Виртуальная машина низкоуровневой обработки видеоизображений в Акторном Прологе

А.А. Морозов[1], О. С. Сушкова[1]

[1]Институт радиотехники и электроники им. В.А. Котельникова РАН, Моховая 11-7, Москва, Россия, 125009

Аннотация. Разработана и реализована виртуальная машина для низкоуровневой обработки видео в объектно-ориентированном логическом языке Акторный Пролог. Принцип работы этой машины заключается в следующем: (1) Машина хранит последовательность команд низкоуровневой обработки видеоизображений, которые должны быть последовательно применены к каждому кадру видеоизображения. Загрузка требуемых команд в машину низкоуровневой видеообработки производится с помощью предикатов встроенного класса *VideoProcessingMachine*. (2) Машина хранит внутренние массивы данных, соответствующие различным этапам низкоуровневой обработки видео. В настоящее время реализованы следующие этапы обработки: предварительная обработка изображений, обработка изображений в пиксельном представлении, выделение и обработка пикселей переднего плана, выделение и трассировка блобов. (3) Машина поддерживает стек массивов пикселей переднего плана, благодаря которому может выделять в каждом кадре различные группы блобов с помощью разных методов обработки видео и выделения пикселей переднего плана. (4) Результатом обработки каждого кадра видеоизображения является список блобов заданных типов, выделенных в кадре, или списки (графы), содержащие информацию о траекториях перемещений этих блобов в течение заданного интервала времени.

1. Введение

Первые исследовательские проекты по применению логических языков для обработки и анализа изображений возникли несколько десятилетий назад, когда появились достаточно быстрые реализации языка Пролог, и возникла техническая возможность соединить Пролог с аппаратными средствами и (или) программными библиотеками для обработки изображений [1, 2, 3]. Можно перечислить несколько простых принципов, которые делают логическое программирование привлекательным средством для обработки изображений, и которые были реализованы в различных комбинациях в разных проектах [1, 2, 4, 5, 6, 7, 8, 9, 10]:

(i) Формулы логики первого порядка (то есть, формулы, содержащие предметные переменные) удобны для декларативного описания пространственных и временных отношений между элементами распознаваемых изображений.

(ii) Логические формулы позволяют легко создавать описания графических объектов и сцен на основе композиции более простых формул. Получаемые таким образом декларативные описания могут быть достаточно сложными, но при этом они всегда остаются принципиально доступными для понимания человеком.

(iii) Логические формулы могут быть выведены средствами индуктивного логического программирования [11], и этот принцип может быть использован для решения различных задач в области машинного обучения.

Одной из фундаментальных проблем, так или иначе проявляющейся при попытке реализовать эти принципы на практике, является противоречие между переборным характером методов логического вывода и необходимостью эффективного представления и обработки графических данных в компьютере. Например, механизм отката (бэктрэкинга), применяемый в логических языках и являющийся одним из самых простых и эффективных механизмов логического вывода, предполагает восстановление состояния переменных и составных структур данных программы в ходе её отката. Применительно к графическим изображениям, это означает сохранение в памяти копий обрабатываемых изображений на всех этапах обработки, чтобы логическая программа с помощью отката могла вернуться к какому-либо пройденному этапу обработки изображения и пойти по иному пути исполнения.

В разных проектах по логическому программированию обработки изображений названная проблема решается по-разному. Самым простым решением является, очевидно, отказ от сохранения копий изображений в памяти в ходе логического вывода. Именно этот подход был применён в одном из первых проектов [2]. Заметим, что отказ от восстановления состояния изображений при откате программы вовсе не обязательно означает нарушение декларативной семантики логического языка. Дело в том, что команды (встроенные предикаты), изменяющие изображение, могут рассматриваться как всегда истинные утверждения, а изменение изображений как побочный эффект, возникающий при проверке этих утверждений (при исполнении этих предикатов). Такой подход, назовём его добавлением *внелогических операций* в логический язык, может быть эффективным с точки реализации, однако неизбежно вызывает следующие вопросы:

- Если в логический язык введены средства, позволяющие прочитать (а не только изменить) содержимое обрабатываемого изображения (например, яркость и цвет пикселей), и ход исполнения логической программы как-то зависит от промежуточного состояния обрабатываемого изображения, что является вполне естественным, декларативная семантика логической программы неизбежно будет нарушена.
- Если же в логическом языке не предусмотрена возможность проверки состояния обрабатываемого изображения, то такой язык вряд ли является практически полезным.

