Моделирование исключительных ситуаций в императивных программах в терминах сетей Петри*

Д. В. Леонтьев, Д. С. Одякова, Р. В. Парахин, Д. И. Харитонов†

Институт автоматики и процессов управления ДВО РАН, Владивосток, Россия
†Контактный e-mail: demiurg@dvo.ru

Предложен подход к моделированию обработки исключительных ситуаций в императивных программах. Рассмотрены проблематика использования исключительных ситуаций в программах, общий подход к автоматическому построению моделей программ, описан минимальный набор шаблонов семантических конструкций, необходимый для построения моделей императивных программ. В качестве примера описан процесс моделирования небольшой программы и приведена ее результирующая модель в композициональном виде.

Ключевые слова: язык программирования, автоматическое построение моделей программ, верификация программ, сети Петри.

Библиографическая ссылка: Леонтьев Д.В., Одякова Д.С., Парахин Р.В., Харитонов Д.И. Моделирование исключительных ситуаций в императивных программах в терминах сетей Петри // Вычислительные технологии. 2019. Т. 24, № 6. С. 60–68. DOI: 10.25743/ICT.2019.24.6.008.

Введение

Обработка исключительных ситуаций — известный механизм в различных языках программирования, позволяющий описать реакцию программы на некоторую нестандартную ситуацию, возникшую во время ее выполнения. Причиной возникновения исключительной ситуации являются значения переменных программы, при которых бессмысленно продолжать конкретный участок кода. При этом момент определения значений может отстоять настолько далеко по времени и месту в программе, что исправление ситуации в локальной окрестности кода становится невозможным. Для обработки исключительной ситуации формируются специально выделенный блок кода, который реализует реакцию на ошибочную ситуацию, и контролируемый блок, содержащий потенциально опасный код, которые в совокупности предназначены для изоляции исключительных ситуаций. Суть реакции обычно сводится к подконтрольному завершению одного или нескольких компонентов программы с корректным освобождением всех использованных системных ресурсов и информированию пользователя о невозможности обработки данных.

Первоначально механизм обработки исключений был разработан в языке Lisp в 1960–1970-х гг. Немного позже вышли работы Goodenough [1], в которых на примере

*Title translation and abstract in English can be found on page 68.

языка PL/I (прародитель языка C) описаны основные элементы современного механизма обработки исключений и его поддержки со стороны компилятора. На основе этих идей, а также развития объектно-ориентированной парадигмы обработка исключений была реализована в языках программирования Smalltalk и C++ [2, 3]. Позднее концепции реализации перешли в такие популярные языки, как Java, Python, Ruby [4–6] и многие другие. В настоящее время, пожалуй, все языки программирования общего назначения в той или иной степени предоставляют средства описания и обработки исключительных ситуаций.

Несмотря на развитые механизмы обработки ошибок, которыми вооружен современный разработчик, проблема корректности программного обеспечения остается. Регулярно в программах обнаруживают новые ошибки, которые приводят к серьезным последствиям. Внедрение формальной верификации в процесс разработки ПО поможет частично или полностью автоматизировать процедуру проверки корректности программы. Однако для достижения этой цели необходимо разработать и реализовать ряд подготовительных этапов. В частности, требуется разработка соответствующего математического аппарата и алгоритма генерации формальной модели программы по ее исходному коду для последующего анализа. Существующие подходы построения модели программы с исключениями в основном базируются на построении графа потока управления [7–9], который дополняется соответствующими ветками, описывающими обработку исключений. Также в качестве промежуточного применяют представление программы в виде абстрактного синтаксического дерева [10].

В настоящей работе рассматривается подход к построению модели потока управления программы с исключительными ситуациями в терминах сетей Петри, основанный на работе с абстрактным семантическим графом программы. Отличительной особенностью данного подхода является шаблонный способ построения модели, который позволяет изменять шаблоны, моделирующие конструкции языка программирования без переделки алгоритма генерации графа сети Петри.

