rangle, \ldots, \langle e_n, i_n\rangle)$ with $e_1, \ldots, e_n \in E$, $i_1, \ldots, i_n \in \mathbb{N}$, $s(e_1) = v$, $t(e_n)|_{i_n} = v'$ [3], and $t(e_j)|_{i_j} = s(e_{j+1})$ for $j = 1, \ldots, n-1$. By convention, each node $v \in V$ is connected to itself by the empty path $()$. H is called *acyclic* if for each $v \in V$ there is no non-empty path from v to v.

A hypergraph $H' = (V', E', s', t', l')$ is a *subhypergraph* of H, denoted by $H' \subseteq H$, if $V' \subseteq V$, $E' \subseteq E$, and s', t' and l' are restrictions of the mappings s,t, and l, respectively. Given two hypergraphs H, H' over Σ, a *hypergraph morphism* $f: H \to H'$ consists of two mappings $f_V : V_H \to V_{H'}$ and $f_E : E_H \to E_{H'}$ that preserve sources, targets and labels, that is, $s_{H'} \circ f_E = f_V \circ s_H$ and $t_{H'} \circ f_E = f_V^* \circ t_H$ (where f_V^* is the natural extension of f_V to sequences), and $l_{H'} \circ f_E = l_H$.

Collapsed λ-trees

Let C be a set of *constants* with $\lambda, A \notin C$. Then an acyclic hypergraph $H = (V, E, s, t, l)$ over $\{\lambda, A\} \cup C$ is a *collapsed λ-tree* if there is a unique node $root_H \in V$ with no incoming hyperedge, and if for all $v \in V$ and all $e \in E$, (1) $outdegree_H(v) \leq 1$, (2) $|t(e)| = 2$ [3] if $l(e) \in \{\lambda, A\}$ and $|t(e)| = 0$ if $l(e) \in C$, (3) $outdegree_H(t(e)|_1) = 0$ if $l(e) = \lambda$, and (4) $s(e)$ is on every path from $root_H$ to $t(e)|_1$ if $l(e) = \lambda$.

Let H be a collapsed λ-tree. Then each $v \in V_H$ represents a λ-term $term_H(v)$ over the variable set $\mathcal{V}_H = \{v \in V_H \mid outdegree_H(v) = 0\}$ and the set C as follows. If $outdegree(v) = 0$ then $term_H(v) = v$. Otherwise, let e be the unique hyperedge with $s_H(e) = v$. Then $term_H(v) = l_H(e)$ if $l_H(e) \in \mathcal{C}$, $term_H(v) = (term_H(t_H(e)|_1) term_H(t_H(e)|_2))$ if $l_H(e) = A$, and $term_H(v) = (\lambda t_H(e)|_1 . term_H(t_H(e)|_2))$ if $l_H(e) = \lambda$. In the following, $term(H)$ stands for $term_H(root_H)$. The two hypergraphs shown in Figure 2 are collapsed λ-trees, representing the λ-terms $(((\lambda v_1.(v_1\, v_2))a)((\lambda v_1.(v_1\, v_2)a)))$ and $((a\, v_2)(a\, v_2))$. It can be shown that each λ-term can be represented by a collapsed λ-tree up to α-conversion. In the following we do not distinguish between λ-terms that are equal up to α-conversion.

[2] Usually, each hyperedge has an arbitrary number of sources; but this is not needed here.
[3] For a sequence w, $w|_i$ denotes its i^{th} element and $|w|$ its length.

3 Collapsed λ-tree rewriting

Collapsed λ-tree rewriting consists of *collapsed λ-tree reduction* on the one hand and a copying mechanism on the other hand. Both kinds of manipulating collapsed λ-trees are based on hypergraph rewriting and can be executed in arbitrary order. Before introducing collapsed λ-tree rewriting, we briefly recall hypergraph rewriting. (For a precise formal definition see, e.g. [19].)

