Multimedia Adaptation Decisions Modelled as Non-Deterministic Operations

TR-GTI-UAM-UPM-ITEC-AINF-UKLA-December-2007

F. López, J. M. Martínez, D. Jannach, C. Timmerer, H. Hellwagner, N. García

Appendix A: Algorithms

1. For each property q_k into the preconditions o the c_i conversion:
   a) If there exists a property p_j in the postconditions of c_{i+1} where its label corresponds to the label of q_k:
      a. If $p_j \cap q_k = \varnothing$ the matching process fails because c_{i+1} is not capable of producing a property requested by the preconditions of c_i.
      b. Otherwise
         i. Add $p_j \cap q_k$ to the target parameters of c_i. This operation allows selecting the target parameters of the conversion.
         ii. Add (or replace) $p_j \cap q_k$ into the postconditions of c_i. This operation allows to know the accumulate effects of executing a conversion, and also guarantee that the postconditions always remain or increase. This condition is necessary to guarantee that the algorithm always finishes and is complete.
   b) Otherwise add q_k to the preconditions of c_{i+1}. In this way we are signalling that the q_k property does not change during the c_{i+1} conversion and the algorithm has to find a previous conversion $c_{i+1},\ldots,c_n$ that fulfil this condition.

Algorithm 1: Matching process algorithm

1. Use the goal state to build a conversion (named *finish*) that represents the goal conversion. This conversion must have only preconditions storing the properties of the UED and target parameters selecting the preconditions' properties.
2. Build a **list of prospective conversions** where their postconditions match with the goal conversion target parameters according to the matching process given in Algorithm 1.
3. For each c_{i+1} conversion in the list of prospective conversions:
   a. If the intersection between the property list of the postconditions of the c_{i+1} conversion and the property list of the target parameters of the c_i goal conversion is a set without empty properties, then this configuration can be interpreted in the sense that this conversion does not contribute to adapt the content, i.e., the same result can be reached with the previous generated conversions, and the c_{i+1} conversion is discarded. This condition avoids infinite loops in the search algorithm. Note that this condition never occurs when the c_i conversion is the *finish* conversion, because this conversion always has an empty target parameters

```
        property list.
    b. If the $c_{i+1}$ conversion is the *select* conversion and has preconditions, the
       preconditions cannot be fulfilled and the $c_{i+1}$ conversion is discarded.
4. If there are no remaining conversions in the list of prospective conversions that match
   with the goal conversion, then the request cannot be satisfied and we terminate the
   search in this path. Otherwise, the list of prospective conversions becomes the **list of
   valid conversions**.
5. If in the list of valid conversions there exist initial conversions (*select* conversions), we
   have found one successful path for each initial conversion and we terminate the tree of
   conversions construction on these paths. Note that if the initial conversion matches with
   the UED conversion, this means that the content is adapted.
6. For each conversion in the list of valid conversions that is not an initial conversion,
   consider the conversion a new goal conversion and repeat the algorithm recursively.
```

Algorithm 2: Tree of conversion computing algorithm

```
1. Create a stack $S_i$ for each *select* conversion $c_i$ of the tree of conversions. Each stack
   represents a sequence of states at the end of the algorithm.
2. For each *select* conversion $c_i$ of the tree of conversions:
    a. Create a state $s$ that represents the initial state of a sequence of states made up of
       the target properties of the $c_i$ conversion.
    b. Push $s$ into $S$
    c. While $c_i \neq finish$
         i.   $c_i = padre(c_i)$
         ii.  Add or replaces the target parameters of $c_i$ into $s$.
         iii. Push $s$ into $S$.
3. Return $\{S_1,S_2,...S_n\}$
```

Algorithm 3: Sequences of states computing algorithm

Appendix B: Use cases

In order to demonstrate the way in which the algorithm proposed in Appendix A works, we evaluate two use cases.

Use case 1

Let us consider the following adaptation problem (inspired by a multistep adaptation example developed in [1]) where we have an image *Component* described by the properties:

visual_format = {*bmp*}
visual_frame_width = {640}
visual_frame_height = {480}

Note that although *Component* elements can be represented with unique potential states, i.e., with variables to simplify the model we are going to generalize the representation of states using always multiple potential states, i.e., properties. Note also that the *Component* in this example has not specified whether the image is colour or greyscale.