1. Метод построения моделей программ

Положим, что для целевого языка программирования построена формальная грамматика, записанная в форме Бэкуса — Наура (БНФ). Тогда по описанию правил грамматики можно построить дерево синтаксического разбора, которое можно считать начальным представлением программы для последующей трансформации в модель. Для автоматической генерации модели использование представления в виде дерева разбора не является эффективным. Во-первых, для описания современного языка программирования требуется грамматика, содержащая значительное количество правил. Например, описание грамматики языка C++, представленное в стандарте языка, содержит сотни правил вывода, записанных в форме, близкой к БНФ. Во-вторых, результат синтаксического разбора, выполненного на основании грамматики языка в БНФ, достаточно громоздкий, отягощен большим количеством промежуточных выводов.

Более подходящей структурой данных для трансформации программы в модель является абстрактное синтаксическое дерево или его обобщенный вариант — абстрактный семантический граф, который имеет более краткое представление и не зависит напрямую от изменений в грамматике языка программирования.

Рассмотрим пример небольшого фрагмента кода, описывающего обработку исключений в нескольких функциях (рис. 1). Для этого фрагмента кода обработки исключе-

```
int *Func1()
{
  Xcls* x=new Xcls;
  try
  {
    x->Act();
    delete x;
    Func2();
  }
  catch(...)
  {
    printf("Error");
  }
}
```

```
int main()
{
  Func1();
}
```

```
void Xcls::Act()
{
  if(rand()%2)
  {
    throw;
  }
}
```

```
void Func2()
{
  Xcls* x=new Xcls;
  try
  {
    x->Act();
  }
  catch(...)
  {
    printf("Error");
  }
  delete x;
}
```

Рис. 1. Фрагмент программы на языке C++

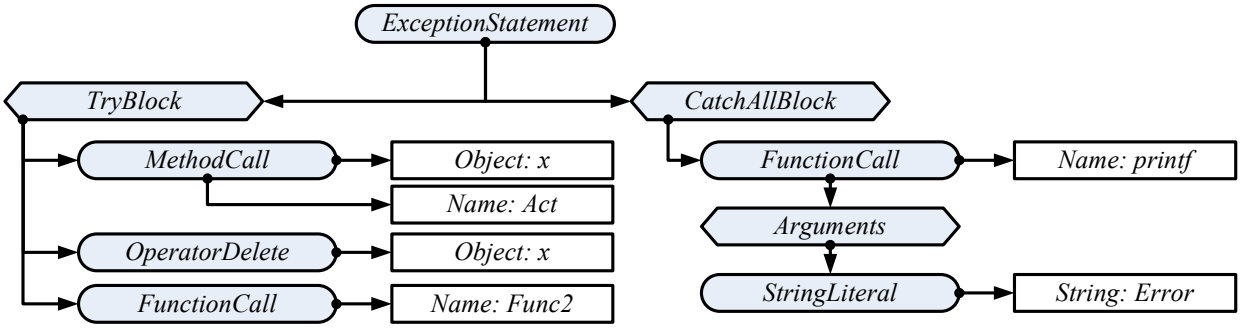

Рис. 2. Фрагмент абстрактного семантического графа для обработки исключения

ний был построен вариант абстрактного семантического графа, частично изображенный на рис. 2. Полученный граф преобразуется в модель программы в терминах сетей Петри достаточно простой подстановкой в узлах графа, соответствующих конструкциям языка шаблонных сетей Петри. Для этого выполняется обход абстрактного семантического графа, во время которого посещение вершины порождает операцию композиции над шаблонными конструкциями. В результате получается композициональная сеть Петри, которая описывает связи между простыми компонентами модели программы. Выполнение всех операций композиции дает общую сеть Петри, которая и будет конечной моделью программы.

2. Шаблоны синтаксических конструкций императивных языков программирования

Модели императивных программ в терминах сетей Петри строятся с использованием шаблонов синтаксических конструкций [11, 12]. На рис. 3 изображен минимальный набор из одиннадцати базовых шаблонов, с помощью которых такая модель может быть построена. При изображении шаблонов используются следующие обозначения. Сеть Петри, описывающая структуру шаблона, помещается в прямоугольник, на границах которого размещаются символические изображения точек доступа по местам в виде окружности и по переходам в виде квадрата. Все входные точки доступа по местам размещаются сверху прямоугольника, а выходные — снизу. Все входные точки доступа