Возможность анализировать изображения, применяя поиск с откатом, очевидно, является главным достоинством соединения логического программирования с обработкой изображений, и отказ от такой возможности делает применяемый подход бессмысленным. В частности, исходя из этих соображений, разработчики упомянутого выше проекта [2] через несколько лет пришли к идее отката состояния обрабатываемых изображений [12, 3].

Те же самые проблемы, но в ещё более острой форме возникли позже перед разработчиками логических систем интеллектуального видеонаблюдения [13]. Основным стимулом для разработки таких систем послужила возможность использования логических правил для компактного и понятного для человека декларативного описания сложных ситуация и событий, таких как попытка обмануть систему контроля доступа в охраняемое здание, кража и др. Проблема логической обработки изображений осложняется тем, что осуществление интеллектуального видеонаблюдения, как правило, требует обработки последовательности изображений в реальном времени (в темпе поступления видеоданных с камер).

Интересно, что усложнение задачи логической обработки изображений послужило толчком для отказа от наивных подходов к решению проблемы и появлению более сложных, но при этом более реалистичных решений. Необходимость обработки видео в реальном времени естественным образом привела к разделению задачи анализа изображений на

подзадачи низкоуровневого анализа и подзадачи высокоуровневого анализа. В частности, в системе [13] вся предварительная обработка видеоизображения, включая вычитание фона и трассировку блобов, была выведена за пределы логического анализа. При этом результаты низкоуровневой обработки видео преобразуются в факты (формулы) логической программы и анализируются с применением встроенного в язык Пролог механизма поиска с откатом.

Похожий подход был применён в проекте логического программирования интеллектуального видеонаблюдения на основе языка Акторный Пролог [14, 15, 16, 17, 18, 19, 20, 21]. В рамках этого подхода различаются низкоуровневый анализ видео (включающий вычитание фона, выделение блобов, трассировку блобов, поиск пересечений траекторий блобов и пр.), реализуемый с помощью встроенных классов языка, запрограммированных на языке Джава, и высокоуровневый анализ (собственно, распознавание заданных ситуаций и событий), реализуемый средствами логического вывода.

Первоначально в языке Акторный Пролог был разработан и реализован встроенный класс *ImageSubtractor*, решающий основные задачи низкоуровневого анализа, такие как:

(i) Предобработка видеокадров, включая двумерную гауссову фильтрацию, двумерную ранговую фильтрацию, вычитание фона с помощью простой гауссовой модели фона.

(ii) Выделение блобов и оценка координат и скорости блобов в различные моменты времени.

(iii) Вычисление траекторий движения блобов. Траектории блобов автоматически разделяются на сегменты, при этом точки пересечения, соединения и разделения траекторий считаются началом и концом сегментов.

(iv) Распознавание и удаление траекторий неподвижных и медленно перемещающихся объектов. В некоторых приложениях полезно исключать из анализа медленно перемещающиеся объекты для уменьшения размера структур данных, описывающих видеосцену.

(v) Вычисление графов траекторий движения блобов и преобразование их в структуры данных (термы) Акторного Пролога. В создаваемые структуры данных заносится информация о координатах, скорости блобов, а также значения специальных статистических метрик [14, 18], оценивающих плавность движения объектов.

Перечисленные средства класса *ImageSubtractor* удобны для решения простых задач интеллектуального видеонаблюдения, однако применение жёстко заданной последовательности операций низкоуровневого анализа видео, конечно же, не является универсальным и гибким подходом. Опыт использования Акторного Пролога для анализа видео выявил потребность в более универсальных средствах низкоуровневого анализа. В частности, была выявлена необходимость формирования специальных последовательностей операций низкоуровневой обработки видеокадров для разных задач анализа видео. Кроме того, в задачах интеллектуального видеонаблюдения за лабораторными животными возникла потребность работать одновременно с блобами нескольких типов, таких, что для каждого типа блобов применяются разные методы предварительного анализа изображений и детектирования блобов.

