PathLP - הוראות הפעלה.

תיאור:

pathlp הינה תוכנית שמקמפלת ומריצה קוד בשפת PathLP. מדובר בשפה בנויה מעל פרולוג טבלאות. השפה מטפלת בגרפים, מחלקות וטיפוסים. היא פותחה ע"י פרופסור מירה בלבן מאוניברסיטת בן גוריון בנגב, ישראל, ופרופסור מיכאל קיפר מאוניברסיטת סטוני ברוק, ניו יורק, ארצות הברית.
התוכנית מוגדרת באמצעות Well-Founded Model, כלומר התשובות לשאילתות יכולות להיות שלושה: false, undefined, true.

התקנה:

Linux או Unix.

יש למלא כתובות במקומות המסומנים בקובץ "pathlp.sh" לפי המחשב המסוים. לאחר מכן ניתן להשתמש בפקודה "pathlp" להרצת התוכנית.
דרישות המערכת: התקנת הגרסה האחרונה של xsb.
כדאי מאוד להחזיק בתוכנית readlink (כמעט בטוח יש לכם).
כדאי להתקין לעבודה נוחה גם את rlwrap. התוכנית יודעת להשלים בעזרת tab את הפרדיקטים, את הפרמטרים של הפרדיקטים לקביעת מצב המערכת, את הטקסט שהוקלד עד כה ואת תוכן מערכת הקבצים.

Windows או DOS.

יש למלא כתובות במקומות המסומנים בקובץ "pathlp.bat" לפי המחשב המסוים. לאחר מכן ניתן להשתמש ב-shortcut "pathlp" להרצת התוכנית מתוך Windows או באותו קובץ "pathlp.bat" מתוך Dos.
דרישות המערכת: התקנת הגרסה האחרונה של xsb.
כמו תמיד ב-Windows כדי להימנע מבעיות, עדיף לא להתחיל את שמות הקבצים או תיקיות באותיות מתוך abfnrtvx.

תוספות.

ניתן למצוא מספר קבצים למעוניינים:

1. קובץ pathlp.png ובו icon עבור Linux או Unix וקובץ pathlp.ico עבור Windows.
2. קובץ pathlp-install.xml ליצירת Mime type חדש ב-Linux.
3. קובץ pathlp.xml לצביעה נוחה של קוד בעבודה במערכת KDE (כמו Kate).
4. קבצים katetest.ppl, katetest.pdf לבדיקת הקובץ הקודם.

