

# گزارش نهایی پروژه درس «روش‌های صوری»

## Detecting Redundant Preconditions

خلاصه، پیاده‌سازی و شبیه‌سازی

مهدی مالوردی\*

### چکیده

این گزارش مقاله‌ی Detecting Redundant Preconditions را خلاصه می‌کند و یک پیاده‌سازی/شبیه‌سازی آموزشی برای بررسی افزونگی پیش‌شرط‌ها در یک دامنه‌ی محدود ارائه می‌دهد. هدف، مشاهده‌ی تفاوت روش‌های مبتنی بر Implication و Dependency با روش «حذف پیش‌شرط و بررسی دوباره» در سناریوهای ساده است. **کلمات کلیدی:** قراردادهای پیش‌شرط، افزونگی، Design by Contract، روش‌های صوری

### ۱ خلاصه مقاله مرجع

**عنوان:** Detecting Redundant Preconditions  
**نویسندگان:** Nicola Thoben, Heike Wehrheim  
**منبع:** FormaliSE 2025 (IEEE/ACM)  
**DOI:** 10.1109/FORMALISE66629.2025.00015

#### ۱.۱ هدف و دامنه

مقاله روی «کیفیت قراردادهای» تمرکز دارد، نه روی تولید قرارداد (contract synthesis) و نه روی اعتبارسنجی صرف. فرض مقاله این است که برنامه‌ها نسبت به قرارداد فعلی صحیح هستند و

---

Student number: 404443150\*

سؤال اصلی این است که آیا بخشی از پیش‌شرط‌ها واقعاً برای برقرار شدن پس‌شرط لازم هستند یا خیر. نکته‌ی مهم مقاله این است که در برخی سناریوها—even اگر یک پیش‌شرط از نظر صحت پس‌شرط «لازم» نباشد—ممکن است از نظر مستندسازی یا تعریف دامنه‌ی ورودی‌های معتبر، نگه داشتن آن مطلوب باشد؛ با این حال کشف افزونگی می‌تواند قرارداد را ساده‌تر، خوانا‌تر و قابل‌استفاده‌تر کند.

### ۲.۱ پیش‌زمینه مفهومی

**قراردادها در Design by Contract** قرارداد تابع/ماژول معمولاً با Preconditions و Postconditions بیان می‌شود. در مدل مقاله، پیش‌شرط‌ها و پس‌شرط‌ها به ترتیب به صورت assume و assert در ابتدای/انتهای برنامه (یا تابع بسته) قرار داده می‌شوند تا ابزارهای تحلیل/اثبات بتوانند آن‌ها را بررسی کنند.

**نمودار جریان کنترل و تحلیل گزاره‌ای** برای تحلیل، مقاله از مفاهیم رایج در راستی‌آزمایی نرم‌افزار استفاده می‌کند: نمایش برنامه به صورت CFA (نمودار جریان کنترل)، توسعه‌ی آن با یال‌های assume/assert و مدل کردن نقض assert با رسیدن به مکان خطا. همچنین از predi-cate analysis و CEGAR (پالایش مبتنی بر ضدنمونه) برای ساخت ARG (گراف دسترسی‌پذیری انتزاعی) استفاده می‌شود.

### ۳.۱ تعریف رسمی افزونگی

مقاله افزونگی را بر اساس «درستی برنامه نسبت به قرارداد» تعریف می‌کند:

• **افزونگی تکی:** پیش‌شرط  $pre\_i$  در مجموعه‌ی Pre زائد است اگر با حذف آن همچنان برنامه پس‌شرط‌ها را برقرار کند.

• **افزونگی گروهی:** یک مجموعه از پیش‌شرط‌ها وقتی زائد گروهی است که بتوان همه‌ی آن‌ها را همزمان حذف کرد و قرارداد همچنان برقرار بماند.

این دو مفهوم مهم‌اند چون ممکن است هر پیش‌شرط به تنهایی «قابل حذف» باشد، ولی حذف همزمان چند پیش‌شرط باعث شکستن قرارداد شود (وابستگی بین پیش‌شرط‌ها/نقش‌شان در محدود کردن دامنه‌ی حالات اولیه).

### ۴.۱ رویکرد ساده اما پرهزینه (Baseline)

تعریف افزونگی مستقیماً یک روش ساده پیشنهاد می‌کند: برای هر پیش‌شرط (یا مجموعه‌ای از آن‌ها) یک بار آن را حذف کنیم و دوباره با یک راستی‌آزما بررسی کنیم که آیا پس‌شرط‌ها هنوز اثبات

می‌شوند یا نه. مشکل: اجرای چندباره‌ی ابزارهای اثبات/راستی‌آزمایی برای هر برنامه بسیار پرهزینه است، بنابراین مقاله دنبال تکنیک‌های کارتر است.