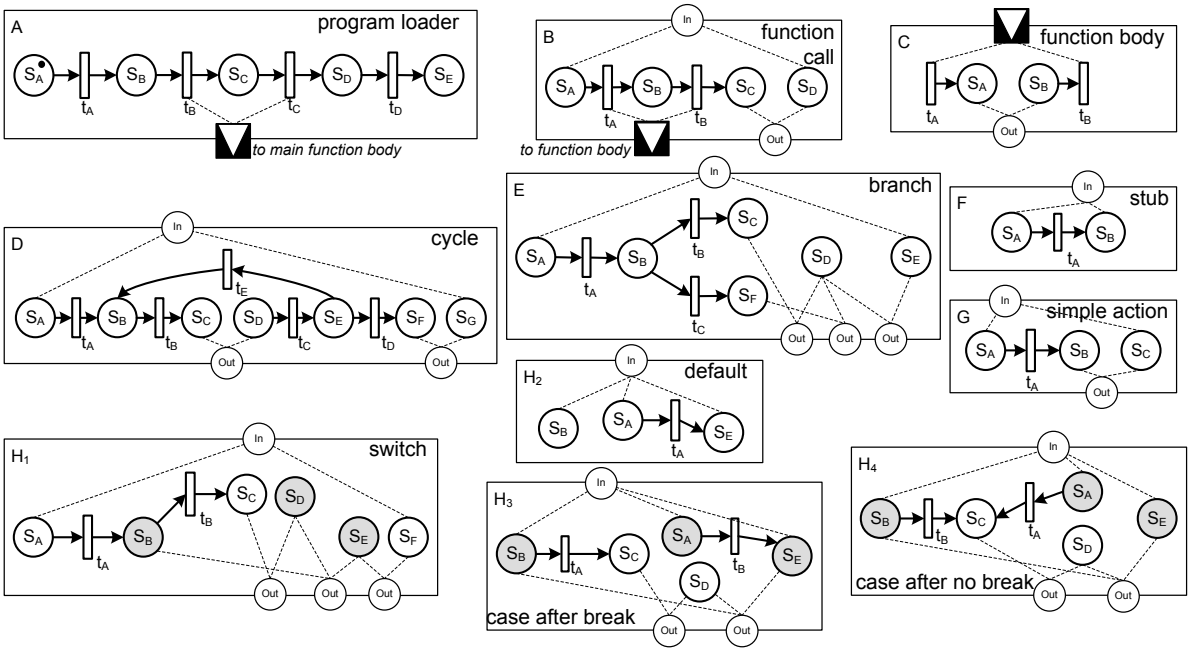

Рис. 3. Шаблоны императивных конструкций

по переходам помечаются треугольником, направленным вершиной внутрь, а выходные — вершиной наружу. Шаблон A называется загрузчиком и моделирует начало и конец последовательного процесса. Место, в котором находится токен, является стартовой точкой модели программы, с которой модель начинает свою работу. Шаблон B моделирует вызов функции в императивной программе, не использующей исключительные ситуации. Первая точка доступа выходного интерфейса шаблона B предназначена для синхронизации с телом функции, вторая — для продолжения программы после вызова функции.

Шаблон C моделирует тело функции. Данный шаблон имеет одну входную точку доступа для синхронизации с вызовом функции и выходную для формирования потока управления. Шаблон D предназначен для моделирования циклов. Первая точка доступа выходного интерфейса этого шаблона используется для моделирования тела цикла. Вторая точка доступа — для продолжения программы после цикла. Шаблон E предназначен для моделирования оператора ветвления. Данный шаблон имеет три точки доступа в выходном интерфейсе, предназначенные для формирования частей программы, соответствующих ветке then, ветке else, и для продолжения программы после оператора ветвления. Шаблон F называется заглушкой. Этот шаблон имеет только входной интерфейс, поэтому после "склейки" с точкой доступа входного интерфейса в результирующей сети будет на одну выходную точку доступа меньше. Шаблон G называется линейным участком и моделирует простое математическое выражение в императивной программе. Шаблоны $H_1 - H_4$ моделируют части конструкции *выбор* императивного языка программирования: основной шаблон H_1 — начало и конец конструкции, завершение как случай `default` H_2, продолжение после оператора `break` H_3 и продолжение без оператора break H_4 соответственно.