<u>עבודה עם התוכנית:</u>

1. במערכת Linux או Unix ניתן ליצור קבצי הרצה. בשורה הראשונה של קובץ PathLP יש לכתוב !#/usr/bin/env pathlp. יש להרשות הרצה לקובץ בעזרת chmod +x <file>. הקובץ pathlp צריך להיות במקום המזוהה על ידי ה-shell. במידה והוא לא, יש להוסיף בסוף הקובץ tcshrc./ (או tcshrc./~, cshrc./, cshrc./~) שורה
setenv PATH ${PATH}:<address of PathLP>
2. אין אפשרות לקרוא לעותק נוסף של PathLP מתוך עצמו.
3. בהתחלת עבודת התוכנית יופיע prompt. במידה והתוכנית לא תצליח לזהות את רצועת הזמן המדויקת, השעה לא תודפס.
4. הקלט ייקרא כשורה בודדת.
5. בכדי להמשיך בשורה הבאה יש לשים \ כסימן האחרון בשורה.
6. בכדי להמשיך ליותר שורות ניתן להשתמש בסימן הנ"ל מספר פעמים. לעתים הדבר לא נוח. הוספת \number\ - מספר בין שני סימני \ - יגרום לתוכנית לקרוא את אותו מספר של השורות הבאות במקשה אחת. בסיום האחרונה שבהן ניתן לשים שוב את אחד הסימנים הללו. ניתן להשתמש בהם לסירוגין. יש להיזהר מרוב משמעיות: אם ברצונכם לכתוב ?x = pred \ 5, help. בשתי שורות, תדאגו לא לכתוב בשורה הראשונה \5\ ?x = pred, כי 5 יובן כמספר שורות ולא כקלט. מספיק רווח לפני סימן ההעברה.
7. התוכנית מאפשרת שורות ריקות בקלט.
8. לחיצת סימן סוף הקובץ באמצע השורה לא תשפיע. הלחיצה בתחילת השורה תסיים את עבודת התוכנית.
9. ניתן להשתמש בסימן ה-! (cut) בצורה הרגילה.
10. כל הקלט שלא משורת הפקודה עובר דרך preprocessor לפני קימפולו. #define תגדיר קבועים או מקרואים עם פרמטרים. #define X 3 יחליף את כל הופעותיו של X (שלא כחלק מהמילה, כמו XYZ) במספר 3, #define same(x) x..x יחליף את הביטוי {same(3)} בביטוי {3..3}.
11. #defeval דומה ל-#define אך יחשב את כל המקרואים הפנימיים בזמן קריאת ההגדרה ולא בזמן השימוש (הגדרה סטטית).
12. #include "filename" יוסיף את הטקסט של הקובץ במיקום הנוכחי.
13. #ifeq X 3 יקמפל את השורות הבאות רק עם הקבוע X אכן שווה לערכו הנקוב.
14. בצורה דומה יעבוד #ifneq, אם לא שווה.
15. #ifdef X יצליח עם הקבוע הוגדר.
16. #ifndef X יצליח עם הקבוע לא הוגדר.
17. #if מצפה בתור ארגומנט ביטוי אריתמטי לפי כללים רגילים. הביטוי נחשב מצליח כאשר שווה לאפס. ניתן לצרף כמה תנאים בעזרת &&, ||, ולשלול בעזרת !. ניתן גם להשתמש בפרדיקט defined(X) לבדיקה נוסח #ifdef.
18. חמשת הפקודות תקפות עד להופעת אחת משלושת הבאות. #else יצליח במידע והפקודה הראשית נכשלה.
19. #elif Y 8 פירושה אחרת, ואם. ניתן להשתמש בה מספר פעמים, ובסוף אפשר להוסיף את #else.
20. #endif מסיים את הפקודה. ניתן להשתמש במבנים מקוננים.
21. #exec פרושו לבצע פקודה בעזרת ה-shell של מערכת ההפעלה. כך לדוגמה, הכנסת #exec echo ":- pwriteln(" `date` ")." תחליף את השורה בפקודת הדפסה של זמן הקמפול.

22. #eval יחשב את הביטוי לאחריו. לדוגמה:

```
:- writeln(\2\
#eval  3 + 4
)@_prolog.
```

ידפיס 7 על המסך.