Для решения новых задач распознавания был разработан встроенный класс Акторного Пролога *VideoProcessingMachine*, реализующий некоторую виртуальную машину низкоуровневой обработки. Во второй части статьи рассматриваются архитектура и принципы работы этой машины. В третьей части статьи рассматривается пример логической программы, использующей класс *VideoProcessingMachine* для низкоуровневой обработки видео.

2. Архитектура виртуальной машины низкоуровневой обработки видео

Встроенный класс *VideoProcessingMachine* реализует некоторую виртуальную машину для низкоуровневой обработки видео. Принцип работы этой машины заключается в следующем:

(i) Машина хранит последовательность команд низкоуровневой обработки видеоизображений, которые должны быть последовательно применены к каждому кадру видеоизображения. Загрузка требуемых команд в машину низкоуровневой видеообработки производится с помощью предикатов встроенного класса *VideoProcessingMachine*. После загрузки команды автоматически применяются к каждому поступающему кадру видеоизображения до тех пор, пока последовательность команд не будет изменена.

(ii) Машина хранит внутренние массивы данных, соответствующие различным этапам низкоуровневой обработки видео. В настоящее время реализованы следующие этапы обработки: предварительная обработка изображений, обработка изображений в пиксельном представлении, выделение и обработка пикселей переднего плана, выделение и трассировка блобов. Преобразование одних внутренних массивов данных в другие происходи автоматически в ходе обработки каждого кадра.

(iii) Машина поддерживает стек массивов пикселей переднего плана, благодаря которому может выделять в каждом кадре различные группы блобов с помощью разных методов обработки видео и выделения пикселей переднего плана.

(iv) Результатом обработки каждого кадра видеоизображения является список блобов заданных типов, выделенных в кадре, или списки (графы), содержащие информацию о траекториях перемещений этих блобов в течение заданного интервала времени.

Список этапов низкоуровневой обработки, реализованных в классе *VideoProcessingMachine*, а также перечень соответствующих команд виртуальной машины низкоуровневой видеообработки и внутренних массивов данных приведён в таблице 1.

С точки зрения декларативной семантики логического языка, виртуальная машина является внелогическим средством языка, то есть, упомянутые выше встроенные предикаты класса *VideoProcessingMachine* являются истинными утверждениями и не оказывают непосредственного влияния на ход исполнения программы. Между тем, побочным эффектом исполнения этих предикатов является формирование последовательности команд низкоуровневой обработки видеокадров. Разумеется, такой подход не обеспечивает наличие декларативной семантики у любой логической программы, обрабатывающей видео, однако он позволяет обеспечить математическую строгость отдельных частей логической программы, там, где это действительно необходимо для обеспечения корректности (и полноты) результатов логического анализа. Например, логическая программа может быть разбита на два взаимодействующих параллельных процесса, один из которых будет формировать последовательность команд виртуальной машины низкоуровневого анализа видео, а другой анализировать графы, вычисляемые этой машиной. При этом первый процесс будет обеспечивать корректность и полноту соответствующей ему части логической программы, если рассматривать встроенные предикаты класса *VideoProcessingMachine* как внелогические операции с побочными эффектами, не влияющие на исполнение рассматриваемого процесса. Второй процесс, сам по себе, также будет обеспечивать корректность и полноту результатов анализа поступающих в него графов траекторий движения блобов.

3. Пример логической программы, анализирующей видео

Рассмотрим пример программы на Акторном Прологе, выделяющей в видеопотоке блобы заданных типов. Предположим, что программа должна создать диалоговое окно и показать в нём последовательность кадров видеоизображения, выделяя при этом блобы трёх

Таблица 1. Архитектура виртуальной машины низкоуровневой обработки видеокадров.

Этап обработки	Примеры команд виртуальной машины	Внутренние массивы данных
Предварительная обработка изображений	Выделение фрагмента изображения, изменение размера изображения, гауссова фильтрация и др.	Изображение (тип Джавы *BufferedImage*)
Обработка изображений в пиксельном представлении	Выбор канала для обработки в пространствах RGB, HSB или режима оттенков серого, сглаживание, вычисление градиентов, нормализация и др.	Матрица пикселей (тип данных Джавы *int[][]*)
Выделение и обработка пикселей переднего плана	Втолкнуть новую маску пикселей в стек, вытолкнуть элемент стека, добавить в маску пиксели, отвечающие условию, исключить из маски пиксели, отвечающие условию, вычесть фон с помощью заданного алгоритма, ранговая фильтрация, сузить маску пикселей, расширить маску пикселей и др.	Маска пикселей переднего плана (тип данных Джавы *boolean[][]*)
Выделение и трассировка блобов	Выделить блобы с помощью заданного алгоритма, заполнить блобы, отобрать блобы, отвечающие заданному условию, трассировать блобы, вычислить цветовые гистограммы блобов и др.	Списки координат и других атрибутов блобов
Дополнительная обработка траекторий блобов	Выбрать трэки, отвечающие заданному условию, и др.	Список трэков блобов