In this example we also have a terminal that only support BMP greyscale images and the size of the terminal is 320x240. Thus the terminal properties are:

image_decoding_format = {*bmp*}
display_horizontal_resolution = {320}
display_vertical_resolution = {240}
display_color_capable = {*false*}

Note that this UED is a unique potential state, this due to the fact that in [1] they always consider that the UED is unique. In order to demonstrate the usefulness of non-deterministic planners, in Appendix B we are going to consider a multiple UED.

Additionally, we have one *ImageCAT* capable of performing two kinds of conversions: *image_resizer* that resizes BMP images and *image_greyscaler* that converts colour BMP images to greyscale BMP images. Specifically, the preconditions and postconditions of the *image_resizer* are respectively:

image_resizer

preconditions:

media_content = {*image*}
visual_forma t= {*bmp*}
visual_frame_ width = *
visual_frame_heigh t= *

postconditions:

media_content = {*image*}
visual_format = {*bmp*}
visual_frame_width = *
visual_frame_heigh t= *

image_greyscaler

preconditions:

media_content = {*image*}
visual_format = {*bmp*}

postconditions:

media_content = {*image*}
visual_format = {*bmp*}
visual_color_domain = {*grayscale*}

Note that both conversions demand the *media_content* property with the value *image*, and the *visual_format* with the value *bmp*. This request avoids the use of these conversions when the Component is not a BMP image. Note also that the *image_resizer* has not specified the *visual_color_domain* neither in the preconditions nor in the postconditions. This configuration must be interpreted in the sense that the conversion preserves the value of that property. Similarly, the *image_greyscaler* has not specified the *visual_frame_width* and the *visual_frame_height*, which, in the same way must be interpreted in the sense that the conversion does not modify these properties. In addition, the *image_greyscaler* has specified the *visual_color_domain* in the postconditions, but not in the preconditions. This configuration means that the *image_greyscaler* accept both colour and greyscale images, but always produces greyscale images.

Before initiating the fist move of the algorithm, it is important to mention that there is a mismatch between the MPEG-7 description of the *Component* elements and conversions, and the MPEG-21 description of the terminal. For instance, in this example there is a mismatch between the MPEG-7 description of the colour domain that can take the values *color*, *colorized*, *grayscale*, *binary* and the MPEG-21 description of the terminal's colour display capabilities that can take the values *true*, *false*. To cope with this problem we always transform the MPEG-21 properties associated with the UED into MPEG-7 properties, so we are going to represent the UED with the following MPEG-7 properties:

visual_format = {*bmp*}
visual_frame_width = {320}
visual_frame_height = {240}
display_color_capable = {*false*}

Figure 1 shows the way in which the Algorithm 2 constructs the tree of conversions with the *finish* conversion as the root of the tree. According to the algorithm, the fist step is to use the goal state to build the *finish* conversion. This *finish* conversion must have only preconditions storing the properties of the UED and target parameters selecting the preconditions, and then the *finish* conversion has the values given into the conversion labelled *conversion_a* on the Figure 1. Since there are not variations, we can anticipate that there is going to exist only one *select* conversion with the values given into the *conversion_f* and *conversion_g*, which really are two instances with the same values.

The second step of Algorithm 2 proposes to build a list of prospective conversions where their postconditions match with the goal conversion target parameters according to the Algorithm 1. In particular, as shown in Figure 1, the list of prospective conversions is made up of two conversions: *image_resizer* (labelled *conversion_b* on Figure 1) and *image_greayscaler* (labelled *conversion_c* on Figure 1). In this case both prospective conversions become valid conversions. In particular, the a rule of the algorithm does not apply because (1) the intersection between the property list of the postconditions of the *image_resizer* conversion and the property list of the target parameters of the *finish* conversion is a set with empty properties:

media_content = {}
visual_format = {*bmp*}
visual_frame_width = {320}
visual_frame_height = {240}
display_color_capable = {}

