DBtk: A Toolkit for Directed Bigraphs

DBtk: a Toolkit for Directed Bigraphs

Giorgio Bacci Davide Grohmann Marino Miculan

Department of Mathematics and Computer Science, University of Udine, Italy
{giorgio.bacci,grohmann,miculan}@dimi.uniud.it

Abstract. We present DBtk, a toolkit for Directed Bigraphs. DBtk supports a textual language for directed bigraphs, the graphical visualization of bigraphs, the calculation of IPO labels, and the calculation of redex matchings. Therefore, this toolkit provides the main functions needed to implement simulators and verification tools.

1 Introduction

Bigraphical Reactive Systems (BRSs) [7] are a promising meta-model for ubiquitous (i.e., concurrent, communicating, mobile) systems. A calculus can be modeled as a BRS by encoding its terms (or states) as *bigraphs*, semi-structured data capable to represent at once both the location and the connections of the components of a system. The reduction semantics of the calculus is represented by a set of rewrite rules on this semi-structured data. Many calculi and models have been successfully represented as BRSs, such as CCS, Petri Nets, Mobile Ambients and, in the "directed" variant of [2], also Fusion calculus [5,6,3].

BRSs offer many general and powerful results. Particularly important for verification purposes is the possibility to derive systematically labelled transition systems via the so-called *IPO construction* [5], where the labels for a given agent are the *minimal* contexts which trigger a transition. Interestingly, the strong bisimilarity induced by this LTS is always a congruence.

Moreover, a "bigraphical simulation engine" would allow to obtain immediately a simulator for any calculus/model formalized as a BRS. The core of this engine will be the implementation of *redex matching*, that is, to determine when and where the left-hand side of a bigraphical reaction rule matches a bigraph; then this redex is replaced with the right-hand side of the same rule.

These are the main features offered by the *Directed Bigraphs Toolkit* (DBtk), which we describe in this paper. The architecture of DBtk is shown in Fig. 1. First, the toolkit defines data structures and operations for representing and manipulating bigraphs (Section 2). For more conveniently interacting with the user, bigraphs can be described using a language called DBL (Section 3), and also graphically visualized by means of a SVG representation (Section 4). Then, the toolkit provides the functions for calculating matchings of a redex within a bigraph (Section 5) and the RPOs and IPOs of directed bigraphs (Section 6).

Comparison with related work and directions for future work are in Section 7. DBtk can be found, with examples and more detailed descriptions, at `http://www.dimi.uniud.it/grohmann/dbtk`.

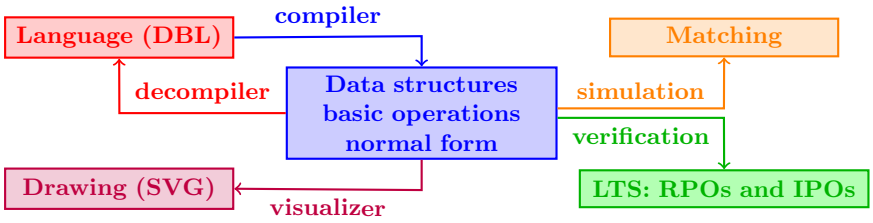


Fig. 1. Architecture of DBtk.

2 Basic data structures and operations

The Main module contains the basic data structures for representing directed bigraphs, and the two basic categorical operations: composition and tensor product. Two bigraphs can be composed by putting the roots of the "lower" one inside the sites (or holes) of the "upper" one, and by pasting the links having the same names in the common interface, whereas the tensor product juxtaposes two bigraphs if they do not share names on their inner and outer interfaces, respectively. For user convenience, the derived operations (such as sharing products and merge product) are also provided. Moreover, the module provides the support for the bigraph algebra, i.e., functions for constructing the elementary bigraphs (see Table 1), and the *normalization procedure*, which takes a generic bigraph and yields a triple of bigraphs (more precisely two wirings and a discrete bigraph) that represent its directed normal form (see [4]).