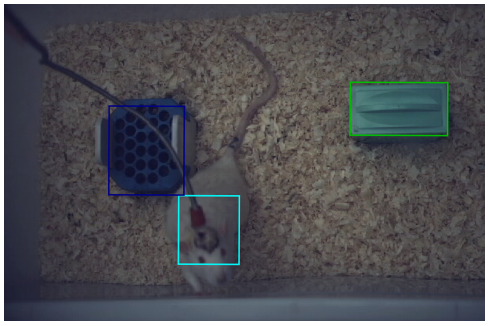

Рисунок 1. Логическая программа распознала три блоба на видеоизображении: ЭЭГ-шапочку, соединяющую голову лабораторной крысы с кабелем, а также зелёный и синий объекты, помещённые в клетку для проведения когнитивного теста.

заданных типов: ЭЭГ-шапочка на голове у лабораторной крысы, зелёный и синий предметы, помещённые в клетку с животным для проведения когнитивного теста (см. рисунок 1).

Для чтения видеофайла воспользуемся встроенным классом Акторного Пролога *FFmpeg*, реализующим связь с открытой библиотекой FFmpeg. Для этого сделаем главный класс программы *Main* наследником класса *FFmpeg*. Слот *name* содержит имя видеофайла. Кроме

того, в классе *Main* определены вспомогательные слоты *vpm*, *dialog*, *graphic_ window*, *image1* и *image2*:

```
class 'Main' (specialized 'FFmpeg'):
constant:
     name = "time 17-07-10 15-16-42.avi";
internal:
     vpm       = ('VideoProcessingMachine');
     dialog    = ('DemoPanel',
                    graphic_window);
     graphic_window = ('Canvas2D');
     image1    = ('BufferedImage');
     image2    = ('BufferedImage');
```

Слот *vpm* содержит экземпляр класса *VideoProcessingMachine*. Слот *dialog* содержит диалоговое окно, в котором будет демонстрироваться видео. Слот *graphic_ window* содержит графическое окно, в котором будут рисоваться кадры видеоизображения и контуры блобов. Слоты *image1* и *image2* содержат экземпляры класса *BufferedImage*, служащие для хранения и передачи массивов видеоданных.

Целевое утверждение программы *goal* загружает команды в экземпляр класса *VideoProcessingMachine*, потом открывает диалоговое окно и запускает чтение видеофайла:

```
CLAUSES:
goal:-!,
     store_VPM_program,
     dialog ? maximize,
     start.
```