23. ניתן להשתמש גם בפקודות נוסח #mode בצורה הרגילה של preprocessor-ים למיניהם.
24. ניתן להכניס בקוד הערות בשתי צורות. הערות שמתחילות בסימן % נגמרות בסוף השורה.
25. הערות שמתחילות בעזרת */ תסתיימנה בעזרת /*. מותר לכתוב הערות באמצע הקוד. כשלא מדובר בהערות בתוך קובץ, כולל מהקלט, יש להשתמש בסמלי העברת שורה.
26. השפה תומכת בבניית DCG, אך יש להיזהר ולשים סוגריים מיותרים במידת הצורך.
27. במהלך קימפול הקוד מתבצעות מספר בדיקות. עובדות נבדקות לקיום משתנים גלויים (לא מתחילים בסימן _) שמופיעים רק פעם אחת (singletons). כללים נבדקים גם למשתנים גלויים שמופיעים רק בחלק הראשי (משמאל לסימן :-) (unsafe). ניתן לבטל בדיקות אלו בעזרת warnings/1.
28. כל קטע שיזוהה בתור איבר בשאילתה, term, או יהווה שורה שלמה של קוד, אשר מוסגר בגרשיים הפוכים (` ... `) יועבר אוטומטית לפרולוג, ללא ביצוע parsing. לדוגמה, `:- import append/3 from basics.` . הסימן ? יזוהה בתור משתנה לפי הכללים הרגילים, כמו ?x = 3, ?y is `?x` + 1.
29. במידה וצריך להשתמש בסימן ? יש לכתוב \?.
30. במידה וצריך להשתמש בסימן \ לפני ? כלשהו יש לכתוב \\.
31. ניתן להשתמש בכל סימני infix של פרולוג, לדוגמה, הזנה בשורת הפקודה ?x is (5 + 4.18) * (-8) תגרום לתוצאה המצופה.
32. עובדות, כללים והגבלות ניתן להזין רק בקובץ (כולל בעזרת [_]).
33. שאילתות ניתן להזין בקובץ או ישירות בשורת הפקודה.
34. בשאילתות בקובץ ניתן להשתמש בצורה . ... -? לביצוע שאילתה ללא הדפסת תשובה, או . ... -?? עם הדפסת תשובה. שימוש בצורה . ... :- יגרום לביצוע ללא הדפסה ולאזהרה במידה של כישלון.
35. בשורת הפקודה ניתן להשתמש בסימן -?. סימן -?? ניתן להשמיט כליל. אין להשתמש בסימן :-.
36. בכדי לקרוא לפרדיקט של פרולוג, ניתן להשתמש בסמל @. למשל, הזנה בשורת הפקודה append([1,2], [3,4], ?x)@basics תקרא לפרדיקט הדרוש מהמודול basics.
37. שימוש בשם המודול prolog_ יקרא לפרדיקט מהמרחב הגלובלי, כמו repeat@_prolog.
38. ניתן להשתמש במשתנה בתור שם המודול, אך עליו להיות בנוי ברגע הקריאה.
39. ניתן לכתוב בקובץ פרדיקטים חדשים, הם ניתנים לקריאה בעזרת pred@_prolog.
40. * של השפה, שפירושה אינסוף חיובי, היא מספר לכל דבר (ניתן לשאול *>3).
41. במקרים מסוימים כשתצטרכו להשתמש בפרדיקטים הפנימיים של pathlp בתור פרמטרים לפרדיקטים אחרים, יש לקרוא להם בעזרת פרדיקט plp/1.
42. אסור להגדיר מחדש תכונות ששמן מתחיל בסימן _. עובדה a._b[c] לא תתקמפל. עובדה a!?x[b] בשילוב שאילתה ?x!_y[?z] -? תיפול בזמן ריצה.
43. בצורת הדפסת תשובות אחת אחרי השנייה הפקודה להמשך היא ";" או " ", לסיום "." או "" (שורה ריקה).

44. הסימנים בשימוש השפה ניתנים להפיכה לצורת ה-infix בעזרת הוספת שני רווחים לאחריהם. למשל, השאילתה ?x = a < b תכשל, וכדי ליצור זוג נתונים בתוך המשתנים, יש לכתוב ?x = a < b. לעומת זאת, הוספת רווח לאחר סימן שווה תכשיל את שתי האופציות. הסימנים שמשמעותם שונה הם [. ,: ,= ,=< ,>= ,< ,>]. גם פסיק הינו כזה, אך יש לו תפקידים מיוחדים בשפה ויש להיזהר במיוחד בשימוש בו בצורה הזאת. ניתן להשתמש בפסיק הנ"ל לצבירת פקודות, כפרמטר:

if((?x < ?y, (plp(help), plp(state)), true).

45. אף פרדיקט פנימי של פרולוג לא ייקרא בעזרת גרשיים הפוכים. ניתן להשתמש לטובת כך בפרדיקט plp/1, לדוגמה plp(answers(all)) או plp([[file1,file2]]).
46. קחו בחשבון שהפרדיקטים pwrite/1+, pwriteln/1+ הינם מקרואים. לכן הם לא צריכים plp/1 מחד, אבל מאידך השימוש בהם שלא לצרכי הדפסה יכול להביא לתוצאות בלתי צפויות.
47. במידה וכחלק משורת הפקודה מופיע ביטוי מהצורה <directory name string> cd הוא ימחק אוטומטית מהקלט ותבוצע פקודת שינוי ספריית העבודה, וזאת כדי להקל את העבודה עם מערכת KDE (כגון Kate). שורה שיש בה רק "cd" תשחזר את המצב המקורי.
48. ניתן להשתמש בצורה &<symbol> לקבלת קוד ה-ASCII של התו. למשל, &A יחזיר 65, & (רווח) יחזיר 32.
49. באטומים שמוגדרים בעזרת גרשיים ניתן לרשום &<xyz> כאשר xyz הינו מספר בין 32 לבין 126, והרצף יוחלף בתו עם קוד ASCII תואם. יש לכתוב &\ כשצריך להשתמש ב-&. יש לכתוב ב-\\ כשצריך להשתמש ב-\ .
50. ניתן להשתמש בגרשיים כפולים לקבלת רשימת הקודים של ASCII, כמו בפרולוג. ניתן להשתמש בתו \ בתוך הגרשיים.