A (hypergraph rewriting) *rule* is a tuple $r = (L, b{:}\, K \to R)$ where L, K, and R are hypergraphs, b is a hypergraph morphism and $K \subseteq L$. Let H be a hypergraph and let $f{:}\, L \to H$ be a hypergraph morphism such that the following *gluing condition* holds. (1) No hyperedge in $E_H - E_{f(L)}$ is incident to any node in $V_{f(L)} - V_{f(K)}$ and (2) for all nodes $x, y \in V_L$, $f(x) = f(y)$ implies $x = y$ or $x, y \in V_K$; analogously for all hyperedges $x, y \in E_L$. Then the *application* of r to H (via f) yields a hypergraph obtained by (1) removing $f(L) - f(K)$ from H and (2) gluing the remaining graph with R in $b(K)$.

Collapsed λ-tree reduction
Let H be a collapsed λ-tree. Then H *reduces* to the hypergraph H', denoted by $H \underset{red}{\Longrightarrow} H'$, if H' is obtained by (1) applying the following rule red to H and (2) deleting all nodes and hyperedges in the resulting hypergraph that do not lie on a path from $root_H$. The rule $red = (L, b{:}\, K \to R)$ consists of a collapsed λ-tree L and two hypergraphs K and R and can be depicted as in Fig. 1 where $v_1 = v_5$ ($v_3 = v_4$) means that the nodes v_1 and v_5 (v_3 and v_4) of K are mapped to the same node in R. For a collapsed λ-tree H and a hypergraph morphism $f : L \to H$, the subhypergraph $f(L)$ of H is called an *L-occurrence* in H. Fig. 2 shows a collapsed λ-tree reduction.

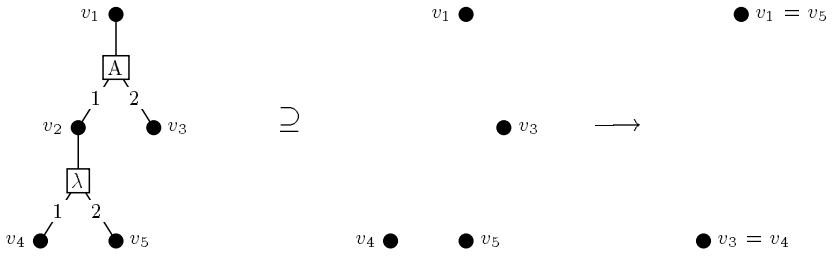

Fig. 1. The rule red

Collapsed λ-tree reduction preserves collapsed λ-trees. Moreover, it performs the β-reduction in the λ-calculus.

Theorem 3.1 *(Soundness of $\underset{red}{\Longrightarrow}$)*
Let H be a collapsed λ-tree and let $H \underset{red}{\Longrightarrow} H'$. Then $term(H) \xrightarrow[\beta]{+} term(H')$.[4]

[4] $\xrightarrow[\beta]{}$ and $\xrightarrow[\beta]{+}$ are the β-reduction relation in the λ-calculus and its transitive closure.