Для обработки видео и выделения блобов использованы следующие команды. Ниже приведена последовательность предикатов класса *VideoProcessingMachine* с объяснением их действия.

```
store_VPM_program:-
     -- Приостановить обработку видеокадров:
     vpm ? suspend_processing,
     -- Удалить все команды виртуальной машины:
     vpm ? retract_all_instructions,
     -- Установить размеры краёв блобов:
     vpm ? blb_set_blob_borders(15,15),
     -- Создать новую маску пикселей переднего плана:
     vpm ? msk_push_foreground,
     -- Выбрать пиксели переднего плана, входящие в
     -- заданный интервал оттенков:
     vpm ? msk_select_foreground(
          'HUE',185,245),
     -- Выбрать пиксели переднего плана, входящие в
     -- заданный интервал насыщенности цвета:
     vpm ? msk_select_foreground(
          'SATURATION',50,255),
     -- Выбрать пиксели переднего плана, входящие в
     -- заданный интервал яркости:
     vpm ? msk_select_foreground(
          'BRIGHTNESS',0,150),
     -- Размыть маску на 2 пикселя:
     vpm ? msk_erode_foreground(2),
     -- Расширить маску на 2 пикселя:
     vpm ? msk_dilate_foreground(2),
```

```
-- Выделить блобы; назвать тип блобов "cap":
vpm ? blb_extract_blobs(
     'cap',
     'TWO_PASS_BLOB_EXTRACTION'),
-- Выделить самый большой блоб:
vpm ? blb_select_superior_blob
     ('FOREGROUND_AREA'),
-- Заполнить пустоты в блобах:
vpm ? blb_fill_blobs,
-- Вытолкнуть маску из стека:
vpm ? msk_pop_foreground,
-- Установить размеры краёв блобов:
vpm ? blb_set_blob_borders(1,1),
-- Создать новую маску пикселей переднего плана:
vpm ? msk_push_foreground,
vpm ? msk_select_foreground(
     'HUE',111,138),
vpm ? msk_select_foreground(
     'SATURATION',50,255),
vpm ? msk_erode_foreground(2),
vpm ? msk_dilate_foreground(2),
-- Выделить блобы:
vpm ? blb_extract_blobs(
     'green_object',
     'TWO_PASS_BLOB_EXTRACTION'),
vpm ? blb_select_superior_blob(
     'FOREGROUND_AREA'),
vpm ? blb_fill_blobs,
-- Вытолкнуть маску из стека:
vpm ? msk_pop_foreground,
-- Создать новую маску пикселей переднего плана:
vpm ? msk_push_foreground,
vpm ? msk_select_foreground(
     'HUE',147,183),
vpm ? msk_select_foreground(
     'SATURATION',103,255),
vpm ? msk_erode_foreground(2),
vpm ? msk_dilate_foreground(2),
-- Выделить блобы:
vpm ? blb_extract_blobs(
     'blue_object',
     'TWO_PASS_BLOB_EXTRACTION'),
vpm ? blb_select_superior_blob(
     'FOREGROUND_AREA'),
vpm ? blb_fill_blobs,
-- Вытолкнуть маску из стека:
vpm ? msk_pop_foreground,
-- Возобновить обработку видеокадров:
vpm ? process_now.
```

При получении каждого нового кадра из видеофайла автоматически вызывается предикат *frame_obtained*. Логическая программа с помощью метода *commit* сообщает встроенному классу *FFmpeg* о том, что намерена обработать текущий кадр, затем с помощью метода *get_recent_image* загружает содержимое кадра в экземпляр класса *image1* и передаёт его в машину низкоуровневой обработки видео *vpm*. Заметим, что при этом

массивы видеоданных всё время остаются внутри экземпляров встроенных классов, что обеспечивает их эффективную обработку.

```
frame_obtained:-
     commit,!,
     get_recent_image(image1),
     vpm ? process_realtime_frame(image1),
     draw_scene.
```

После этого вызывается предикат *draw_ scene*, задачей которого является получение из машины низкоуровневой видеообработки списка выделенных блобов и вывод их на экран. Предикат с помощью метода *commit* сообщает машине низкоуровневой видеообработки о том, что необходимо подготовить структуры данных, описывающих текущий видеокадр, затем с помощью предиката *get_ recent_ image* получает содержимое этого видеокадра, а с помощью предиката *get_ blobs* список блобов, выделенных в ходе низкоуровневой обработки видео. Изображение и список блобов выводятся в графическом окне с помощью предикатов встроенного класса *Canvas2D*:

```
draw_scene:-
     vpm ? commit,
     vpm ? get_recent_image(image2),
     image2 ? get_size_in_pixels(IW,IH),
     vpm ? get_blobs(BlobList),
     graphic_window ? suspend_redrawing,
     graphic_window ? clear,
     graphic_window ?
          draw_image(image2,0,0,1,1),
     draw_blob_list(BlobList,IW,IH),
     graphic_window ? draw_now.
```

Список выделенных блобов разматывается, и каждый блоб отображается на экране в виде прямоугольника.

```
draw_blob_list([Blob|Rest],IW,IH):-
     draw_blob(Blob,IW,IH),
     draw_blob_list(Rest,IW,IH).
draw_blob_list([],_,_).
```