<u>הפרדיקטים הפנימיים של השפה:</u>

1. פרדיקט load/1 טוען קובץ שמועבר בתור פרמטר. load(_) יטען את הקובץ מהמקלדת. יש להשתמש בסימן EOF של מערכת ההפעלה שלכם לסיום ההקלדה (Ctrl-D ב-Linux או Unix, Ctrl-Z ולאחריו Enter ב-Windows או Dos). שם עם סיומת ppl.* מתייחס לקובץ PathLP, שיקומפל ויוטען. במידה וממועד הקמפול האחרון לא היו שינויים בקוד, ייטען הקובץ מהקמפול הראשון. סיומת P.* מתייחסת לקובץ Prolog והוא ייטען בעזרת פרדיקט consult/1. במידה ואין לקובץ אחת משתי הסיומות הנ"ל תיבדק האפשרות לקיומו של קובץ ששמו נוצר מהשם הנתון בתוספת סיומת ppl.*. לדוגמה, load(file3) ינסה לטעון קובץ file3.ppl. במידה ולא קיים, תיבדק האפשרות לקיומו של קובץ פרולוג file3.P. במידה ולא קיים, תיבדק האפשרות לקיומו של קובץ פרולוג file3. יש לרשום את שם הקובץ לפי כללי ה-atom, כלומר להכניסו לגרשיים בודדים במידת הצורך. שימו לב כי כל הטענה תתווסף למה שכבר הוטען.
2. קיצור מקובל של הפרדיקט load(name) הוא [name]. ניתן לשים כמה שמות מופרדים בפסיק והם ייטענו משמאל לימין.
3. פרדיקט loadnew/1 דומה בפעולתו לפרדיקט load/1, אבל המידע הקודם יימחק, וגם equality ו-typing יוחזרו למצב ברירת מחדל. לא יחולו שינויים בהטענת קובץ prolog.
4. ניתן להשתמש במקומו במבנה [[file1, file2, ...]]. הנתונים הקודמים יימחקו בכל מקרה לפני תחילת ההטענה.
5. פרדיקט help/0 ידפיס על המסך קובץ עזרה עם אפשרויות עיקריות.