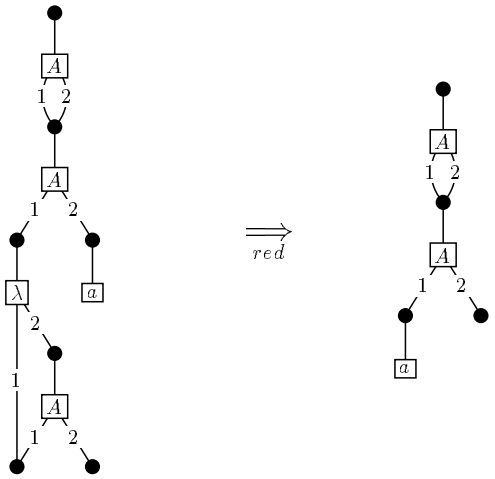

Fig. 2. A collapsed λ-tree reduction

Splitting L-occurrences

Because of the gluing condition, the rule *red* cannot be applied to an L-occurrence X in a collapsed λ-tree H if the source of the λ-hyperedge in X occurs more than once as target of hyperedges in H. Hence, there may occur situations in which a β-reduction may be applied to $term(H)$ but no corresponding collapsed λ-tree reduction can be performed. In such cases, it is desirable to provide a splitting mechanism for L-occurrences (see also [20]). To achieve this aim, one can use a set of so-called *split* rules consisting of the three subsets *begin_split*, *main_split* and *end_split* given in the Appendix. These make use of *negative context* conditions in the sense of [11].[5] The split of an L-occurrence is obtained by applying at most one rule of *begin_split*, then the rules of *main_split* as long as possible, and finally the rules of *end_split* as long as possible. Roughly speaking, splitting an L-occurrence X in a collapsed λ-tree H consists of performing a recursive operation $split(e)$ on the λ-hyperedge e in X that copies each path p from $s_H(e)$ to $t_H(e)|_1$ (provided that it is not already copied), and applies $split(e')$ to each λ-hyperedge e' on p. The resulting derivation relation is denoted by $\underset{split}{\Longrightarrow}$ and preserves collapsed λ-trees as well as the represented λ-terms.

Collapsed λ-tree rewriting

As indicated before, collapsed λ-tree rewriting, denoted by $\underset{\lambda}{\Longrightarrow}$, is the union of the relations $\underset{red}{\Longrightarrow}$ and $\underset{split}{\Longrightarrow}$. From the soundness of $\underset{red}{\Longrightarrow}$ and the fact that $\underset{split}{\Longrightarrow}$ preserves collapsed λ-trees as well as the represented λ-terms follows that collapsed λ-tree rewriting is sound.

Theorem 3.2 *(Soundness of $\underset{\lambda}{\Longrightarrow}$)*

Let H be a collapsed λ-tree and let $H \underset{\lambda}{\Longrightarrow} H'$. Then $term(H) \xrightarrow[\beta]{*} term(H')$.[6]

[5] If one admits larger sets of rules one can renounce the negative context conditions.

[6] $\xrightarrow[\beta]{*}$ denotes the reflexive and transitive closure of $\xrightarrow[\beta]{}$.

Since collapsed λ-tree representation of λ-terms may involve sharing, the application of *red* corresponds to a (non-empty) sequence of β reduction steps in the represented λ-term. Hence, for a collapsed λ-tree H and a λ-term t, $term(H) \underset{\beta}{\longrightarrow} t$ does not imply that there is a collapsed λ-tree H' such that $H \underset{\lambda}{\Longrightarrow} H'$ and $term(H') = t$. But the Gross-Knuth strategy $\underset{gk}{\longrightarrow}$ (see [3]) that roughly speaking reduces all redexes in a λ-term in parallel, can be implemented by a sequence of collapsed λ-tree rewriting steps. From this fact, from the soundness of $\underset{\lambda}{\Longrightarrow}$ and from the normalizing property of $\underset{gk}{\longrightarrow}$, it follows that a collapsed λ-tree H has a normal form if and only if $term(H)$ has a normal form.

Theorem 3.3 *(completeness w.r.t.* $\underset{gk}{\longrightarrow}$*)*
Let H be a collapsed λ-tree and let $term(H) \underset{gk}{\longrightarrow} t$. Then there is a collapsed λ-tree H' such that $H \overset{*}{\underset{\lambda}{\Longrightarrow}} H'$ and $term(H') = t$.

Corollary 3.4 *(Normal forms)*
Let H be a collapsed λ-tree. Then H has a normal form with respect to $\underset{\lambda}{\Longrightarrow}$ if and only if $term(H)$ has a normal form with respect to $\underset{\beta}{\longrightarrow}$.

4 Work to be done

There are several points of investigation that remain open. Some of them are given here. (1) The presented split procedure has to be compared with the copying mechanism proposed in [20]; (2) collapsed λ-tree rewriting should be compared with optimal λ-calculus reduction considered in [17,16,10,2] and with algebraic term graph rewriting presented by Kahl ([15]); (3) it should be studied which other properties of the λ-calculus (like the Church-Rosser property) carry over to collapsed λ-tree rewriting; and (4) reduction strategies for collapsed λ-tree rewriting could be considered.