Каждый блоб описывается с помощью недоопределённого множества [22] Акторного Пролога, содержащего информацию о типе, координатах, высоте, ширине блоба и пр.

```
draw_blob(Blob,IW,IH):-
     Blob == { type:Type,x:X0,y:Y0,
               width:W1,height:H1|_},
     X2== (X0 - W1/2) / IW,
     Y2== (Y0 - H1/2) / IH,
     W2== W1 / IW,
     H2== H1 / IH,
     select_blob_colour(Type,Color),
     graphic_window ?
          set_pen({color:Color,lineWidth:3}),
     graphic_window ?
          draw_rectangle(X2,Y2,W2,H2),
     fail.
draw_blob(_,_,_).
```

Предикат *select_ blob_ color* определяет цвет прямоугольников, соответствующих блобам разных типов:

```
select_blob_colour('cap','Cyan').
select_blob_colour('green_object','DkGreen').
select_blob_colour('blue_object','Navy').
```

Приведённый простой пример иллюстрирует оба этапа обработки видео – низкоуровневую и высокоуровневую обработку видеокадров, а также использование виртуальной машины низкоуровневой обработки видео для выделения блобов видеоизображения.

4. Заключение

Разработан новый подход к управлению низкоуровневым анализом видеокадров в системе логического программирования интеллектуального видеонаблюдения. Достоинством этого подхода является возможность формирования специальных последовательностей операций низкоуровневой обработки видеокадров для разных задач анализа видео, а также возможность работать одновременно с блобами нескольких типов, таких, что для каждого типа блобов применяются разные методы предварительного анализа изображений и детектирования блобов. Разработанный подход реализован в виде встроенного класса *VideoProcessingMachine* объектно-ориентированного логического языка Акторный Пролог. Программное обеспечение опубликовано в открытом доступе на сайте [21].

5. Благодарности

Авторы выражают благодарность ИВНД и НФ РАН за предоставленные видеоматериалы. Исследование выполнено при поддержке РФФИ (проект № 16-29-09626-офи-м).

6. Литература

[1] Bell, B. Contour tracking and corner detection in a logic programming environment / B. Bell, L. Pau // IEEE Transactions on Pattern Analysis and Machine Intelligence. — 1990. — Vol. 12(9). — P. 913–917.

[2] Batchelor, B.G. Intelligent Image Processing in Prolog / B.G. Batchelor. — London: Springer-Verlag, 1991.

[3] Jones, A.C. Image processing in Prolog: Getting the paradigm right / A.C. Jones // The ALP Newsletter. — 1995. — Vol. 8(4).

[4]Bell, B. Context knowledge and search control issues in object-oriented Prolog-based image understanding / B. Bell, L. Pau // Pattern Recognition Letters. — 1992. — Vol. 13(4). — P. 279–290.

[5] Jones, A.C. A software architecture for intelligent image processing using Prolog / A.C. Jones, B.G. Batchelor // Proc. Intelligent Robots & Computer Vision XIII. Int. Society for Optical Engineering. — Bellingham WA, 1994. — P. 178–188.

[6] Stafford-Fraser, Q. BrightBoard: A video-augmented environment / Q. Stafford-Fraser, P. Robinson // CHI Electronic Proceedings, 1996.

[7] Esposito, F. Machine learning for intelligent processing of printed documents / F. Esposito, D. Malerba, F.A. Lisi //Journal of Intelligent Information Systems. — 2000. — Vol. 14. — P. 175–198.

[8] Batchelor, B.G. Intelligent Vision Systems for Industry / B.G. Batchelor, P.F. Whelan. — London: Springer, 1997.

[9] Буряк, Д.Ю. Метод автоматизированного конструирования процедур обнаружения объектов по их структурному описанию / Д.Ю. Буряк, Ю.В. Визильтер // Журнал радиоэлектроники. — 2003. — Т. 4. - URL: http://jre.cplire.ru/jre/apr03/1/text.html.

[10]Cacko, A. Evaluating the mutual position of objects on the visual scene using morphological processing and reasoning / A. Cacko, M. Iwanowski // Image Processing & Communications Challenges. Advances in Intelligent Systems and Computing. — Heidelberg: Springer International Publishing. — 2015. —Vol. 313. — P. 13–20.

[11] Drobics, M. FS-FOIL: an inductive learning method for extracting interpretable fuzzy descriptions / M. Drobics, U. Bodenhofer, E.P. Klement // International Journal of Approximate Reasoning. — 2003. — Vol. 32. —P. 131–152.