6. לסיום העבודה יש להשתמש בפרדיקט halt/0.
7. פרדיקט resetsystem/0 ימחק את כל המידע שהצטבר עד כה מטעינת קבצים.
8. פרדיקט stable/0 פירושו לבצע את בדיקת יציבות המערכת.
9. פרדיקט warnings/1 קובע את הדפסת ההזהרות בזמן הקימפול. לפי ברירת המחדל ההזהרות מודפסות. warnings(off) יכבה אותם, warnings(on) ידליק אותם, warnings(d) יציב אותם במצב ברירת מחדל וידפיס את הערך שנקבע. קריאה עם משתנה בתור פרמטר לא תשנה את המצב, אלא תציב את הערך הקיים במשתנה.
10. פרדיקט answers/1 דומה בצורת קריאתו לפרדיקט warnings/1. answers(all) יעביר את PathLP למצב בו כל שאילתה מדפיסה את כל התשובות האפשריות (ללא כפילויות). יש להיזהר מכניסה ללולאות אינסופיות. answers(wait), ברירת המחדל, תצפה לתגובת המשתמש בטרם תחליט אם להדפיס את התשובה הבאה, כמקובל בפרולוג.
11. פרדיקט equality/1 דומה גם הוא, ותפקידו להדליק או לכבות את פעולתו של הסימן :=. האפשרויות הן normal (לפי חוקי השפה), או empty (ברירת מחדל), בה הוא עובד כמו כל פרדיקט רגיל בתוספת שוויון, ולא מופעל במקומות אחרים. לדוגמה, שאילתה שתחזיר a.b[c] לא תחזיר a.b[d], גם אם קיימת אחת העובדות c:=:d או d:=:c.
12. פרדיקט typing/1 דומה גם הוא, ותפקידו להדליק ולכבות את תכונת type inference, אשר נותנת לתוכנית אפשרות להסיק את טיפוס המשתנה לפי הנתונים הקיימים. האפשרויות הן inference (ברירת מחדל), checking.
13. פרדיקט stability/1 דומה גם הוא, ותפקידו לקבוע האם תתבצע בדיקת יציבות המערכת לאחר כל הטענת קבצים (לאחר כל קריאה ל-load/1, למשל, אבל רק בסוף הטענת רשימת הקבצים בעזרת [...], וגם בסוף עיבוד כל הפרמטרים בשורת הפקודה). האפשרויות הן automatic (ברירת מחדל), initiated.
14. פרדיקט tracing/1 קובע את הדפסת מחסנית הקריאות בזמן טיפול בשגיאות זמן ריצה. האפשרויות הן no (ברירת מחדל) או trace.
15. פרדיקט state/0 ידפיס את הקונפיגורציה הנוכחית של המערכת.
16. פרדיקט +pwrite/1+ ידפיס את כל הארגומנטים שלו, משמאל לימין.
17. פרדיקט +pwriteln/0+ יעשה את אותו הדבר ובסיום יעביר שורה. ללא ארגומנטים רק תועבר שורה.
18. פרדיקט pislist/1 בודק האם הארגומנט הוא רשימה תקינה – רגילה או רשימת הפרש.
19. פרדיקט plength/2 מצליח כשאורכה של הרשימה בארגומנט הראשון שווה לארגומנט השני. נותן תשובה X undefined כאשר X הוא הגודל המינימלי של הרשימה במקרה של [...|?var].
20. פרדיקט pmember/2 מצליח כאשר הארגומנט הראשון הוא איבר מתוך הרשימה שבארגומנט השני. במידה ואין אפשרות לקבוע, ניתנת התשובה undefined. הפרדיקט ידפיס הזהרה במידה ורשימה כוללת איברים זהים, גם כשהם אותו משתנה, או תנסה לבדוק אם משתנה שייך לרשימה שהוא חלק בה. שלושת הפרדיקטים יודעים לטפל היטב ברשימה אינסופית.
21. פרדיקט AND/2 הוא פרדיקט prefix של and.
22. פרדיקט OR/2 הוא לפרדיקט prefix של or.
23. פרדיקט if/3 מהווה צורה נוחה יותר למבנה cond -> doiftrue ; else של פרולוג, מהצורה if(cond, doiftrue, else).
24. פרדיקט if/2, בדומה לקודמו, תואם למבנה cond -> doiftrue, if(cond, doiftrue), וכמוהו נכשל כשהתנאי לא מתקיים.

25. פרדיקט if/4, בדומה לקודמיו, תואם למבנה if(cond, doiftrue, doifundefined, else). במידה והתוצאה שתתקבל תהיה undefined לפי מושגי ה-3-valued logic יתבצע הארגומנט השלישי, והתוצאה תישאר undefined.
26. ניתן להשתמש בפרדיקטים true, otherwise, false, fail, undefined.

<u>שורת פקודה:</u>

1. ניתן להוסיף פרמטרים לפקודה וכולם יבוצעו לפי הסדר. כל פרמטר שלא יזוהה יתקבל כשם של קובץ ויטען ע"י load/1. במידה וקיים צורך במחיקת המידע לאחר הזנת הקובץ, יש לציין זאת במפורש ע"י הוספת reset- לאחר שמו.
2. הפרמטר stdin- יבצע את הפקודה load (_).
3. הפרמטר help- או h- ידפיס מסך עזרה ע"י help/0. (קריאה עם ארגומנט בודד h- תגרום לסיום ההרצה מיד לאחר ההדפסה).
4. הפרמטר halt- יעצור את פעולת התוכנית. אל תשכחו לבדוק את יציבות המערכת.
5. הפרמטר reset- יבצע את הפקודה resetsystem/0.
6. הפרמטר stab- יבצע את הפקודה stable/0.
7. הפרמטרים won-, woff- יבצעו את הפקודה warnings/1.
8. הפרמטרים aall-, await- יבצעו את הפקודה answers/1.
9. הפרמטרים enormal-, eempty- יבצעו את הפקודה equality/1.
10. הפרמטרים tcheck- ,tinf- יבצעו את הפקודה typing/1.
11. הפרמטרים sinit-, saut- יבצעו את הפקודה stability/1.
12. הפרמטרים btrace-, bno- יבצעו את הפקודה tracing/1.
13. הפרמטר state- יבצע את הפקודה state/0.
14. ניתן להעביר פרמטרים ישר למנוע XSB. ב-Linux או ב-Unix ניתן להגדיר לפני הקריאה (בחלק מה-shell-ים, כמו tcsh ,csh)

```
setenv xsbparam -p
```