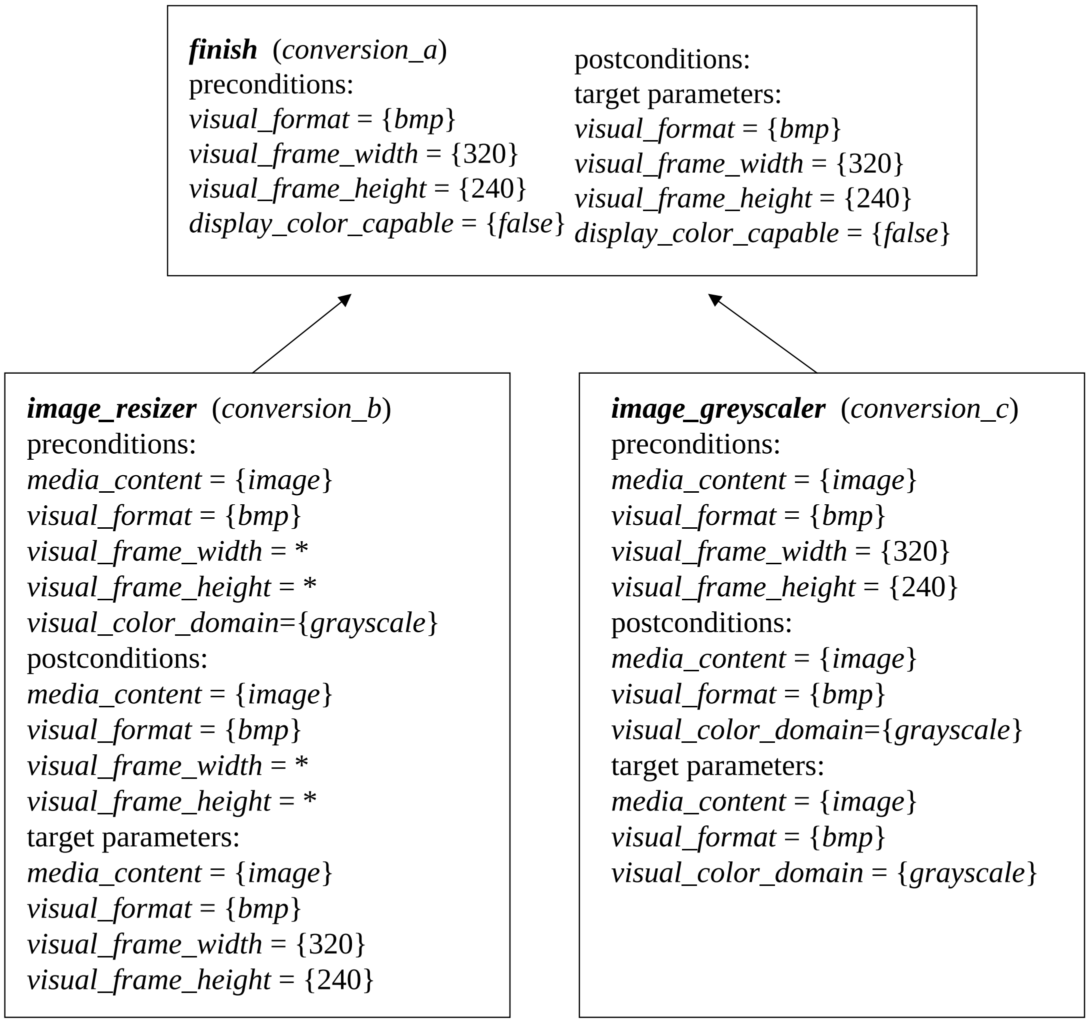
And also (2) the intersection between the property list of the postconditions of *image_greyscaler* conversion and the property list of the target parameters of the *finish* conversion is a set with empty properties:

media_content = {}
visual_format = {*bmp*}
visual_frame_width = {}
visual_frame_height = {}
display_color_capable = {*grayscale*}

The b rule of the algorithm does not apply since those conversions are not *select* conversions. Finally observe that the *select* conversion is not in the list of prospective conversions since the rule a of the matching process fails because the *visual_frame_width* and *visual_frame_height* properties of the *finish*'s preconditions exist in the *select*'s postconditions, but it is intersection is an empty set. This configuration indicates that the *select* conversion is not capable of producing the properties demanded by the preconditions of the *finish* conversion.

According the step 6 of the Algorithm 2 each valid conversion becomes a new goal conversion and we repeat Algorithm 2 recursively. With respect to *conversion_b* as a new goal, *image_resizer* and *image_greyscaler* bypass the filter of the matching process of Algorithm 1 and the list of prospective conversions is made up of those conversions. The a rule of Algorithm 2 is bypass by *image_greyscaler* (labelled *conversion_d*) because *visual_color_domain* property results in an empty intersection, i.e., *conversion_d* contribute with that property to the adaptation. However the a rule of Algorithm 2 is not bypass by *image_resizer* because all the properties in the intersection have values, i.e., *image_resizer* does not contribute to the adaptation.