[12] Jones, A.C. Implementing full backtracking facilities for Prolog-based image processing / A.C. Jones, B.G. Batchelor // Proc. Machine Vision Applications, Architectures and Systems Integration IV. Int. Society for Optical Engineering. — Bellingham WA, 1995.

[13] Shet, V. VidMAP: Video monitoring of activity with Prolog / V. Shet, D. Harwood, L. Davis // AVSS 2005. — IEEE, 2005. — P. 224–229.
[14] Morozov, A.A. Real-time analysis of video by means of the Actor Prolog language / A.A. Morozov, O.S. Sushkova // Computer Optics. — 2016. — Vol. 40(6). — P. 947–957. DOI: 10.18287/2412-6179-2016-40-6-947-957.
[15] Morozov, A.A. Object-oriented logic programming of 3D intelligent video surveillance: The problem statement / A.A. Morozov, O.S. Sushkova, A.F. Polupanov // IEEE 26th International Symposium on Industrial Electronics (ISIE), Edinburgh, United Kingdom. — Washington: IEEE Xplore Digital Library, 2017. — P. 1631–1636. — URL: http://ieeexplore.ieee.org/document/8001491.
[16] Morozov, A.A. Towards the distributed logic programming of intelligent visual surveillance applications / A.A. Morozov, O.S. Sushkova, A.F. Polupanov // Advances in Soft Computing: 15th Mexican International Conference on Artificial Intelligence, MICAI 2016, Cancun, Mexico, Proceedings, Part II. — Cham: Springer International Publishing, 2017. — P. 42–53.
[17] Morozov, A.A. Development of a method for intelligent video monitoring of abnormal behavior of people based on parallel object-oriented logic programming / A.A. Morozov // Pattern Recognition and Image Analysis. — 2015. — Vol. 25(3). — P. 481–492.
[18] Morozov, A.A. Development of the logic programming approach to the intelligent monitoring of anomalous human behaviour / A.A. Morozov, A.F. Polupanov // OGRW2014. — Koblenz: University of Koblenz-Landau. - 2015. — Vol. 5. — P. 82–85.
[19] Morozov, A.A. Intelligent visual surveillance logic programming: Implementation issues / / A.A. Morozov, A.F. Polupanov // CICLOPS-WLPE 2014. — Aachener Informatik Berichte no. AIB-2014-09. — RWTH Aachen University, 2014. — P. 31–45.
[20] Morozov, A.A. Development of concurrent object-oriented logic programming platform for the intelligent monitoring of anomalous human activities / A.A. Morozov, A. Vaish, A.F. Polupanov et al. // BIOSTEC 2014. — Springer. - 2015. - Vol. 511. — P. 82–97.
[21] Morozov, A.A. The intelligent visual surveillance logic programming Web Site / A.A. Morozov, O.S. Sushkova, 2017. - URL: http://www.fullvision.ru.
[22] Morozov, A.A. Actor Prolog: an object-oriented language with the classical declarative semantics / A.A. Morozov // IDL. — Paris, France, 1999. — P. 39–53.

A virtual machine for low-level video processing in Actor Prolog

A.A. Morozov[1], O.S. Sushkova[1]

[1]Kotel'nikov Institute of Radio Engineering and Electronics of RAS, Mokhovaya 11-7, Moscow, Russia, 125009

Abstract. A kind of a virtual machine for low-level video processing in the Actor Prolog object-oriented logic language was developed. The principle of operation of this machine is the following one: (1) The machine keeps a sequence of commands of the low-level video processing. This sequence of commands is to be applied for every frame of the video. The loading of these commands into the machine is performed using predicates of the *VideoProcessingMachine* built-in class. (2) The machine keeps internal data arrays that are related to various sub-stages of the low-level video processing. Now the following sub-stages of the processing are implemented in the machine: pre-processing of the frame; processing of the frame in the pixel representation; selection and processing of the foreground pixels in the frame; extraction and tracing the blobs in the sequence of the frames. (3) The machine supports a stack of masks of foreground pixels. This stack enables processing of different groups of blobs using different methods of image processing. (4) The result of the processing of every frame of the video is a set of graphs that contain information about the movements and other attributes of the blobs in the video scene during given time interval.

Keywords: intelligent video surveillance, intelligent video monitoring, Actor Prolog, low-level video processing, blob extraction, virtual machine, object-oriented logic programming, Prolog to Java translation.