3 DBL, the directed bigraphical language

The *directed bigraphical language* (DBL) is a (term) language for bigraphs, which follows the algebra defined in [4]. An expression in DBL can be compiled in the internal representation of bigraphs. Also a decompiler is provided, translating a bigraph represented in the internal data structures back to a term expression.

A bigraph definition begins with a signature definition, whose syntax is:

```
Signature [ACTIVITY NAME:#PORTS,...] ;
```

where `ACTIVITY` is picked from the set $\{\text{active}, \text{passive}, \text{atomic}\}$, `NAME` is the name of the control, and `#PORTS` is the number of the node ports. For example:

```
Signature [passive n1:2, atomic node_AT:0, active tr:1] ;
```

Then, the bigraph is described in a functional-style language, which allows to compose elementary bigraphs (whose syntax is in Table 1) and other expressions with the various operators of the algebra. Composition and tensor product are denoted by $\circ$ and `*`, respectively. As a shortcut, also the sharing operators are also provided: outer sharing product (`/\`); inner sharing product (`\/`); sharing product (`||`); prime outer product (`/^\`); and prime sharing product (`|`).

As an example, the following specification

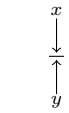
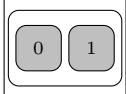
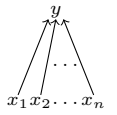
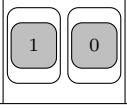
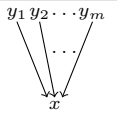
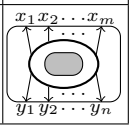
Barren root (1) `merge 0` (special case of `merge`)		Closure ($\bigwedge_y^x$) `x X y` (generalized to `[x,w] X [y,z]`)	
Merge (*merge*) `merge 2` (generalized to `merge n`)		Substitution ($\triangle_X^y$) `y / [x_1,...,x_n]` (also `y / x` and `y /`)	
Swap (γ) `@[1,0]` (generalized to `@[3,4,1,0,2]`)		Fusion (∇_x^Y) `[y_1,...,y_n] \ x` (also `y \ x` and `\ x`)	
Ion ($K_{\vec{y}}^{\vec{x}}(l)$) `K[-y_1,...,-y_n,+x_1,...,+x_m]` (the names in the list can be written in any order)			

Table 1. Syntax expressions for elementary directed bigraphs.

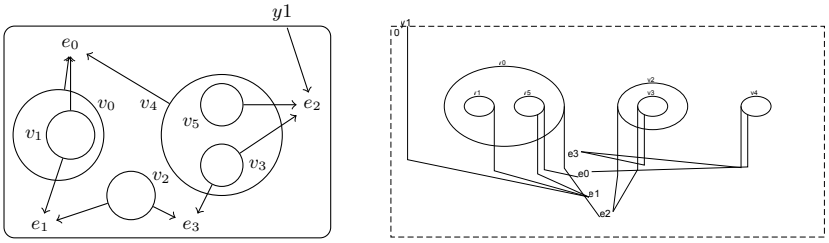


Fig. 2. A directed bigraph (left) and its SVG representation (right).

```
Signature [active t4:1,atomic t5:1,atomic t2:2];
let a = merge 1|X[x,z]|(y1X[a,y]*t4 [-w])°
                                  (w\w*merge 1|t5 [+a]*y/y) in
  let b = (((wX[b,c]*merge 1)°
           (b/b*t4 [+c])°t2 [+b,-d]|t2 [+x,-e])°([d,e]X)) in
    let c = t2 [+z,+y] in a°(b*c)
```

corresponds to the directed bigraph shown in Fig. 2.

4 Graphical Visualization

In addition to the bigraphical language, DBtk allows to represent bigraphs in the XML-based SVG format, which is the W3C open standard for vector graphics; many web browsers have free, native support for SVG rendering. The translation function `dbg2svg` takes a directed bigraph data structure and returns a string containing its SVG representation. This function is fruitful in combination with the compiler module: bigraphs are defined in DBL, compiled to the DBG-data structure and then written to an SVG file (see Fig. 2, right).