## ۵.۱ تکنیک‌های پیشنهادی برای کشف افزونگی

مقاله سه تکنیک با توازن متفاوت بین دقت و هزینه ارائه می‌دهد:

**۱) IC – بررسی استنتاج منطقی (Implication Checking)** در این روش خودِ پیش‌شرط‌ها با هم مقایسه می‌شوند و برنامه/پس‌شرط به صورت مستقیم وارد تحلیل نمی‌شود. ایده: اگر یک پیش‌شرط از ترکیب سایر پیش‌شرط‌ها نتیجه شود، در حضور آن‌ها اضافه است. مقاله همچنین توضیح می‌دهد که افزونگی تکی لزوماً به افزونگی گروهی تبدیل نمی‌شود؛ برای همین یک رویه‌ی تکراری برای بررسی حذف گروهی پیش‌شرط‌های «قابل نتیجه‌گیری» ارائه می‌کند.

**۲) DC – تحلیل وابستگی (Dependency Calculation)** در این روش، برنامه به صورت CFA توسعه‌یافته (با یال‌های assume/assert و مکان خطا) مدل می‌شود و سپس وابستگی‌های داده/کنترل بین پیش‌شرط‌ها و پس‌شرط‌ها بررسی می‌شود. اگر یک پیش‌شرط به هیچ پس‌شرطی (از منظر وابستگی نحوی) وابسته نباشد، می‌توان آن را زائد فرض کرد. این روش سریع است، اما چون صرفاً وابستگی‌های نحوی را می‌بیند، ممکن است پیش‌شرطی که وابستگی نحوی دارد ولی از نظر معنایی لازم نیست را زائد تشخیص ندهد.

**۳) PC – تحلیل گزاره‌ای/انتزاعی (Predicate Checking)** این روش یک بار تحلیل گزاره‌ای را اجرا می‌کند تا ARG ساخته شود. ایده‌ی کلیدی: اگر در یال متناظر با یک پیش‌شرط ((assume(b)) گزاره‌ی انتزاعی قبل و بعد از آن تغییری نکند (یعنی اضافه شدن آن پیش‌شرط در محاسبه‌ی انتزاع اثر نگذارد)، آن پیش‌شرط در این اثبات «لازم» نبوده و می‌تواند زائد باشد. مقاله برای نسخه‌ی مبتنی بر ARG یک قضیه‌ی صحت (soundness) بیان می‌کند: اگر عبور از یال assume موجب تقویت predicate نشود، جایگزینی آن با assume(true) همچنان شواهد کافی برای صحت قرارداد را حفظ می‌کند.

## ۶.۱ ارزیابی تجربی مقاله

**داده‌ها و بنچمارک** برای ارزیابی، نویسندگان مجموعه‌ای از ۶۲ برنامه C را انتخاب می‌کنند (از منابع مختلف) و ابتدا با CPAChecker بررسی می‌کنند که نسبت به قراردادهای موجود صحیح هستند. سپس با سه نوع تبدیل، پیش‌شرط‌های زائد تزریق می‌کنند:

• **Type ۱: (Independency)** معرفی متغیر/قید اضافی که در صحت پس‌شرط نقشی ندارد.

- **Type ۲: (Implication)** ساخت قیدی که توسط یک پیش شرط لازم دیگر نتیجه می شود (با کمک حل کننده ی Eldarica).
- **Type ۳: (Range)** افزودن کران «جهت مخالف» برای متغیری که یک کران لازم دارد (مثلاً اگر  $x > 0$  لازم است،  $x < 100$  اضافه می شود) تا قید بازه ای بسازد که از نظر صحت پس شرط زائد است.

**فرضیه ها** مقاله سه فرضیه مطرح می کند:

- **۱H (اثر بخشی):** IC و DC قابل مقایسه ی ساده نیستند و PC باید از هر دو بهتر باشد.
- **۲H (کامل بودن):** انتظار می رود PC (در صورت اتمام CEGAR) کامل باشد، اما نتایج تجربی خلاف این را نشان می دهد.
- **۳H (کارایی):** IC/DC بسیار سبک تر از PC هستند.

**نتیجه های اصلی** در آزمایش های مقاله، PC بیشترین تعداد پیش شرط های زائد را تشخیص می دهد (به خصوص در نوع Range) ولی از نظر کامل بودن در پیاده سازی CPAchecker بی نقص نیست. همچنین PC از نظر زمان/حافظه پرهزینه تر از روش های مبتنی بر CFA است، در حالی که IC و DC بسیار سریع ترند.