Механизм обработки исключений при построении модели описывается тремя дополнительными шаблонами I, J, K, моделирующими конструкции `throw`, `try` и `catch` соответственно, при этом используется нотация цветных сетей Петри. Введем следующие

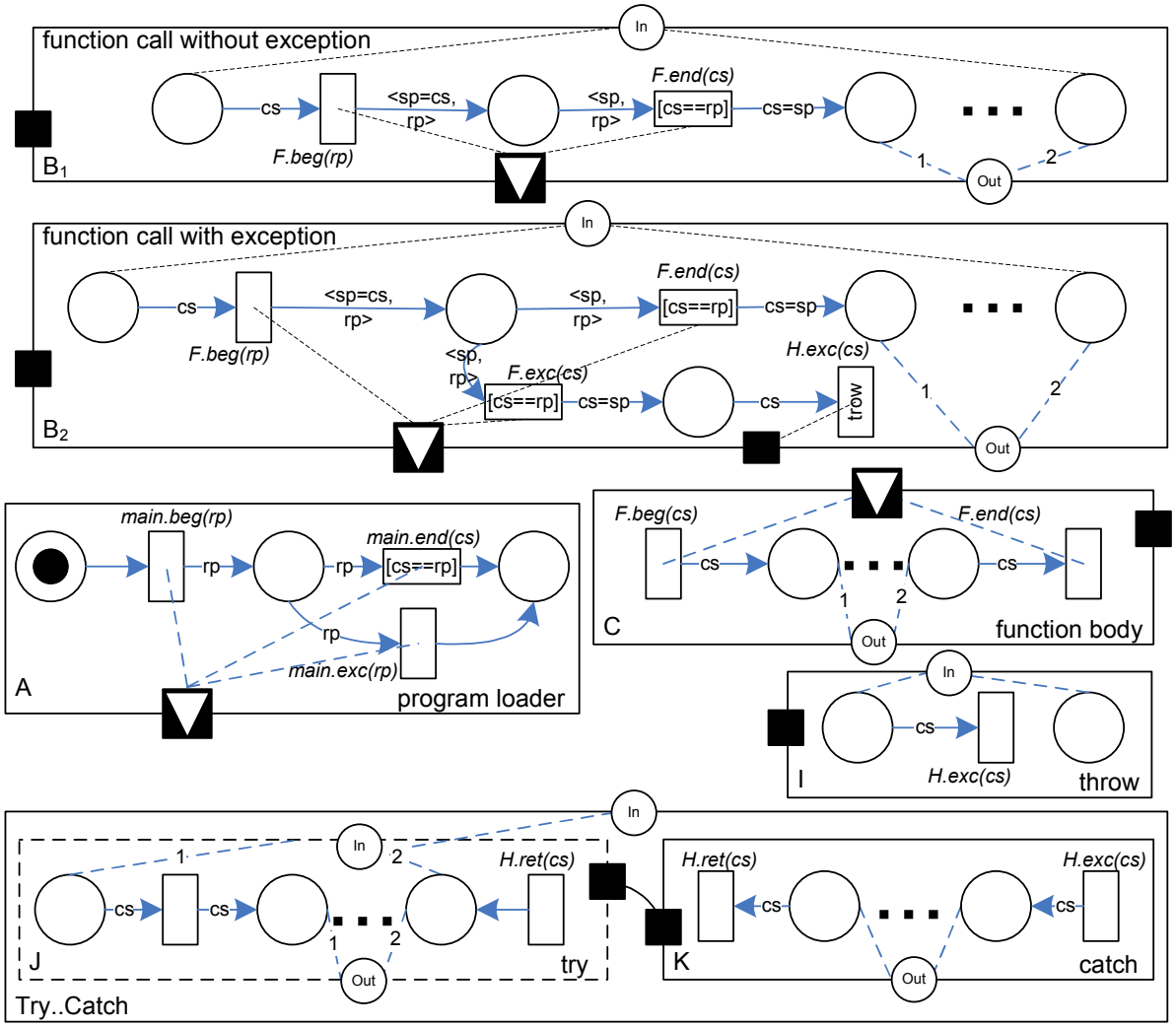

Рис. 4. Шаблоны императивных конструкций для обработки исключительных ситуаций

обозначения: cs (current stack) — текущее состояние стека в данный момент выполнения, rp (return pointer) — адрес возврата из данной функции, sp (save pointer) — копия стека.