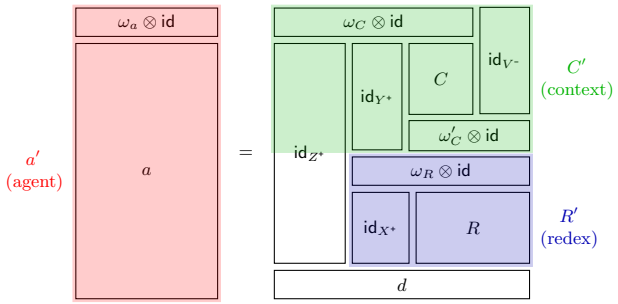


Fig. 3. Schema of a valid matching sentence

5 Matching

The main challenge for implementing the dynamics of bigraphical reactive systems is the *maching problem*, that is, to determine whether a reaction rule can be applied to rewrite a given bigraphical agent. To this end, we have to find when, and where, a redex appears in a bigraph; then this redex can be replaced with the corresponding reactum.

The matching problem can be defined formally as follows: given a ground bigraphical agent a' and a (parametric) redex R', there is a match of R' in a' if there exist a context C' and parameters d such that $a' = C' \circ (R' \otimes id) \circ d$. A matching can be decomposed in a set of simpler components, as in Fig. 3. It is evident that solving this problem is necessary to yield a mechanism for simulating bigraphical computations.

To solve this equation we have defined a set of inference rules that, by induction on the structure of a' and R', find the matching context C' and parameters d, preserving the equality at each inference step. So, we give an inductive characterization of a matching algorithm for directed bigraphs, along the lines of [1]. The main difference between the characterization presented in [1] is that we consider *directed* bigraphs (without binders). For this reason, both the structure of a matching sequence and many rules have to be changed and generalized.

We base our presentation on discrete decomposition of directed bigraphs:

Proposition 1 (Discrete decomposition). *Any directed bigraph G can be decomposed uniquely (up to iso) into a composition of the following form:*

$$G = (\omega \otimes \mathsf{id}) \circ D \circ (\omega' \otimes \mathsf{id})$$

where D is discrete and ω, ω' are two wirings satisfying the following conditions:

1. *in ω if two outer downward names are peer, then their target is an edge;*
2. *in ω' there are no edges, and no two inner upward names are peer (i.e. on inner upward names is a renaming, but outer downward names can be peer).*

For a formal proof see the discrete normal form in [4]. The two conditions ensure that all edges are in ω and that the two wirings are of the following forms:

$$\omega = \delta \blacktriangledown_{\vec{y}}^{\vec{x}} \sigma \otimes \alpha \qquad\qquad \omega' = \delta' \otimes \beta$$

for some downward renaming α, upward renaming β, fusions δ, δ' and substitution σ. Note that for a ground bigraph g the wiring ω' is simply id_ε.

Using the same approach of [1] we now define *matching sentences* and a set of rules for deriving valid matching sentences.

Definition 1 (Matching Sentence). *A* matching sentence *is a 7-uple over wirings and directed bigraphs, written*

$$\omega_a, \omega_R, \omega'_C, \omega_C \vdash a, R \hookrightarrow C, d$$

where a, R, C, d are discrete directed bigraphs, R and C have not renamings (i.e. are just composed by discrete molecules), ω_a, ω_R, ω_C are wirings with no inner downward names, and ω'_C is a generic fusion.

A matching sentence is valid *if $a' = C'(\mathsf{id}_{Z^+} \otimes R')d$ holds, where*

$$\begin{aligned} a' &= (\omega_a \otimes \mathsf{id})a && \textit{(agent)} \\ R' &= (\omega_R \otimes \mathsf{id})(\mathsf{id}_{X^+} \otimes R) && \textit{(redex)} \\ C' &= (\omega_C \otimes \mathsf{id} \otimes \mathsf{id}_{V^-})\big(\mathsf{id}_{(Z \uplus Y)^+} \otimes (\mathsf{id}_{V^-} \otimes C)(\omega'_C \otimes \mathsf{id})\big) && \textit{(context)} \end{aligned}$$

Note that the notion of valid sentence precisely capture the abstract definition of matching: for a general match $a' = C'(\mathsf{id}_{Z^+} \otimes R')d$, by Proposition 1 and some simplifications due to the groundness of a and d, we can decompose a', R', C' obtaining a corresponding valid matching sentence; conversely, if $\omega_a, \omega_R, \omega'_C, \omega_C \vdash a, R \hookrightarrow C, d$ is valid, by definition, there exist a', R' C' such that $a' = C'(\mathsf{id}_{Z^+} \otimes R')d$.