Acknowledgement. I am grateful to Renate Klempien-Hinrichs, Detlef Plump, and to the referees for their helpful comments.

References

[1] Z.M. Ariola and J.W. Klop. Cyclic lambda graph rewriting. In *Proc. Ninth annual IEEE Symposium on Logic in Computer Science*, pages 416–426. IEEE Computer Society Press, 1994.

[2] A. Asperti. $\delta o! \epsilon = 1$: Optimizing optimal λ-calculus implementations. In Jieh Hsiang, editor, *Rewriting Techniques and Applications*, volume 914, pages 102–116, 1995.

[3] H.P. Barendregt. *The Lambda Calculus, its Syntax and Semantics.* North-Holland, Amsterdam, 1984.

[4] H.P. Barendregt. Functional programming and lambda calculus. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, volume B*, pages 321–364. Elsevier - The MIT Press, 1992.

[5] V. Claus, H. Ehrig, and G. Rozenberg, editors. *Graph Grammars and Their Application to Computer Science and Biology*. LNCS 73, 1979.

[6] Andrea Corradini and Francesca Rossi. Hyperedge replacement jungle rewriting for term-rewriting systems and logic programming. *Theoretical Computer Science*, 109:7–48, 1993.

[7] H. Ehrig, H.-J. Kreowski, and G. Rozenberg, editors. *Graph Grammars and Their Application to Computer Science*. LNCS 532, 1991.

[8] H. Ehrig, M. Nagl, and G. Rozenberg, editors. *Graph-Grammars and Their Application to Computer Science*, volume 153, 1983.

[9] H. Ehrig, M. Nagl, G. Rozenberg, and A. Rosenfeld, editors. *Graph-Grammars and Their Application to Computer Science*. LNCS 291, 1987.

[10] G. Gonthier, M. Abadi, and J.J. Lévy. The geometry of optimal lambda reduction. In *Proc. of the 19th Symposium on Principles of Programming Languages (POPL 92)*, 1992.

[11] A. Habel, R. Heckel, and G. Taentzer. Graph grammars with negative application conditions. *Fundamenta Informaticae*, 1995. To appear.

[12] A. Habel, H.-J. Kreowski, and D. Plump. Jungle evaluation. *Fundamenta Informaticae*, XV:37–60, 1991.

[13] J. Hindley and J. P. Seldin. *Introduction to Combinators and λ-calculus*, volume 1 of *London Mathematical Society Student Texts*. Cambridge University Press, 1986.

[14] B. Hoffmann and D. Plump. Implementing term rewriting by jungle evaluation. *RAIRO Theoretical Informatics and Applications*, 25 (5):445–472, 1991.

[15] W. Kahl. Algebraic term graph rewriting with bound variables, 1994. Talk on the fifth International Workshop on Graph Grammars and Their Application to Computer Science, November 1994, Williamsburg, USA.

[16] J. Lamping. An algorithm for optimal lambda calculus reduction. In *Proc. 17th ACM Symp. of Principles of Programming Languages*, pages 16–30, San Francisco, 1990.

[17] J.J. Lévy. Optimal reductions in the lambda-calculus. In J.P. Seldin and J.R. Hindley, editors, *To H.B. Curry: Essays on Combinatory Logic, Lambda-Calculus and Formalism*, pages 159–191. New York: Academic Press, 1980.

[18] *Graph Grammars and Their Application to Computer Science*. Lecture Notes in Computer Science, 1995. To appear.

[19] D. Plump. *Evaluation of Functional Expressions by Hypergraph Rewriting*. PhD thesis, University of Bremen, 1993.

[20] C.P. Wadsworth. *Semantics and pragmatics of the lambda-calculus*. PhD thesis, University of Oxford, 1971.