Графическое представление шаблонов обработки исключительных ситуаций отображено на рис. 4, где для краткости пропущены шаблоны, не связанные с обработкой исключительных ситуаций. Процедура раскрутки стека моделируется следующим образом. На вызывающей стороне в месте ожидания возврата из функции формируется токен, с которым ассоциируется пара: состояние стека до вызова этой функции и адрес возврата. В процессе ожидания возврата управления из функции возможны два пути: нормальный выход и выход по исключению. В обоих случаях контролируется адрес возврата из функции условиями на переходах — текущий стек из параметризованной точки доступа (пометки F.end(cs) или H.exc(cs)) сравнивается с сохраненным адресом возврата. В случае нормального хода событий состояние стека восстанавливается и вызывающая сторона продолжает свое исполнение. В случае возникновения исключения, в зависимости от места нахождения оператора `throw`, произойдет либо склейка переходов с передачей токена в обработчик `catch`, либо возврат из функции. Способ подстановки шаблонов `try` и `catch` гарантирует, что исключение внутри `try`

будет обработано только внутри `catch`. Исключение же вне конструкции `try` и `catch` передается вызову функции как особый вид возврата из функции (*return by exception*), требующий обработки исключительной ситуации. При этом не обработанный в месте вызова функции *return by exception* вызовет повторную исключительную ситуацию.

3. Пример модели программы

Рассмотрим пример модели программы с исключениями, построенной по исходному тексту программы, фрагмент которой показан на рис. 1. Фрагмент программы состоит из трех функций, одного метода класса и дополнительно сопровождается загрузчиком программы. Полная модель фрагмента программы показана на рис. 5. В композициональной модели на рисунке все связи по точкам доступа по местам, отражающие структурные связи между шаблонами, уже раскрыты, но показаны связи по точкам