The discrete decomposition of the agent, context and redex in a valid matching sentence is schematically shown in the picture in Figure 3. Composition and tensor product on bigraphs are depicted as vertical and horizontal juxtaposition.

We infer valid matching sentences using the inference system given in Fig. 4.

A detailed discussion of this system is out of the scope of this paper; we refer to [1] for explanation of a similar system. We just notice that we can prove that this matching inference system for directed bigraphs is sound and complete, that is, all valid matching sequences can be derived using these rules.

A redex can occur many times inside an agent, hence the resolving procedure is highly non-deterministic. This is implemented by a back-tracking technique based on two stacks: one for storing all the applicable rules, and the other for tracing all the different ways to apply a single rule. To solve a matching problem we create a `match` object for the pair (a', R'), whose constructor initializes the two stacks; then invoking the `next` method the next match (i.e. the pair (C', d)) found exploring the searching tree is returned, until no matches are found anymore.

An example of matching is given in Fig. 5. Consider the agent a in the picture above, and the redex R below. A matching of R into a split the agent itself in three parts: a context, the redex and some parameters. In this case, there exist two possible splitting: one identifies the redex's round node with the agent's round node inside the left rectangle, and the rectangle node with the one on the right in the agent. So, the context has the left rectangle and the below round

$$\textsc{Prime-Axiom}\quad \frac{p\colon \langle W^+\rangle \qquad \sigma\colon Z^+ \to \qquad \alpha\colon W^+ \to Z^+}{\sigma\alpha, \mathsf{id}_\varepsilon, \mathsf{id}_\varepsilon, \sigma \vdash p, \mathsf{id}_1 \hookrightarrow \mathsf{id}_1, (\alpha \otimes \mathsf{id}_1)p}$$

$$\textsc{Switch}\quad \frac{\omega_a, \mathsf{id}_\varepsilon, \mathsf{id}_\varepsilon, \omega_C(\mathsf{id}_{Z^+} \otimes \sigma_R) \vdash p, \mathsf{id}_1 \hookrightarrow P, d \qquad d\colon \langle m, Z^+\rangle}{\omega_a, \sigma_R, \mathsf{id}_\varepsilon, \omega_C \vdash p, P \hookrightarrow \mathsf{id}_1, d}$$

$$\textsc{Ion}\quad \frac{\omega_a, \omega_R, \omega'_C, \omega_C \vdash p, R \hookrightarrow P, d \qquad \sigma\colon \{\vec{y}\}^+ \to}{\sigma \curlywedge \omega_a, \omega_R, \omega'_C, \sigma\vartriangle^{\vec{y}}_{\vec{x}} \curlywedge \omega_C \vdash (K^{\vec{y}}_{\epsilon} \otimes \mathsf{id}_{U^+})p, R \hookrightarrow (K^{\vec{x}}_{\epsilon} \otimes \mathsf{id}_W)P, d}$$

$$\textsc{Par}\quad \frac{\omega_a, \omega_R, \omega'_C, \omega_C \curlywedge \sigma \vdash a, R \hookrightarrow C, d \qquad \omega_b, \omega_S, \omega'_D, \omega_D \curlywedge \sigma \vdash b, S \hookrightarrow D, e}{\begin{array}{c}\omega_a \curlywedge \omega_b\ ,\ \omega_R \curlywedge \omega_S,\\ \omega'_C \otimes \omega'_D\ ,\ \omega_C \curlywedge \omega_D \curlywedge \sigma\end{array} \vdash a \otimes b, R \otimes S \hookrightarrow C \otimes D, d \otimes e}$$