If we repeat the Algorithm 2 again with *conversion_d* as a new goal, *select*, *image_resizer* and *image_greyscaler* bypass the filter of the matching process of Algorithm 1 and the list of prospective conversions is made up of those conversions. However the a rule of Algorithm 2 is not bypass by *image_resizer* and *image_greyscaler*, and only the select conversion remains in the list of valid conversions. According to rule 5 of Algorithm 2, we have found a successful path, and we come to an end of the search in this path.



finish (*conversion_a*)
preconditions:
visual_format = {*bmp*}
visual_frame_width = {320}
visual_frame_height = {240}
display_color_capable = {*false*}
postconditions:
target parameters:
visual_format = {*bmp*}
visual_frame_width = {320}
visual_frame_height = {240}
display_color_capable = {*false*}

image_resizer (*conversion_b*)
preconditions:
media_content = {*image*}
visual_format = {*bmp*}
visual_frame_width = *
visual_frame_height = *
visual_color_domain={*grayscale*}
postconditions:
media_content = {*image*}
visual_format = {*bmp*}
visual_frame_width = *
visual_frame_height = *
target parameters:
media_content = {*image*}
visual_format = {*bmp*}
visual_frame_width = {320}
visual_frame_height = {240}

image_greyscaler (*conversion_c*)
preconditions:
media_content = {*image*}
visual_format = {*bmp*}
visual_frame_width = {320}
visual_frame_height = {240}
postconditions:
media_content = {*image*}
visual_format = {*bmp*}
visual_color_domain={*grayscale*}
target parameters:
media_content = {*image*}
visual_format = {*bmp*}
visual_color_domain = {*grayscale*}