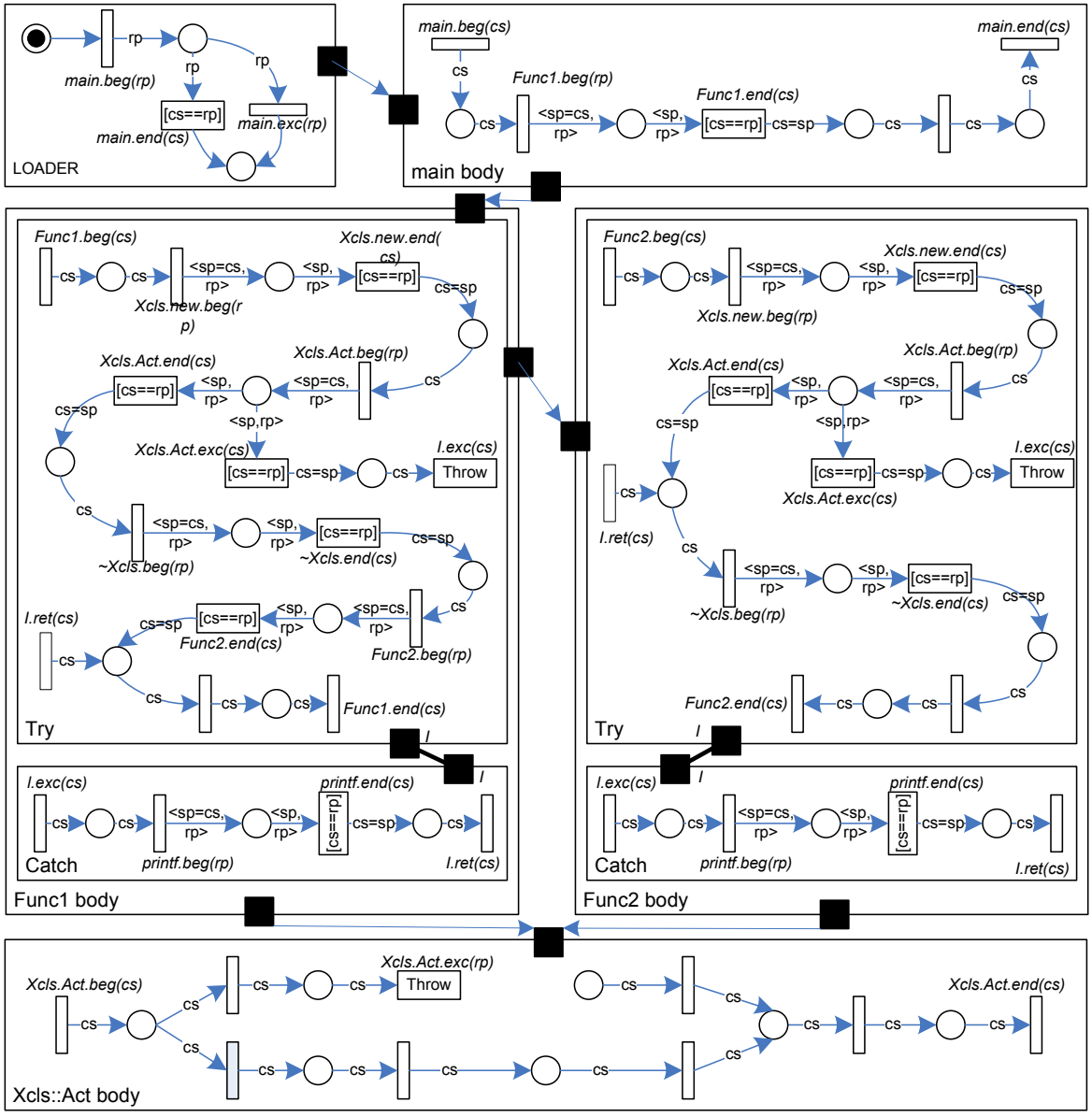

Рис. 5. Композициональная модель программы

доступа по переходам, которые описывают передачу управления между функциями и обработчиками. Поэтому модель включает в себя семь сетей Петри, каждая из которых представляет отдельную единицу программы.

LOADER — это загрузчик программы, который обычно генерируется компилятором. Загрузчик содержит токен в стартовом месте, отражая состояние исполняемого процесса — один токен в загрузчике моделирует единственный поток исполнения. Задача загрузчика в данном примере состоит только в вызове главной функции программы — `main`. Функция `main` состоит из шаблонов тела функции, единственного вызова функции `Func1` и заглушки. Третья сеть примера моделирует тело функции `Func1`. Так как все шаблонные конструкции `try` и `catch` связаны между собой точками доступа по переходам, эта сеть состоит из двух других, моделирующих потоки управления внутри блоков `try` и `catch`. Аналогичным образом из двух сетей состоит и сеть Петри, моделирующая функцию `Func2`, вызываемую из `Func1`. Последней частью композициональной модели является модель метода `Act` класса `Xcls`, который в исходном тексте программы генерирует исключительную ситуацию.

Отметим, что по правилам построения связи шаблонов бывают двух видов: простые — симметричное слияние по пометке, направленные — связь вызова функции и тела функции, когда соответствующие переходы тела функции удаляются после разрешения всех вызовов этой функции. Для всех элементов модели проставлена соответствующая пометка, описывающая действия над стеком.

Заключение

Предложен и описан подход к автоматической генерации моделей императивных программ, использующих ислючительные ситуации, по исходному коду. Подход определяет три последовательных преобразования программы из исходного текста к дереву разбора программы, генерируемому на основе грамматики языка в форме Бэкуса — Наура, далее к абстрактному семантическому графу, который сокращенно представляет программу, и наконец к композициональной модели в терминах сетей Петри.

Продемонстрировано, что предложенный подход позволяет построить полную модель программы с исключениями, состоящую из нескольких функций. Результирующая модель программы дает возможность анализировать поведение программы стандартными для сетей Петри методами. В частности, может быть проверено, существует ли возможность аварийного завершения из-за исключительной ситуации, где обрабатывается каждая конкретная исключительная ситуация и какие исключительные ситуации обрабатываются в конкретном блоке `catch`.

Благодарности. Работа выполнена при финансовой поддержке госбюджетного задания № АААА-А17-117040450019-8.

Список литературы / References

[1] **Goodenough, J.B.** Exception handling: Issues and a proposed notation // Commun. ACM. 1975. Vol. 18, No. 12. P. 683–696.

[2] **Goldberg, A., Robson, D.** Smalltalk-80: the language and its implementation. Boston: Addison-Wesley Longman Publ. Co., 1983. 714 p.

[3] **Stroustrup, B.** The C++ programming language. 4th edition. Boston: Addison-Wesley Professional, 2013. 1376 p.