$$\textsc{Perm}\quad \frac{\omega_a, \omega_R, \omega'_C, \omega_C \vdash a, \bigotimes_i^m P_{\pi^{-1}(i)} \hookrightarrow C, d}{\omega_a, \omega_R, \omega'_C, \omega_C \vdash a, \bigotimes_i^m P_i \hookrightarrow C\pi, d}$$

$$\textsc{Merge}\quad \frac{\omega_a, \omega_R, \omega'_C, \omega_C \vdash a, R \hookrightarrow C, d}{\omega_a, \omega_R, \omega'_C, \omega_C \vdash (merge \otimes \mathsf{id}_Y)a, R \hookrightarrow (merge \otimes \mathsf{id}_X)C, d}$$

$$\textsc{Edge-Lift}\quad \frac{\begin{array}{c}\delta_C\colon T^- \to V^- \qquad \zeta_C\colon S^- \to W^- \qquad C\colon \langle n, W^-\rangle \to\\ \omega_a\ ,\ \sigma_R \otimes \tau_R\ ,\ \mathsf{id}_\varepsilon,\\ (\delta_C \curlyvee \triangledown_S)\delta_R \blacktriangledown^{\vec{x}}_{\vec{y}} (\mathsf{id}_{\{\vec{y}\}^+} \curlywedge \vartriangle^{\vec{y}}_{\vec{x}} \widehat{\delta}_R(\vartriangle^T \curlywedge \widehat{\zeta}_C)) \otimes \omega_C\end{array} \vdash a, R \hookrightarrow \widehat{C}, d}{\omega_a, \delta_R \blacktriangledown^{\vec{x}}_{\vec{y}} \sigma_R \otimes \tau_R, \delta_C \curlyvee \zeta_C, \omega_C \vdash a, R \hookrightarrow C, d}$$

$$\textsc{Wiring-Axiom}\quad \frac{}{\vartriangle^y, \vartriangle^{\vec{x}}, \mathsf{id}_\varepsilon, \vartriangle^y_{\vec{x}} \vdash \mathsf{id}_\varepsilon, \mathsf{id}_\varepsilon \hookrightarrow \mathsf{id}_\varepsilon, \mathsf{id}_\varepsilon}$$

$$\textsc{Close}\quad \frac{\sigma_a, \omega_R, \omega'_C, \sigma_C \vdash a, R \hookrightarrow C, d}{\begin{array}{c}(\delta \otimes \mathsf{id}_{U^+})(\blacktriangledown^{\vec{x}}_{\vec{y}} \otimes \mathsf{id}_{U^+})\sigma_a\ ,\ \omega_R\\ \omega'_C\ ,\ (\delta \otimes \mathsf{id}_{W^+})(\blacktriangledown^{\vec{x}}_{\vec{w}} \otimes \mathsf{id}_{W^+})\sigma_C\end{array} \vdash a, R \hookrightarrow C, d}$$

Fig. 4. Matching inference system.

node of the agent, whilst the parameter contains both nodes contained by the agent's rectangle node on the right. In the other case, the matching identifies both the redex's nodes with the nodes inside the agent's right rectangle. Hence, the context contains all the remaining nodes, and the parameter is empty.

6 RPO and IPO

The RPO/IPO module provides the functions for constructing relative pushouts and idem pushouts (see Fig. 6). The implementation of the RPO construction follows faithfully the algorithm given in [2]; it takes four bigraphs (f_0, f_1, g_0, g_1) as depicted in Fig. 6(1), and yields the RPO triple (h_0, h_1, h) as in Fig. 6(2).

An example of an RPO construction is in Fig. 7. Intuitively, the construction works as follows: the common parts of the span f_0, f_1 are removed from the bound g_0, g_1 and placed in h; the nodes/edges of f_0 not present in f_1 are added to h_1, and analogously for h_0; finally the middle interface is computed and the parent and link maps are computed, preserving compositions.