Appendix

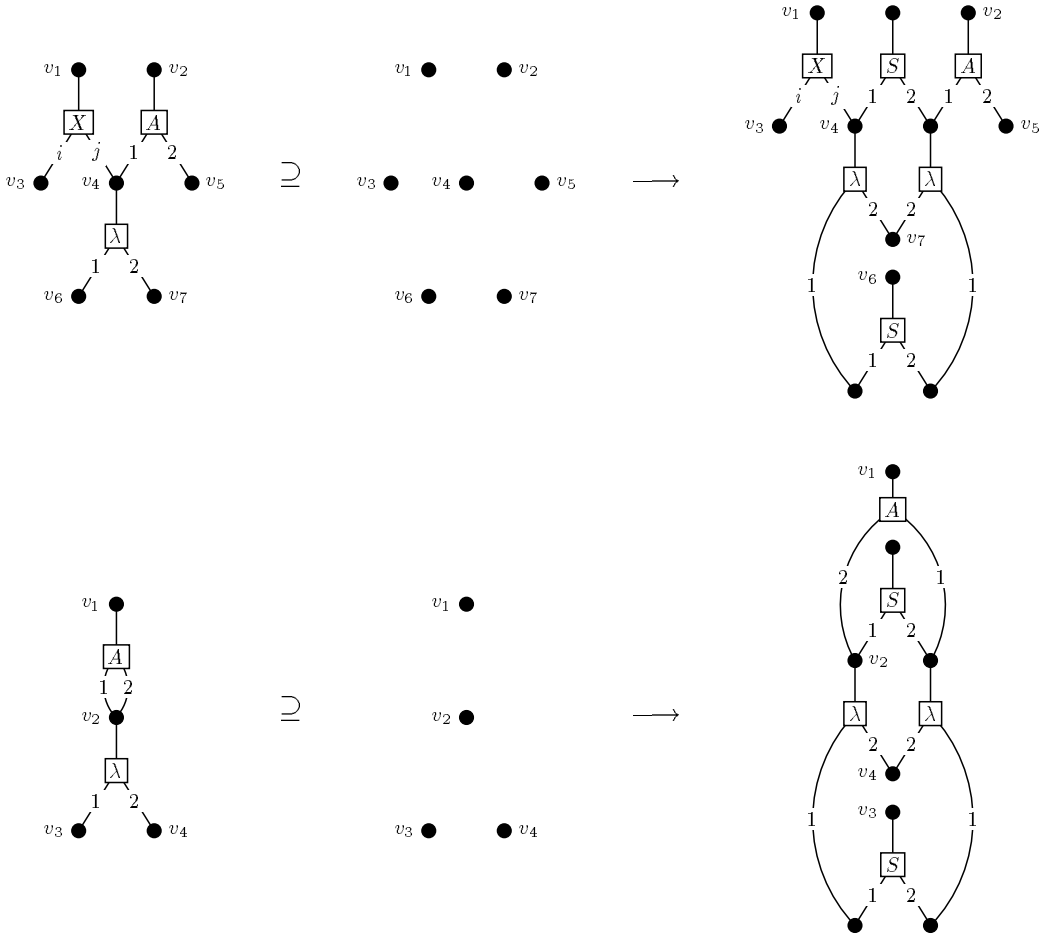

Fig. 3. The rules of *begin_split* where $i, j \in \{1, 2\}, i \neq j$ and $X \in \{A, \lambda\}$