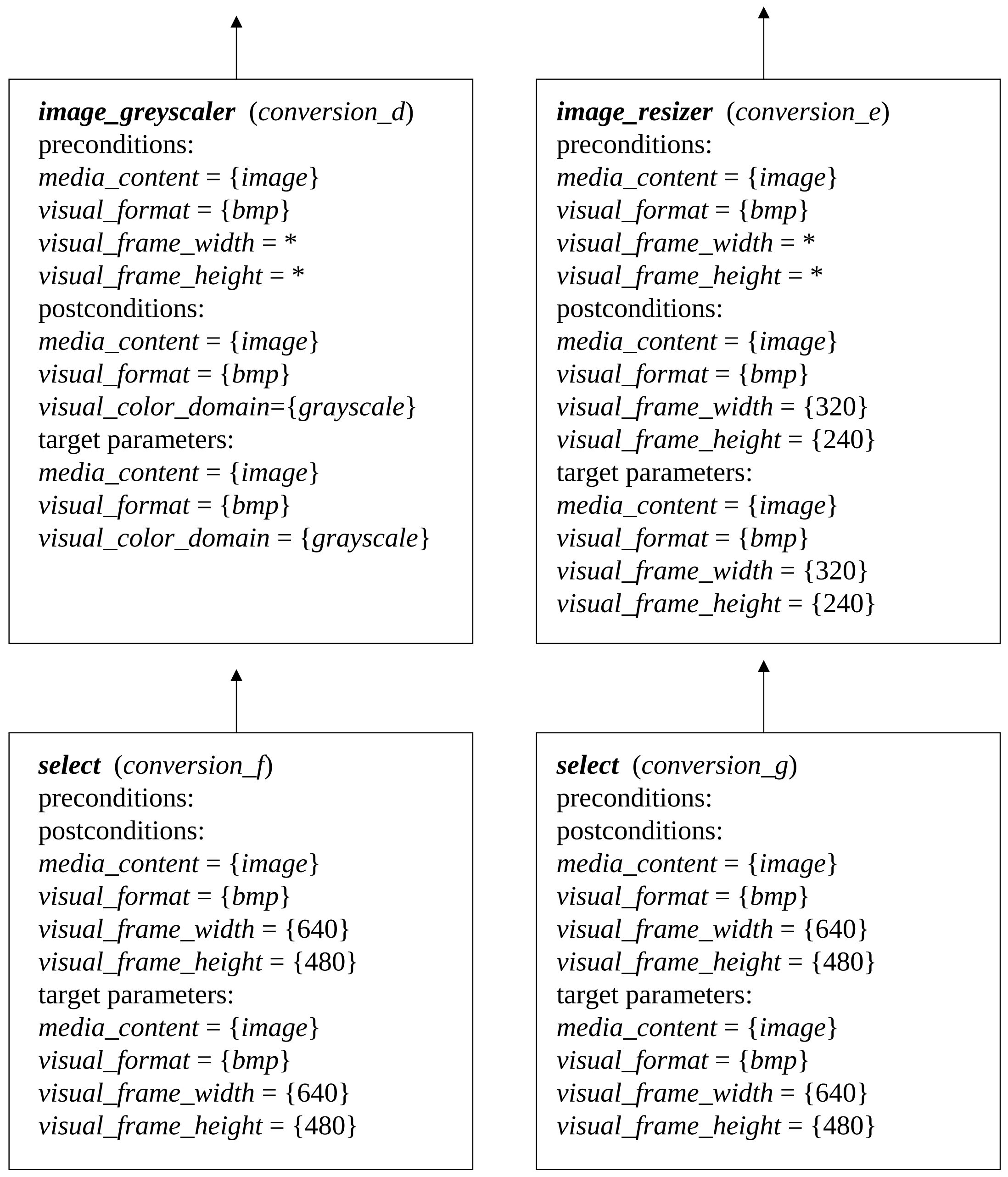

image_greyscaler (*conversion_d*)
preconditions:
media_content = {*image*}
visual_format = {*bmp*}
visual_frame_width = *
visual_frame_height = *
postconditions:
media_content = {*image*}
visual_format = {*bmp*}
visual_color_domain={*grayscale*}
target parameters:
media_content = {*image*}
visual_format = {*bmp*}
visual_color_domain = {*grayscale*}
image_resizer (*conversion_e*)
preconditions:
media_content = {*image*}
visual_format = {*bmp*}
visual_frame_width = *
visual_frame_height = *
postconditions:
media_content = {*image*}
visual_format = {*bmp*}
visual_frame_width = {320}
visual_frame_height = {240}
target parameters:
media_content = {*image*}
visual_format = {*bmp*}
visual_frame_width = {320}
visual_frame_height = {240}
select (*conversion_f*)
preconditions:
postconditions:
media_content = {*image*}
visual_format = {*bmp*}
visual_frame_width = {640}
visual_frame_height = {480}
target parameters:
media_content = {*image*}
visual_format = {*bmp*}
visual_frame_width = {640}
visual_frame_height = {480}
select (*conversion_g*)
preconditions:
postconditions:
media_content = {*image*}
visual_format = {*bmp*}
visual_frame_width = {640}
visual_frame_height = {480}
target parameters:
media_content = {*image*}
visual_format = {*bmp*}
visual_frame_width = {640}
visual_frame_height = {480}

Figure 1: Use case 1: Tree of conversions

In reference to *conversion_c* as a new goal, the process is symmetric and, as shown in Figure 1, we conclude the tree construction with another *select* conversion.

Use case 2

In the previous use case the UED is a unique state because there is only one value for each property. In addition the previous use case has only one initial state. In this use case we are going to consider a *Component* with two variations: therefore there exist two possible initial states. Let us suppose that only one *format* property exists and that we have one *Component* that convey a video with two variations: One of them in *mpeg*-1 format and the other one in *mpeg*-2 format. Suppose also that we have three different CATs with only one conversion per CAT: (1) *VideoCAT* with the *video_transcoder* conversion, (2) *ImageCAT* with the *image_transcoder* conversion and (3) *Video2SlideshowCAT* with the *video2slideshow* conversion. Specifically, the preconditions and postconditions of the conversions are respectively:

video_transcoder

preconditions:

format = {*mpeg*-1,*mpeg*-2,*mpeg*-4}

postconditions:

format = {*mpeg*-1,*mpeg*-2,*mpeg*-4}

image_transcoder

preconditions:

format = {*gif*,*jpeg*,*jpeg*2000}

postconditions:

format = {*gif*,*jpeg*,*jpeg*2000}

video2slideshow

preconditions:

format = {*mpeg*-1}

postconditions:

format = {*jpeg*}

In addition we have a terminal that accepts two resource formats: images with *jpeg2000* format and videos with *mpeg*-4 format.