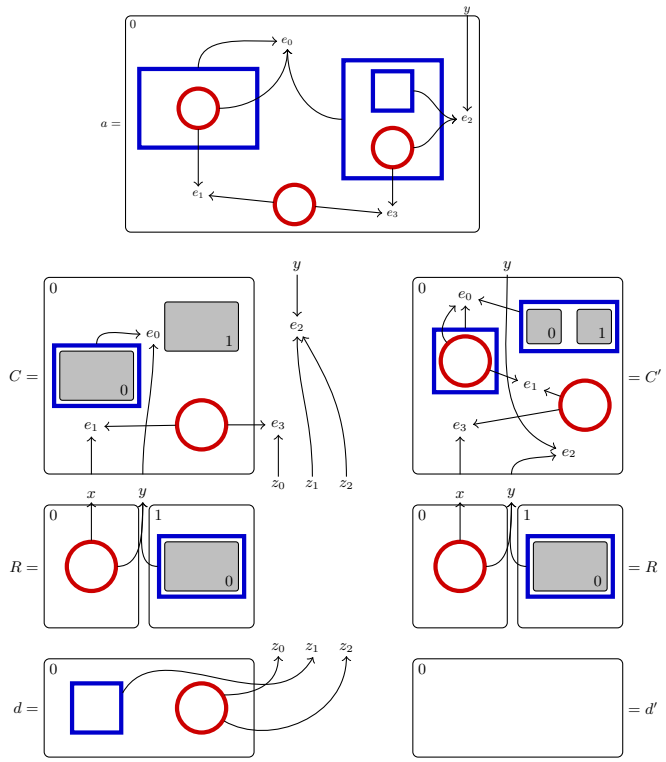


Fig. 5. An example: there are two matches of the redex R into the agent a.

The procedure calculating IPOs (see [2]) is split into two steps:

1. a check if the span of bigraphs has bounds;
2. the effective computation of the IPO.

Since the IPO construction is non-deterministic, an implementation must calculate all possible solutions. To do this, we define an object `IPO`, whose constructor takes two bigraphs (satisfying the consistency conditions, i.e., they must admit bounds) and internally computes all the possible sets (L^+, Q^+) and auxiliary functions $(\theta, \phi, \xi, \eta)$ needed in the construction. After the initialization, the IPO computation proceeds as follows: when the object method `next` is called, an IPO is calculated by using the current internal state of the object, which is then updated for the next call to `next`. If no IPO is found, an exception is raised.

An example of an IPO calculation is shown in Fig. 8. Where, after checking that the span f_0, f_1 has bounds, the computation of IPO yields out to different IPOs. The first (shown above) have an arrow connecting the node v_2 to y, whilst the second an arrow connecting v_2 to z. Both cases are suitable, for the fact that v_2 is linked to e_1 in f_1, and v_2 is not present in f_0, but e_1 belongs to f_0 and it is accessible by two names y and z. So, to get a bound for our span, we must

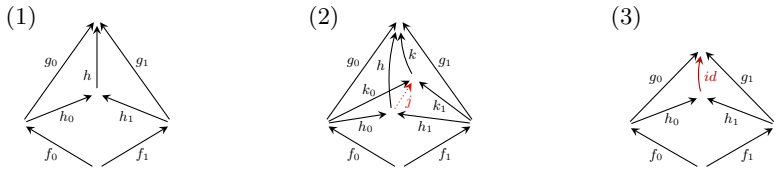


Fig. 6. A candidate RPO (1), an RPO (2) and an IPO (3).

add v_2 to h_0 and connect its port to the edge e_1 in f_0, to do so, there are two possibility to target e_1: trough the name y or trough the name z.

7 Related and future work

Related work. The closest work is Birkedal et al. [1], where a language and a matching algoritm for *binding* bigraphs are presented. That work has been of inspiration for our matching inference system, although we had to change quite many details both in the rules and in the implementation due to the fact that we consider directed bigraphs and not binding bigraphs. Also, as far as we know, DBtk provides the first constructions of RPOs and IPOs for bigraphs.

Future work. The present system is still a prototype, and needs some refinements to mature to a ready-to-use tool. Among the features that can be added, we mention a graphical user interface (GUI), where the user can "draw" interactively a bigraph, a reaction rule and ultimately a whole BRSs. Another interesting development will be to consider *stochastic bigraphic reactive systems*, where rules are endowed with *rates*. We think that the matching procedure given in this paper can be successfully used within a Gillespie algorithm.