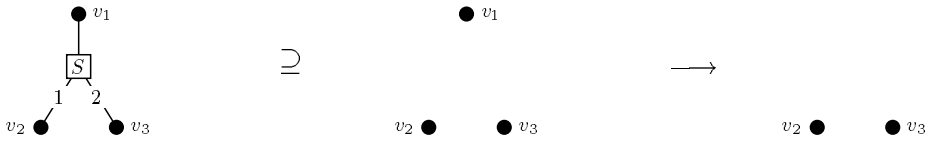

Fig. 4. The rule of *end_split*

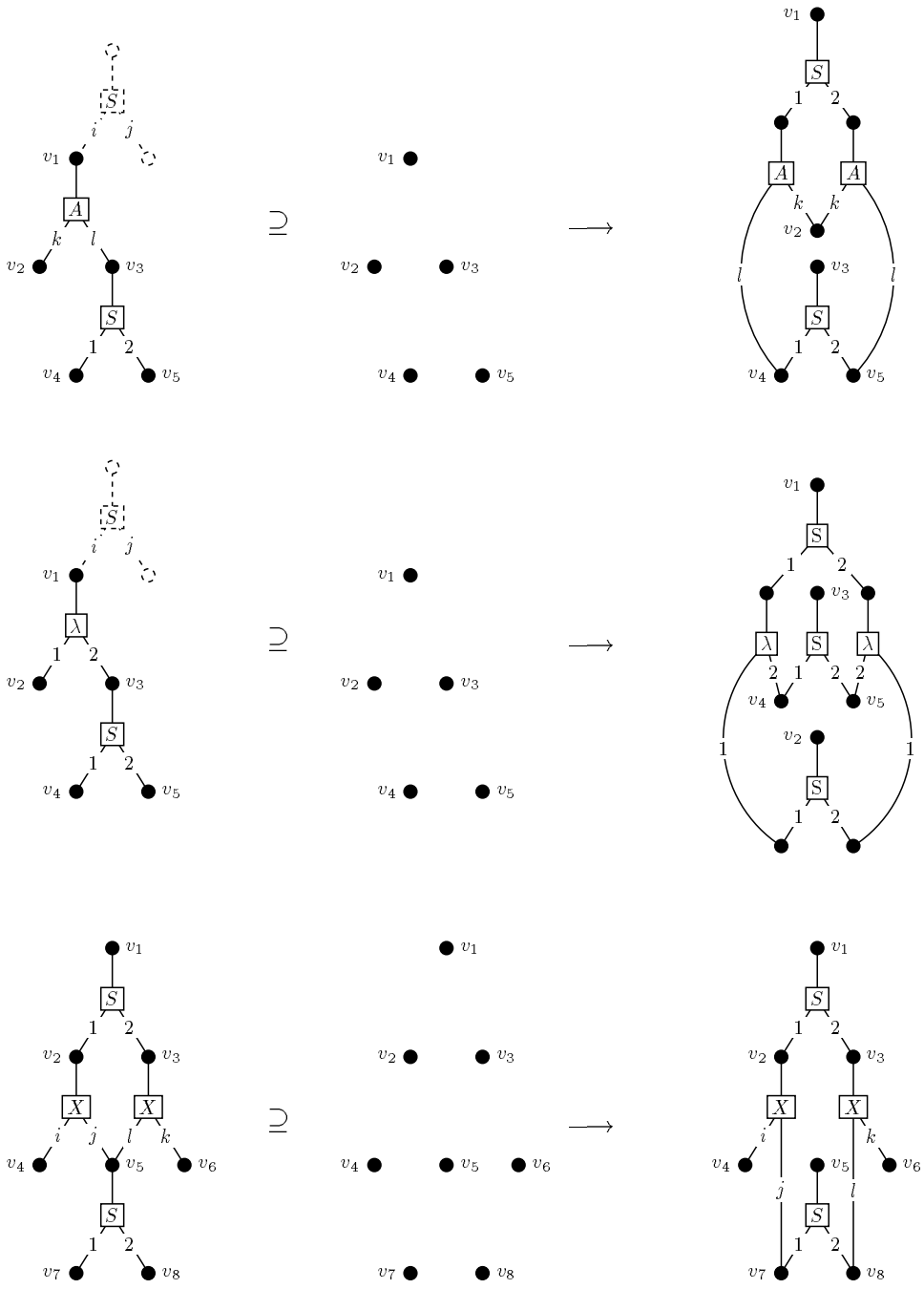

Fig. 5. The rules of *main_split* where dashed parts represent negative context, $i, j, k, l \in \{1, 2\}$ $i \neq j$, $k \neq l$ and $X \in \{A, \lambda\}$.