## ۷.۱ جمع بندی و کارهای آینده (طبق مقاله)

مقاله سه تکنیک عملی برای کشف افزونگی پیش شرط ها ارائه می دهد و نشان می دهد بین سرعت و دقت یک trade-off جدی وجود دارد. به عنوان کار آینده، توسعه ی روش هایی برای کشف «روابط علی» بین پیش شرط ها و پس شرط ها و همچنین استفاده از ACSL برای نوشتن قراردادها پیشنهاد می شود. در بسته ی پروژه، فایل PDF مقاله مرجع نیز پیوست شده است تا خواننده در صورت نیاز به جزئیات کامل، مستقیماً به متن اصلی مراجعه کند.

## ۸.۱ مسئله

در «Design» by «Contract» قراردادها معمولاً با مجموعه ای از پیش شرط ها (Preconditions) و پس شرط ها (Postconditions) نوشته می شوند. هدف مقاله بررسی کیفیت قراردادهاست: آیا برخی پیش شرط ها زائد هستند (یعنی حتی بدون آن ها هم پس شرط ها برقرار می مانند)؟

## ۹.۱ تعریف افزونگی (Redundancy)

یک پیش شرط  $pre\_i$  در مجموعه  $Pre$  زائد است اگر برنامه با حذف آن پیش شرط، همچنان قرارداد را ارضا کند (به صورت  $P \models (Pre \{pre\_i\}, Post)$ ). مقاله علاوه بر افزونگی تکی، مفهوم **افزونگی گروهی** (group-redundant) را هم تعریف می‌کند.

## ۱۰.۱ روش‌های پیشنهادی مقاله

مقاله سه تکنیک برای کشف پیش شرط‌های زائد ارائه می‌کند:

۱. **(Implication Checking) IC**: بررسی می‌کند آیا  $pre\_i$  به صورت منطقی از بقیه‌ی پیش شرط‌ها نتیجه می‌شود یا نه.

۲. **(Dependency Calculation) DC**: روی CFA توسعه یافته، وابستگی‌های داده/کنترل بین  $assume$ ها و  $assert$ ها را بررسی می‌کند؛ اگر پیش شرط به هیچ پس شرطی وابسته نباشد، زائد است (روش نحوی/سینتکتیک).

۳. **(Predicate Checking) PC**: با  $predicate\ analysis$  و CEGAR و ساخت ARG بررسی می‌کند آیا در یال  $(b, assume)$   $predicate$  بعد از آن تغییر/تقویت می‌شود یا نه؛ اگر تقویت نشود، آن پیش شرط زائد است.

## ۱۱.۱ نتایج تجربی مقاله (خیلی خلاصه)

روی ۶۲ برنامه C و ۱۵۵ پیش شرط زائد تزریق شده:

• IC: 61/155

• DC: 62/155

• PC: 147/155

همچنین PC منابع بیشتری (زمان/حافظه) مصرف می‌کند.

## ۲ نوآوری این پروژه (افزونگی گروهی با شاهد نقض)

• **ایده/نوآوری**: علاوه بر تشخیص افزونگی تکی، یک ماژول **افزونگی گروهی** اضافه شده است که وقتی مجموعه‌ای از پیش شرط‌ها به صورت تکی زائد هستند اما حذف همزمان آن‌ها قرارداد را می‌شکند، یک **شاهد نقض** (ورودی مشخص) تولید می‌کند.

- **چرایی ارزشمند بودن:** این کار، خروجی را از یک برچسب ساده‌ی «زائد/غیرزائد» به یک نتیجه‌ی explainable تبدیل می‌کند و برای ارائه/دییاک قراردادها بسیار قابل استفاده است.
- **جزئیات پیاده‌سازی:** ابزار این موارد را محاسبه می‌کند:
  - مجموعه‌ی پیش‌شرط‌های **single-redundant**
  - یک مجموعه‌ی **group-redundant** (با حذف حریصانه)،
  - و در صورت عدم افزونگی گروهی، **input counterexample**.
- **مثال ارائه‌ای: (Counterexample)** در فایل `examples/group_redundancy.js` هر سه پیش‌شرط به صورت تکی زائد هستند، اما حذف همزمان همه‌ی آن‌ها قرارداد را می‌شکند و ابزار یک شاهد نقض مانند  $a=-2, b=-2$  تولید می‌کند. خروجی: `outputs/group_redundancy_report.json`.