Acknowledgements. We thank our students Raul Pellarini, Patrik Osgnach, Federico Chiabai for their help in developing the code.

References

1. L. Birkedal, T. C. Damgaard, A. J. Glenstrup, and R. Milner. Matching of bigraphs. *ENTCS*, 175(4):3–19, 2007.
2. D. Grohmann and M. Miculan. Directed bigraphs. In *Proc. XXIII MFPS*, *ENTCS*, 173:121–137. Elsevier, 2007.
3. D. Grohmann and M. Miculan. Reactive systems over directed bigraphs. In *Proc. CONCUR 2007*, *LNCS* 4703, pages 380–394. Springer-Verlag, 2007.
4. D. Grohmann and M. Miculan. An algebra for directed bigraphs. In *Proceedings of TERMGRAPH 2007*, *ENTCS* 203(1):49–63. Elsevier, 2008.
5. O. H. Jensen, R. Milner. Bigraphs and transitions. In *Proc. POPL*, 38–49, 2003.
6. J. J. Leifer and R. Milner. Transition systems, link graphs and petri nets. *Mathematical Structures in Computer Science*, 16(6):989–1047, 2006.
7. R. Milner. Bigraphical reactive systems. In *Proc. 12th CONCUR*, *LNCS* 2154, pages 16–35. Springer, 2001.