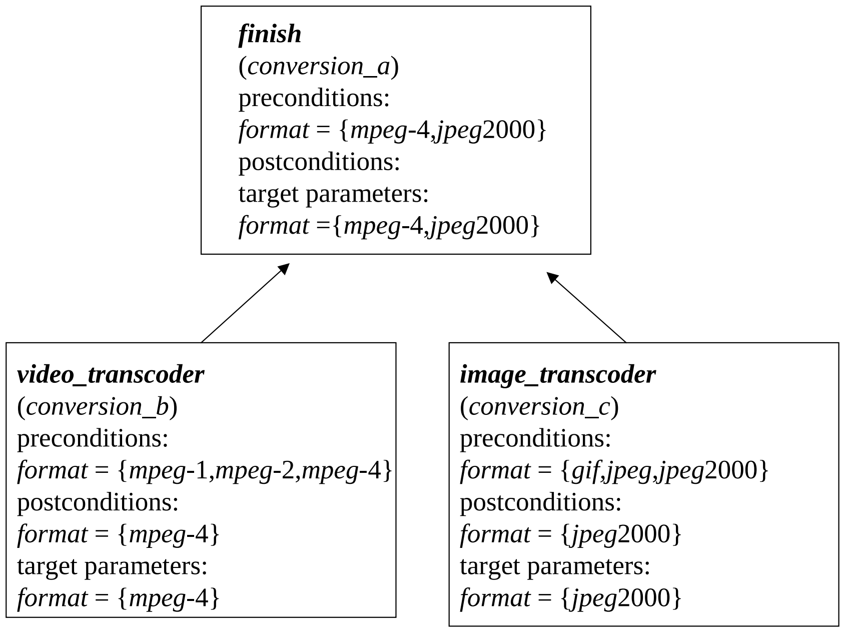
Figure 2 shows the way in which the Algorithm 2 constructs the tree of conversions with the *finish* conversion as the root of the tree. Similarly to use case 1, the *finish* and *select* conversion are constructed, but in this case *finish* is a multiple potential state and there exist two *select* conversions (the first one for the *mpeg*-1 variation and the other one for *mpeg*-2 variation).

The second step of Algorithm 2 proposes to build a list of prospective conversions where their postconditions match with the goal conversion target parameters according to the Algorithm 1. In particular, as shown in Figure 2, the list of prospective conversions is made up of two conversions: *video_transcoder* (labelled *conversion_b* on Figure 2) and *image_transcoder* (labelled *conversion_c* on Figure 2). In this case both prospective conversions become valid conversions. In particular, the a rule of the algorithm does not apply because (1) the intersection between the property list of the postconditions of the *video_transcoder* conversion and the property list of the target parameters of the *finish* conversion is a set with empty properties (*format* = {*mpeg*-4}) and (2) the intersection between the property list of the postconditions of the *image_transcoder* conversion and the property list of the target parameters of the *finish* conversion is a set with empty properties (*format* = {*jpeg*2000}). The b rule of the algorithm does not apply since those conversions are not *select* conversions. Finally observe that the *select* conversion and the *video2slideshow* conversion are not in the list of prospective conversions since in both cases the rule a of the matching process fails because the *format* property of the *finish*'s preconditions exist in the *select*'s postconditions, but it is intersection is an empty set. This configuration indicates that both the select conversion and the *video2slideshow* conversion are not capable of producing the properties demanded by the preconditions of the *finish* conversion.

According the step 6 of the Algorithm 2 each valid conversion becomes a new goal conversion and we repeat Algorithm 2 recursively. With respect to *conversion_b* as a new goal, the list of prospective conversion (according to the matching process of Algorithm 1) are: *video_transcoder* and the *select* conversion for both variations (*mpeg-1* and *mpeg-2*), but the rule a of Algorithm 2 holds and the *video_transcoder* is discarded. Thus, the list of valid conversion includes only the *select* conversion for both variations. Because both *select* conversion fulfil the rule 5 of Algorithm 2, we have found two successful paths and the search process terminate in these paths.

In reference to *conversion_c* as a new goal, the matching process produces a list of prospective conversions with the *image_transcoder* conversion and *video2slideshow* conversion, but the step b of Algorithm 2 removes the *image_transcoder*. Again the Algorithm 2 is repeated with *video2slideshow* as a new goal, and three conversions are added to the list of prospective conversions by the matching algorithm: *video2slideshow*, *video_transcoder* y *select*. The step b of Algorithm 2 removes the *video2slideshow* conversion and only *conversion_g* and *conversion_h* remain. Since *conversion_g* is a select conversion we finish the search in this path.