או

```
setenv xsbparam '-p -S'
```

ומכאן והלאה הפרמטרים יועברו. לביטול יש להגדיר מחדש פרמטרים אחרים או לקרוא

```
unsetenv xsbparam
```

בשאר ה-shell-ים (כמו sh.distrib ,dash ,bash ,sh) יש להשתמש בהתאם בפקודות:

```
export xsbparam=-p
export xsbparam='-p -S'
unset xsbparam
```

ב-Windows או ב-Dos יש להשתמש בהתאם בפקודות:

```
set xsbparam=-p
set xsbparam=-p -S
set xsbparam=
```

15. במידה וקיימים פרמטרים שנדרשים באופן קבוע, יש להוסיפם בתום הכתובת של XSB בקבצים pathlp.sh או pathlp.bat.

<u>עקרונות השפה:</u>

1. PathLP היא שפת תכנות לוגי המפותחת ע"י מירה בלבן ומיכאל קיפר, בהתבסס על F-Logic. היא נועדה מלכתחילה להיות רמה בסיסית שמעליה תבנה שפת המידול F-OML. כמקובל בתכנות הלוגי, אפשר להתייחס לתוכנית בשפת PathLP כאל בסיס ידע (knowledge base), הכולל נתונים (עובדות), כמו גם כללי היסק ואילוצי נכונות (integrity constraints).
2. מדובר בשפה הבנויה על המושג "***מסלול***" (***path***). תכנית PathLP מתארת גרף מכוון עם תוויות על הצמתים ועל הקשתות וגם מידע על הטיפוסים של התוויות. לכן ישנם שני סוגים של קשתות: קשת ערך וקשת טיפוס, וישנם שני סוגי יחסים: יחס שייכות של ערך לטיפוס, ויחס הכלה בין טיפוסים.
3. **משתנה (variable):** מצוין בשמו, כשלפניו סימן שאלה: "?x" פירושו משתנה בשם x; "?_x" פירושו משתנה בשם _x. משתנה ששמו מתחיל בסימן _ לא מודפס כתשובה לשאילתה (don't care variables). "?_" ו-"?" הם משתנים חד פעמיים, כלומר הופעות חוזרות של השמות הללו מסמנות משתנים שונים כל פעם.
4. **ביטוי מסלול (object) path expression):** המבנה הדקדוקי העיקרי שצורתו $x.y_1[z_1]....y_n$ או $x.y_1[z_1]....y_n[z_n]$, כאשר x הוא התווית של צומת (אובייקט), ממנה יוצאת קשת עם התווית y_1, לצומת שערכה (או התווית שלה) z_1, וממנה יש מסלול עם קשתות y_2, ..., y_n עד לצומת עם תווית z_n. הערך של ביטוי מסלול הוא הצומת האחרונה שלו.
5. הביטוי הכתוב בסוגריים מרובעים נקרא **guard**, והוא יכול להופיע אחרי כל קשת בביטוי מסלול, אך לא בהכרח. אין להציבו לאחר הצומת היוצאת של הביטוי. ביטוי מסלול ללא guard בסופו נקרא unguarded path expression, והוא מתאר קבוצה של כל הצמתים המתאימים הקיימים בבסיס הנתונים. ה-guard יכול להיות מורכב ממספר ביטויי מסלול שאינם guarded, והם מופרדים באמצעות פסיקים. לדוגמא, a.b.c[d.e[?f].g, ?f.h, ?x] מתאר מסלול a.b.c לצומת שהתווית שלה היא הערך של המסלולים d.e.g, d.e.h, ושל המשתנה ?x.
6. **יחס שייכות (membership):** מתואר על ידי הסמל ":". x:y מציין שהערך של x שייך לטיפוס y.
7. **יחס הכלה (subsetting):** מתואר על ידי הסמל "::". x::y מציין שהטיפוס (קבוצה) x מוכל בטיפוס y. זהו יחס טרנזיטיבי.
8. **ביטוי מסלול מסוג טיפוס (type path expression):** בדומה לגרף הערכים של צמתים, ישנו גם גרף של טיפוסי ערכי הקשתות. הביטוי $x!y_1[z_1]!y_2...!y_n[z_n]$ מתאר מסלול מסוג טיפוס, שמשמעו הוא: הטיפוס של הקשת y_1 מ-x הוא z_1, הטיפוס של הקשת y_2 הוא z_2, ..., וכך הלאה עד הטיפוס z_n של הקשת y_n. ביטוי מסלול מסוג טיפוס יכול להגביל גם את מספר הקשתות היוצאות (cardinality constraint). x!y[z]{a..b} מתאר את הטיפוס של הקשת y של הצומת x, ובנוסף קובע שמצומת x יוצאות בין a ל-b קשתות ערך עם תווית y. כל אחד מהגבולות יכול להיות מספר או משתנה. ניתן להשתמש בסימן * (אינסוף) כמספר חוקי בשפה. העדר סוגריים מסולסלים מצביע על {*..0}.
9. ניתן לבנות ביטויי מסלול ארוכים כגון a.b.c[?x].d.e[?y], כלומר לבנות מסלול ולהשמיט guard מכל קדקוד שהוא בגרף. במסלול מסוג טיפוס ארוך ניתן לשים סוגריים מסולסלים רק על האלמנט האחרון.