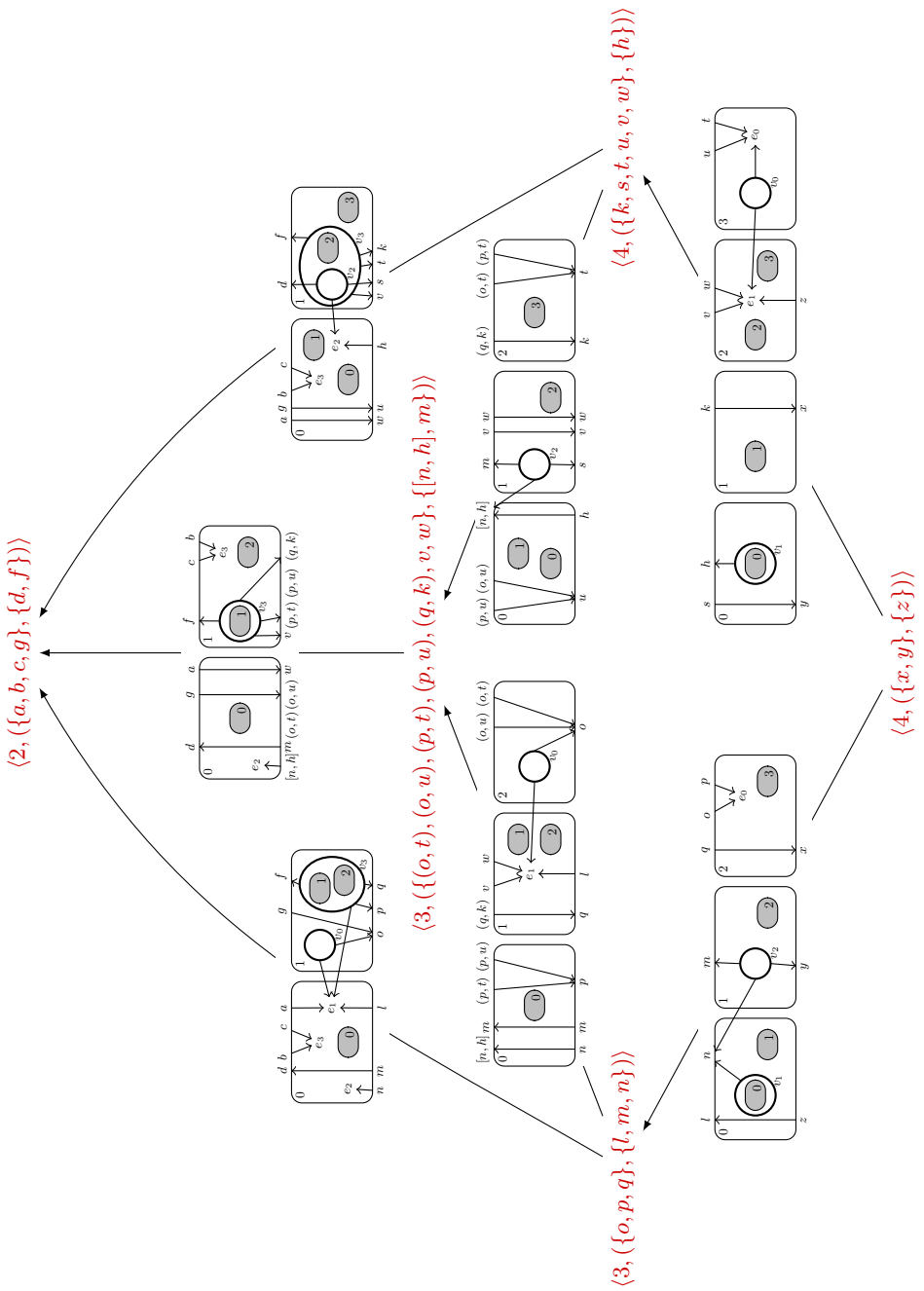


Fig. 7. An example of an RPO construction.

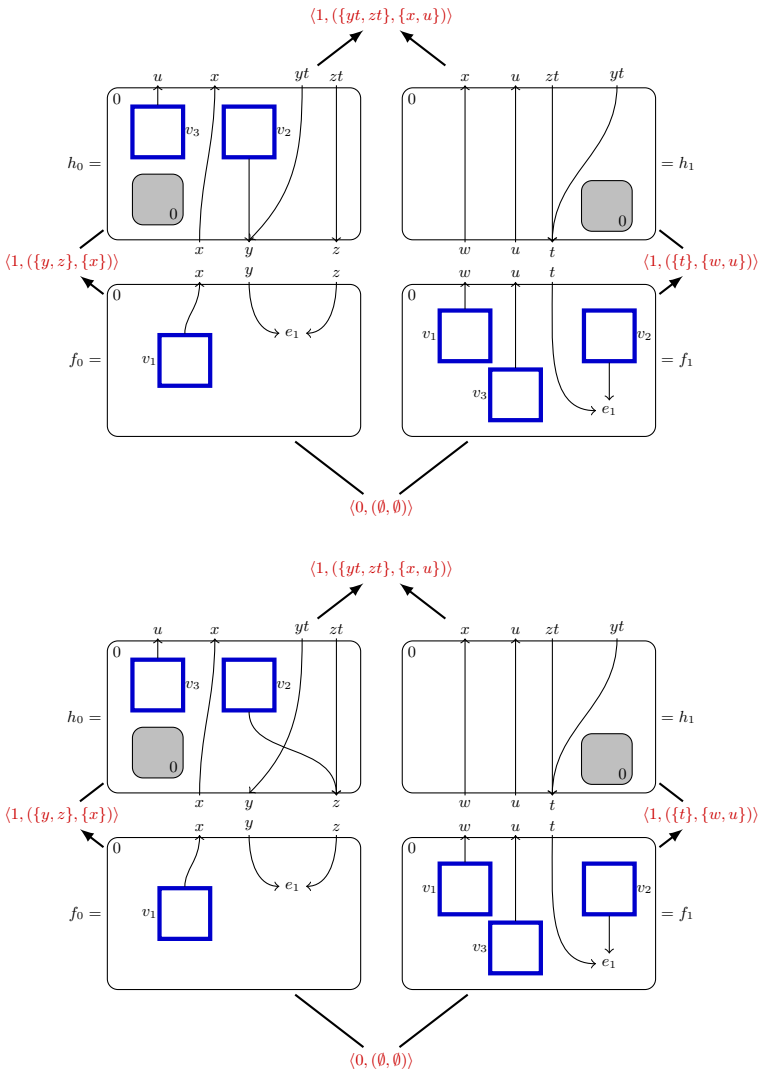


Fig. 8. An example: two IPOs are possible for the span f_0, f_1, one connects v_2 to y and the other v_2 to z.