[4] The Java language specification, Java SE 8 edition / J. Gosling, B. Joy, G.L. Steele, G. Bracha, A. Buckley. Boston: Addison-Wesley Professional, 2014. 685 p.

[5] The Python language reference. Available at: `https://docs.python.org/3/reference/` (accessed 01.10.2019).

[6] **Flanagan, D., Matsumoto, Y.** The ruby programming language. Sebastopol: O'Reilly, 2008. 448 p.

[7] **Mayer, W., Stumptner, M., Wotawa, F.** Debugging program exceptions // Fourteenth Intern. Workshop on Principles of Diagnosis (DX-03). Washington: Institute of Software Technology, 2003. P. 119–124.

[8] **Amighi, A., de C. Gomes, P., Gurov, D., Huisman, M.** Sound control-flow graph extraction for Java programs with exceptions // Proc. of the 10th Intern. Conf. on Software Engineering and Formal Methods. Berlin: Springer-Verlag, 2012. P. 33–47.

[9] **Kastrinis, G., Smaragdakis, Y.** Efficient and effective handling of exceptions in Java points-to analysis // Proc. of the 22Nd Intern. Conf. on Compiler Construction. Berlin: Springer-Verlag, 2013. P. 41–60.

[10] **Robillard, M.P., Murphy, G.C.** Designing robust Java programs with exceptions // Proc. of the 8th ACM SIGSOFT Intern. Symp. on Foundations of Software Engineering. New York: ACM Press, 2000. Vol. 25, No. 6. P. 2–10.

[11] **Тарасов Г.В., Харитонов Д.И., Голенков Е.А.** Об одном представлении функции в модели императивной программы, заданной сетями Петри // Моделирование и анализ информ. систем. 2011. Т. 18, № 2. С. 18–38.
Tarasov, G.V., Kharitonov, D.I., Golenkov, E.A. On a function representation in an imperative program model specified by Petri Nets // Modeling and Analysis of Inform. Systems. 2011. Vol. 18, No. 2. P. 18–38. (In Russ.)

[12] **Kharitonov, D., Tarasov, G.** Modeling function calls in program control flow in terms of Petri Nets // Advances in Computer Sci. 2014. Vol. 3, No. 6. C. 82–91.

Поступила в редакцию 3 октября 2019 г.

Exceptions modelling in imperative programs in terms of Petri nets

Leontev, Denis V., Odyakova, Daria S., Parakhin, Roman V., Kharitonov, Dmitry I.*

Institute of Automation And Control Processes FEB RAS, Vladivostok, 690041, Russia
*Corresponding author: Kharitonov, Dmitry I., e-mail: `demiurg@dvo.ru`

The purpose of the article is to propose an approach to the automatic generation of models of imperative programs with exceptions from the source code.

Methodology. The approach defines consecutive transformations of the program beginning from the source code to the parsing tree of the program, then to an abstract semantic graph and finally to a compositional model in terms of Petri nets. Transformations are based on a set of formal principles and relations and can be performed without

human intervention purely algorithmically. To build a model from the program abstract semantic graph, templates and composition rules are used. Templates describe in terms of Petri net the basic constructions of imperative programming languages: expressions, branching, loops, choice and function call.

Findings. A set of templates for modelling the exception handling mechanism is described. This set includes templates for the try and catch blocks describing the processing of the exception in local places of the program, the throw operator to signal the exception, and the operator of the function call with exceptions.

Originality/value. The article demonstrates that the proposed set of templates allows building a complete model of the program with exceptions, consisting of several functions. The resulting program model makes it possible to analyze the program behavior by standard for Petri nets formal methods. In particular, a possibility of an abnormal termination due to an exceptional situation can be validated and where each particular exception is handled as well as what exceptions are handled in a particular catch block.

Keywords: programming languages, automatic construction of program models, verification of programs, Petri nets.

Cite: Leontev, D.V., Odyakova, D.S., Parakhin, R.V., Kharitonov, D.I. Exceptions modelling in imperative programs in terms of Petri nets // Computational Technologies. 2019. Vol. 24, No. 6. P. 60–68. (In Russ.) DOI: 10.25743/ICT.2019.24.6.008.

Acknowledgements. The research was supported by state funded program No. AAAA-A17-117040450019-8.

Received October 3, 2019