Finally *conversion_h* has three prospective conversions: *video_transcoder*, and *select* with both variations. Again step b of Algorithm 2 removes *video_transcoder* and *select* with *mpeg*-1, because they do not contribute to the adaptation, hence only *select* with *mpeg*-2 remains, and since it is a *select* conversion the tree expansion ends.



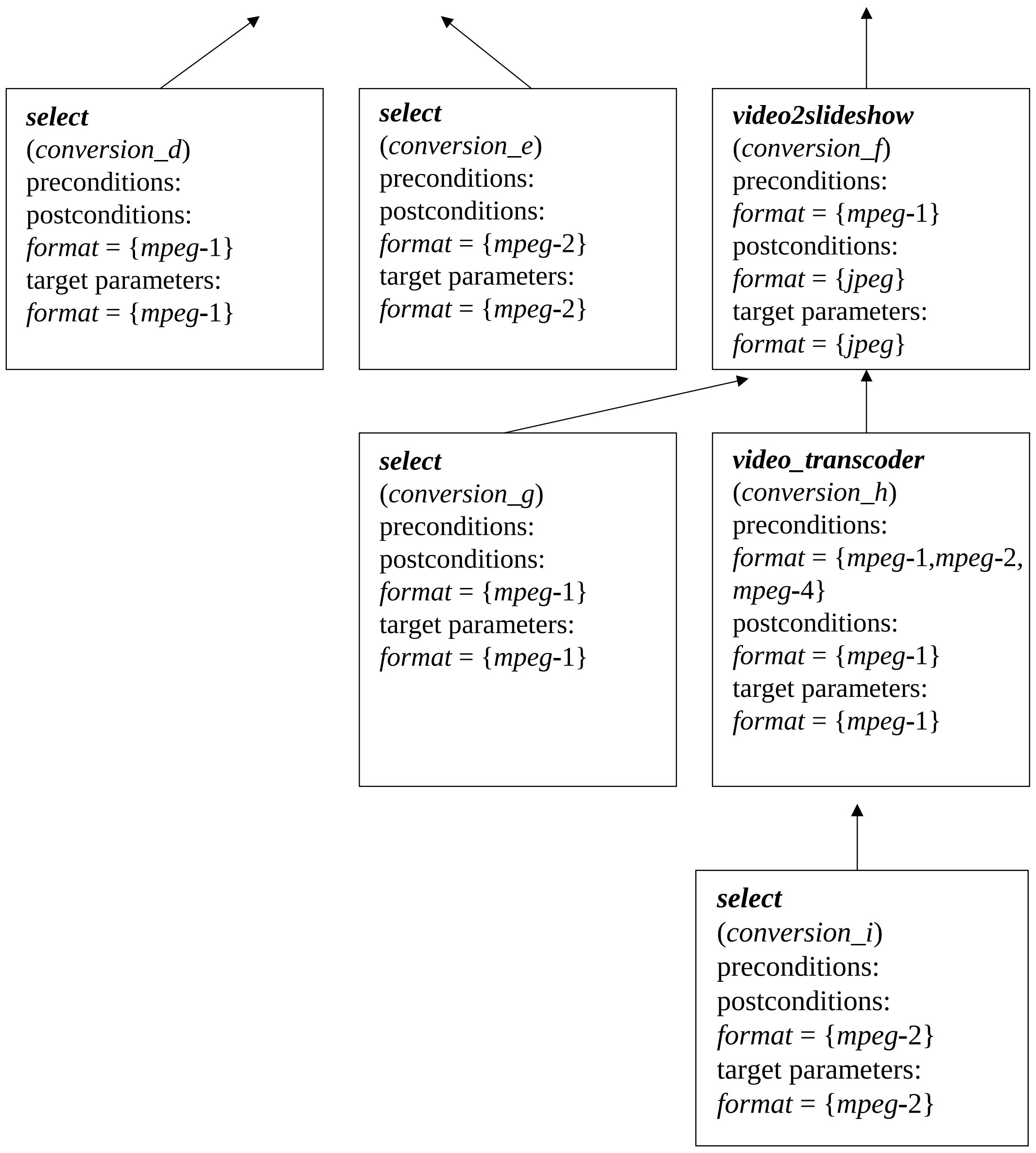


Figure 2: Use case 2: Tree of conversions

References

[1] D. Jannach, K. Leopold, C. Timmerer, Hermann Hellwagner, "A Knowledge-Based Framework For Multimedia Adaptation", Applied Intelligence, vol. 24, no. 2, 2006, pp. 109-125.