**(خلاصه) خروجی نمونه‌ی**

```
single_redundant_indices: [0, 1, 2]
all_single_is_group_redundant: false
counterexample_if_not_group: {"a": -2, "b": -2}
```

**۳ پیاده‌سازی و نتایج اجرای کد****۱.۳ توضیح پیاده‌سازی**

فایل `fm_project/redundancy_checker.py` یک ابزار ساده است که:

- یک برنامه‌ی کوچک را از روی فایل JSON می‌خواند (شامل `pre`, `post` و بدنه‌ی برنامه با `assign/while/if`).
- ورودی‌ها را در یک بازه‌ی محدود (`bounded domain`) شمارش (`enumerate`) می‌کند.
- صحت قرارداد را در این دامنه بررسی می‌کند.
- برای هر پیش‌شرط، با حذف آن و اجرای دوباره، افزونگی تکی را بررسی می‌کند.
- یک بررسی IC-like هم انجام می‌دهد: آیا پیش‌شرط از بقیه‌ی پیش‌شرط‌ها (در همان دامنه محدود) نتیجه می‌شود یا نه.

### ۲.۳ نتایج نمونه (Example)

نمونه‌ی `examples/sub.json` نسخه‌ی ساده‌شده‌ی مثال مقاله/اسلاید است (متغیرهای  $N$  و  $M$  و حلقه).  
دستور اجرا:

```
.venv/bin/python main.py --spec examples/sub.json
```

**خروجی اجرا:** خروجی دقیق اجرا در فایل `outputs/run_results.txt` ذخیره شده است. خلاصه‌ی نتیجه روی این مثال (در دامنه‌ی محدود تعریف‌شده در: JSON):

- پیش‌شرط ضروری:  $N \geq 0$
- پیش‌شرط‌های زائد:  $M \geq 0$ ،  $N \geq -100$ ،  $N \leq 1000$
- نکته: در بررسی IC-like، پیش‌شرط  $M \geq 0$  نتیجه نمی‌شود اما با حذفش همچنان قرارداد برقرار است؛ این دقیقاً نمونه‌ای از تفاوت «استنتاج از پیش‌شرط‌ها» با «لازم بودن برای برنامه» است.

### ۳.۳ محدودیت‌ها

این پیاده‌سازی یک **prototype آموزشی** است و به‌جای تحلیل کامل `C` و ابزارهایی مثل `CPAchecker`، از `bounded execution` استفاده می‌کند؛ بنابراین نتیجه‌ها «تضمین کامل برای همه‌ی ورودی‌ها» نیست مگر دامنه ورودی‌ها کامل باشد.

## ۴ شبیه‌سازی (Benchmark)

برای مقایسه‌ی ایده‌های مقاله به‌صورت آموزشی، یک شبیه‌سازی کوچک روی مجموعه‌ای از برنامه‌های مصنوعی انجام شد. در هر برنامه یک پیش‌شرط ضروری و سه پیش‌شرط زائد از سه نوع **Range**، **Implication**، **Independency** تزریق شد و سه روش زیر مقایسه شدند:

- IC-like: بررسی استنتاج منطقی از سایر پیش‌شرط‌ها
- DC-like: تحلیل وابستگی نحوی از متغیرهای پس‌شرط
- VC: حذف پیش‌شرط و بررسی مجدد قرارداد (روش معنایی در دامنه محدود)

این شبیه‌سازی با دستور زیر اجرا شد و خروجی در `outputs/simulation_summary.txt` ذخیره گردید:

VC	DC-like	IC-like	تعداد واقعی	نوع پیش شرط زائد
۳۹	۳۹	۰	۳۹	Independency
۳۹	۰	۳۹	۳۹	Implication
۳۹	۰	۰	۳۹	Range
۱۱۷	۳۹	۳۹	۱۱۷	Total

```
.venv/bin/python main.py --simulate --sim_n=39 --sim_seed=1
```

نتیجه‌ی شبیه‌سازی (۳۹ برنامه، مجموع ۱۱۷ پیش شرط زائد):

## ۵ فرآیند اجرای پروژه

- انتخاب مقاله مرجع و مطالعه (PDF مقاله + اسلاید).
- استخراج تعاریف کلیدی (افزونگی تکی/گروهی، ایده‌ی روش‌های IC/DC/PC).
- پیاپیاده‌سازی یک prototype کوچک برای بازتولید ایده‌ی «حذف پیش شرط و بررسی دوباره» در قالب checking. bounded.
- اجرای ابزار روی مثال‌ها و ثبت نتایج.
- آماده‌سازی گزارش و بسته‌ی نهایی زیپ.

## ۶ کدهای پروژه / گیت‌هاب

- کد پروژه کوچک است و به صورت فایل‌های همین زیپ پیوست می‌شود.
- اگر لازم شد روی گیت‌هاب آپلود شود: (لینک اینجا قرار گیرد)

## ۷ پیوست‌ها

- مقاله مرجع (PDF) و اسلایدها در فایل زیپ نهایی قرار داده می‌شوند.



**منابع**

۱. N. Thoben, H. Wehrheim, "Detecting Redundant Preconditions," FormaliSE 2025, DOI: 10.1109/FORMALISE66629.2025.00015.