10. **עובדות וכללים (rules and facts):** מבנה השפה דומה למדי לזה של פרולוג. למשל, בטעינת הקובץ הבא:

```
a.b[c].
a.b[d].
?s:f :- a.b[?s], a.m(?s)[?x], ?x>10.
```

מדובר בשתי עובדות וכלל אחד. השימוש בהם נעשה על ידי הכללים הרגילים של פרולוג.

11. **שאילתה (query):** כמו בפרולוג, זאת פקודה המפעילה את תהליך ה-reasoning כדי לקבל תשובות דרושות. לדוגמא השאילתה:

```
?- (?p = prop4 or ?p = prop11), q!?p[?m], ?m:class1.
```

משמעה: מצא ערכים ל-?p ו-?m כך שמתקיים כי ?p = prop4 או ?p = prop11, וגם טיפוס של קשתות הערך עם תווית ?p היוצאות מ-q הוא ?m, וגם ?m עצמו שייך לטיפוס ?class. בדרך כלל השאילתות נקראות בצורה אינטראקטיבית, בה ניתן להשמיט את -?.

12. ביטוי מורכב (compound term) עם 0 ארגומנטים כגון f() הינו צורת כתיבה נרדפת לביטוי פשוט f.

13. **אילוצים (constraints):** קיים סוג רביעי של מבנים, הנקרא הגבלה או אילוץ. הוא מסומן בעזרת -! ומתאר מצבים אסורים. כלומר, הגבלה המוזנת לבסיס הידע של PathLP כשאילתה, חייבת להיכשל. שאילתות כאלו מתבצעות במסגרת תהליך הנקרא בדיקת יציבות המערכת. שאילתה שנכשלת פירושה הגבלה שמצליחה, ושאילתה שמצליחה פירושה חוסר יציבות במערכת. במקרה זה מודפסות תוצאות השאילתה. לדוגמה, האילוץ -! ?a.b[?x], ?x>10. אוסר קשתות b שיוצאות מ-?a כלשהו ל-?x כלשהו, שערכו גדול מ-10. תהליך בדיקת היציבות מתבצע בדרך כלל אחרי טעינת כל הקבצים. אילוצי קרדינליות נחשבים להגבלות.

14. השפה כוללת אופרטורים לוגיים בשאילתות, כגון: and (או ","), or, not. +\ אינו אופרטור לוגי, אלא fail_if רגיל של פרולוג.

15. השפה כוללת גם שתי קשתות מיוחדות מובנות. השאילתה a._size[?x] מחזירה מספר קשתות כולל שיוצאות מצומת a (נכון לעכשיו ההגדרה שונה מזאת שמופיעה במאמר המתאר את השפה). השאילתה a._size(b)[?x] מחזירה מספר קשתות עם תווית b שיוצאות מצומת a.